CAPACITY CONSIDERATIONS FOR DATA STORAGE IN MEMRISTOR ARRAYS

by

Susanna Rumsey

A thesis submitted in conformity with the requirements for the degree of Master of Applied Science Graduate Department of The Edward S. Rogers Sr. Department of Electrical & Computer Engineering University of Toronto

 \bigodot Copyright 2019 by Susanna Rumsey

Abstract

Capacity Considerations for Data Storage in Memristor Arrays

Susanna Rumsey Master of Applied Science Graduate Department of The Edward S. Rogers Sr. Department of Electrical & Computer Engineering University of Toronto 2019

Memristor arrays are a promising memory technology that store information in a crossbar array of two-terminal devices that can be programmed to patterns of high or low resistance. While extremely compact, this technology suffers from the "sneak-path" problem: certain information patterns cannot be recovered, as multiple low resistances in parallel make a high resistance seem low. In this thesis we review the current state of memristor technology, including proposed solutions to the sneak-path issue. We extend this discussion by proposing some variations on existing encoding schemes involving one-hot encoding that improve reliability and ease of access, taking into account the scaling of peripheral hardware. We also extend the scope of the discussion by deriving the information capacity of models of multi-layer memristor devices, which is higher than that of the same number of single-layer devices stacked.

Acknowledgements

Many thanks to my supervisor Prof. Frank Kschischang and to Prof. Stark Draper for their continued support, insight, and feedback. It is sincerely appreciated.

Contents

1 Background										
	1.1	Introduction								
	1.2	Overvi	iew of physical device development	3						
	1.3	Device	$e $ switching $\ldots \ldots \ldots$	4						
	1.4	Existin	ng hardware-level sneak-path solutions	4						
		1.4.1	Fabrication solutions	5						
		1.4.2	Black-box solutions	6						
		1.4.3	Discussion	7						
	1.5	Outlin	e	8						
2	Сар	pacity of single-layer devices 9								
	2.1	Genera	al description and ideal model	9						
	2.2	Device	e representations $\ldots \ldots 1$	0						
		2.2.1	Matrix representation	0						
		2.2.2	Graph representation	4						
	2.3	Identif	ying and counting equivalent patterns	.5						
		2.3.1	Stirling numbers	.5						
		2.3.2	Number of distinguishable array patterns	.5						
		2.3.3	Proof of Theorem 2.3.1 from graph theory 1	.6						
		2.3.4	Connections to the theory of binary relations	.6						
		2.3.5	Physical array readings	.8						
		2.3.6	Capacity and asymptotics	8						
		2.3.7	Figures of merit	9						
3	Enc	oding	schemes for single-layer devices 2	4						
	3.1	ng encoding scheme 2	24							
		3.1.1	Example	25						
		3.1.2	Comments	27						

		3.1.3 Error probability \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots	28
	3.2	Simple "identity matrix" encoding	29
	3.3	Permuted identity matrix encoding	29
	3.4	Generalized header rows	30
	3.5	One-hot rows	31
		3.5.1 Geometric dependence	33
	3.6	Optimizing one-hot rows with peripherals	33
	3.7	Final notes	34
4	Cap	oacity of multi-layer devices	35
	4.1	Device representation	35
		4.1.1 Matrix representation	35
		4.1.2 Graph representation	37
	4.2	Identifying sneak paths	37
	4.3	Counting data patterns	41
	4.4	Analysis	46
5	Con	clusions and future work	47
	5.1	Single-layer devices	47
	5.2	Multi-layer devices	48

List of Figures

1.1.1 Two representations of an array with sneak-path	2
2.2.1 Two representations of an array with sneak-path	11
2.2.2 Representations of a memristor array	14
2.3.3 Single-layer densities	22
3.1.1 Fraction of bits for two geometry choices	27
3.1.2 Optimal L for existing scheme, with approximations	28
3.3.3 Comparison of schemes discussed	30
3.4.4 Optimal number of header rows	31
3.4.5 Basic header row scheme	32
3.6.6 The values of m and n that maximize $\rho_C(m, n)$ as a function of C	34
4.3.1 Connection pattern example	43
$4.4.2$ Single vs. multi-layer \ldots	46

Chapter 1

Background

1.1 Introduction

Compact data storage systems are a key component of modern computer systems. With advances in materials science and uncertainty about the continued shrinking of the transistor technologies that underlie modern solid-state storage, researchers are increasingly exploring alternative storage technologies. One such technology is the memristor array.

A memristor is a passive two-terminal circuit element whose resistance can be changed when a sufficiently extreme voltage is applied across the terminals (see e.g., Fig. 1 of Pan et al. [1]). The device maintains its new resistance after the applied voltage is removed. Most commonly and most simply, the memristor will change fairly rapidly between a high-resistance state and a low-resistance state, making it a binary device for practical purposes, meaning that a memristor can be used very naturally in computer storage to store a single bit.

To store multiple bits, memristors are arranged at the intersections between two layers of parallel wires, one running horizontally and one vertically. The corresponding circuit diagram for an example array of 16 memristors is shown in Figure 1.1.1 (after [2]). In this figure, black resistors indicate low-resistance, and white resistors high-resistance. The second half of this figure shows the binary representation of the same array, with 1's corresponding to low resistances (high conductivity), and 0's corresponding to high resistances (low conductivity).

This architecture has many benefits. First, it needs only two wires to access each device, compared to the three wires typical to transistor-based architectures such as flash. This results in improvements in density and, depending on the encoding method, possible simplifications in reading and writing schemes. Second, as will be discussed below, for manufacturing technology working at a given scale, it is possible to make



Figure 1.1.1: Two representations of the same 16-memristor array. Black is high-conductance and white is low-conductance. After [2].

memristors smaller (i.e., with dimensions closer to the node's feature length) without many of the issues, such as reliability and leakage, that occur when transistors for data storage are made at the same scale.

There is a major problem with the memristor architecture, distinct from any issues introduced by material behaviour or manufacturing processes. In a large number of data patterns, low-resistance paths, known as "sneak paths" are present in parallel with a single large resistor, with the current passing through these paths known as "sneak currents". For instance, in Figure 1.1.1, a low resistance path through resistors (1, 1), (1, 3), and (4, 3) (where the indices refer to the row and column respectively), runs parallel to the large resistor at position (4, 1).

This problem manifests itself in the following way. We want to determine the resistance of a memristor, as the magnitude of this resistance gives us information. To do this, we apply a voltage to one of the leads connected to this memristor, and measure the current flowing out of the other lead connected to the memristor through some sensing device. This gives us a resistance (or conductance). Suppose in our example we measure the resistance between the leads to row 4 and column 1 in an attempt to measure the resistance at (4, 1). We will see an intermediate resistance, most likely very close to the low resistance, which is typically very small. Effectively, the high resistance is misread as a low resistance.

This is a problem in information storage. Like any storage device, a memristor array can be considered a channel that transmits data over time (i.e., between an initial storage and subsequent readings). However, this channel is far more data-dependent than most typical examples. If binary data is stored randomly, each 0 (high resistance) has a non-negligible probability of being read as a 1 because of the sneak-path effect. This probability depends entirely on the rest of the data in the array (which will not be known a priori, and will be difficult to recover reliably for this same reason).

Moreover, the data-dependence is non-local: a 0 memristor can be corrupted by a sneak-path through any other part of the array. This means that in the general case, the data stored in the array must be considered as a single highly-intertwined unit.

This thesis examines this problem from an information-theoretic perspective. We begin with background information describing the history of memristors and the current state of device development, as well as a review of the (mostly hardware-level) methods previously suggested to combat the sneak path problem. Then, in the case of sufficiently extreme resistances, we examine arguments to count the fraction of data patterns which do not have the sneak path problem. Next is an examination and comparison of ways of encoding these "legal" patterns that minimize complexity while maximizing the number of patterns that may be represented. Our main figure of merit for a device with an encoding scheme is the information density, which we define and develop. Finally, we develop some extensions to the problem, including non-binary measurements of the array, and non-ideal resistances, as well as some optimization considerations including the complexity of the sensing circuits.

1.2 Overview of physical device development

The current work is concerned more with the behaviour of manufactured memristor arrays, rather than the manufacturing process itself. Nevertheless, a brief history of the development of the devices is given here to provide insight into the relevant physical properties of the arrays, and the current state of their development.

The memristor was first proposed theoretically in 1971 by Leon Chua [3]. While it was theoretically intriguing, and helped extend some mathematical models of fundamental circuit quantities, it was not widely studied for several decades.

This changed in 2008, when a group at HP Labs described how a memristor might be fabricated by exploiting the mobility of dopant particles in a thin semiconductor film to create a material whose resistance could be changed by applying a voltage [4]. The memristive effect depends on the inverse square of the device thickness, meaning that it requires modern nanofabrication methods to build.

In the following years, many memristors were built and tested. The most common memristive materials include metal oxides, such as HfO_2 [5]–[7], TiO_2 [8] and CoO_x [9],

with more exotic materials including blood [10] and slime mould [11].

Extensive work is also underway the better to understand the switching mechanisms of memristive materials and the ways in which conductive paths form and are destroyed, thereby changing the resistance [5]–[7], [12]–[15]. Important physical considerations such as variability, long-term behaviour, noise, and degradation mechanisms are also the object of considerable study [8], [16]–[19].

Additional theoretical work modelling these experimental results includes a variety of mathematical models [20]–[27] and SPICE models [28]–[37].

1.3 Device switching

An excellent survey of existing memristor research, including the properties used to characterize memristors, and the values of these properties achieved in various experiments, is provided in [1]. The following are some of the more relevant properties.

In most devices, switching between resistive states occurs when a large enough voltage is applied (at smaller voltages, the device acts like a resistor). Switching in the opposite direction can either occur when a large voltage of the same polarity is applied (in a unipolar device), or a large voltage of the opposite polarity (in a bipolar device) [1]. Thermally-activated switching is also possible [38].

Most physical devices only switch between two resistances: high and low, with many orders of magnitude between them. It is difficult to repeatably program a device to intermediate values, but it has been achieved in some cases [39], [40].

The major pieces of data that characterize a memory device are its size, power consumption (equivalently, its effective resistance), set/reset speeds and voltages, endurance, and stability.

1.4 Existing hardware-level sneak-path solutions

Sneak-paths are widely recognized as a problem in memristor memory architectures (as well as in some less-common memories with analogous architectures [41]). Attempts to reduce the impact of sneak currents largely fall into two categories: first, solutions that act at the fabrication level to make a device that has reduced sneak currents; and second, solutions that take the existing architecture as a black box and change the data reading and writing processes to extract information in the presence of sneak currents.

A few researchers have also investigated information-theoretical methods for avoiding sneak-paths by encoding the data to memristor patterns that do not induce sneak-paths.

These are the most relevant methods for the current work, and will be discussed below after the appropriate mathematical framework is developed.

1.4.1 Fabrication solutions

Fabrication solutions explore ways of modifying the materials or the circuit architecture to redirect sneak currents. Pan et al.'s extensive survey paper provides a good summary of the following devices (section 4 of [1]). Brief surveys of solutions are also given in [42] and in [43].

Selection devices

One of the most natural solutions at the manufacturing level is to control the nodes through which current can flow by adding some type of selection device at each node. These devices are typically either transistors or diodes. In theory, this is an effective solution, but most proposed designs are currently still problematic either due to their size or their reliability. In addition, the extra devices compromise the 2D and 3D densities of the devices.

Incorporating a CMOS transistor in series with each memristor allows us to switch on only the memristor that we want to read. Several authors have proposed transistor-based solutions [44]–[49]. Other work proposes hybrid devices with a transistor for every few memristors [50].

Diodes have several advantages over transistors. They are typically smaller, and they require no additional wiring for switching nodes (i.e., gates), which makes structures with diodes usable in a wider variety of architectures. Several proposals for introducing diodes have been made, some of which were proposed as a general solution for high-density non-volatile memories before the 2008 proposal of a physical memristor [51]–[54].

Nonlinear devices

Another approach that has led to more robust implementations involves designing arrays with inherently nonlinear elements. In [39], the memristor itself effectively acts as a diode, and its rectifying properties drastically reduce sneak currents. In addition, [39] successfully differentiates between 10 different resistance levels at each node to with a few percentage points. A similarly nonlinear device is described in [55].

In an early paper, [56] simulates nonlinear but symmetric devices, specifically devices with a hyperbolic sine V(I) curve. These require a more involved reading procedure, described below. Note that in general, as noted in [57], increasing the off resistance and increasing the ratio of off to on resistances both increase the range of output voltages, and consequently the decodability of the data. However, as further noted, physical constraints limit how much these factors can be increased.

Complementary resistive switching

Another proposed solution involves having two memristors in series at each array node: one in the high state, and one in the low [58]. Which one is high determines the state of the overall device. Since the resistance at each node is still high, the sneak currents are reduced. The main issue is that reading a high bit from the device resets the bit, meaning that all high bits must be rewritten after reading, which reduces the average read time and increases the number of writes to the device [59].

We note in passing the development of a similar circuit element, know as the *memistor*, in [60], that consists of two memristors connected in series, with access to the electrical node between them. This gives greater control over the resistance of the overall device at the expense of volume.

Careful choice of wire pattern

Another approach involves dividing the crossbar wires into smaller pieces. This makes the addressing more complicated, but has positive implications for density and reading [43]. A similar alternative involves placing memristors at only a select set of lead crossings [61].

Reference cells

Another approach, described in [43], involves including memristors of known state on each word line. The current through these memristors can be used to estimate the sneak current through unknown devices on the same line. This method is, however, very sensitive to the resistance of the reference devices.

1.4.2 Black-box solutions

If the devices are kept small, some information can be extracted without changing the basic structure. This is explored in [57]. In general, more involved methods are used.

Multiple read/write operations

As mentioned briefly above, [56] uses a more involved reading procedure to avoid sneak currents at the device level.¹ This involves reading a bit, and then successively setting the bit to each of the high and low states, reading after each operation, and then resetting the bit to the state with the closer measurement to the original state, and reporting that state. A similar approach is described in [42]. This method is more time-consuming, will degrade the device more quickly, and is only effective if the low resistance is a sufficiently large fraction of the high resistance to make the resulting measurements distinguishable (which also means that at a constant voltage the devices will draw more current, leading to a greater power budget).

Only read operations

In [62], a different approach is taken. By biasing all the input voltages appropriately and comparing a combination of three output voltages against an appropriate threshold, the voltage written to a bit can be determined. This gives an upper bound of three reads per bit output, which is a slower solution, but avoids having to rewrite devices.

Restricted patterns

Vontobel [56] mentions that some gains can be seen when the data is restricted to *balanced patterns*, that is, arrays of memristors where the devices in each row and column are divided equally between the low and high states.

1.4.3 Discussion

In the present work, we will be focusing on black-box devices due to the relative simplicity of their structure compared to the various kinds of switched devices that required additional manufacturing steps, and in some cases more complex connectivity requirements. We will focus on a model which treats every memristor as either a short circuit or an open circuit, which is often a valid approximation, and provides a useful mathematical foundation for schemes with intermediate resistances, which will vary based on device parameters.

¹While [56] does explore fabrication, this particular technique is applicable to a range of devices.

1.5 Outline

The remainder of this thesis is divided into four chapters. Chapter 2 focuses on the capacities of single-layer memristor arrays from a mathematical perspective. Chapter 3 provides a discussion of existing encoding schemes for these devices, and proposes and analyzes some alternatives. Chapter 4 extends the capacity discussion of chapter 2 to multi-layer devices. Finally, chapter 5 discusses our conclusions and suggestions for future work.'

Chapter 2

Capacity of single-layer devices

This chapter explores some of the existing ways of representing the state of a singlelayer memristor array¹, and proposes some modifications and new methods that will be useful in subsequent chapters. Using this framework, we will derive the number of distinguishable patterns that may be stored in the array, giving two new derivations for an existing result.

2.1 General description and ideal model

Given a memristor array, we want to identify the pattern of resistances stored in it. In particular, we would like to do this using the smallest number of operations on the array. We will count the number of applied voltages and the number of measured voltages to describe this.

In the typical case, we have a single layer of memristors in a rectangle of m devices by n. The memristors can be programmed to values in a set of conductances, which will usually be the set $\{h, \ell\}$, where h has a high value, and ℓ has a low value. The ideal case has $h \to \infty$ and $\ell \to 0$.

We will choose to apply voltages to some subset of the n+m available nodes (*m* leads on one layer, and *n* on the other), and to take readings at some other subset of the nodes. Practically, these subsets cannot intersect, and in most cases, the inputs will be placed along one edge of the device, and the outputs taken along a perpendicular edge. The applied voltages will be binary on/off signals for the present discussion. If we make the assumptions above that *h* and ℓ are extreme, these readings tell us whether the output nodes are connected to the input nodes.

¹That is, an array with one layer of memristors between two layers of leads.

In practice, the circuits used for reading and writing will have finite positive conductances, although we will take them to zero or infinity as appropriate here.

2.2 Device representations

The state of the device is defined by a series of low and high conductances, often represented by 0s and 1s. For our purposes, it is often most useful to store this array state in one of two mathematical objects: the matrix and the array.

It is worth noting that the array state corresponds to an encoded form of the data that we wish to store in the array. We will use "data" to refer to the information we wish to store, as an unencoded string of 1s and 0s from the set $\{0,1\}^q$ for some integer q, and we will use "array state" to refer to the result of encoding this data into a series of high and low conductances, represented by elements of the set $\{0,1\}^{mn}$. We will typically have k < mn.

These representations make the most sense if we assume, as we will, that the high conductance value is high enough to be considered a short circuit, and the low conductance value is low enough to be considered an open circuit.

2.2.1 Matrix representation

It is natural to represent the array state in matrix form. Sotiriadis provides a rigorous mathematical background for a functionally similar device, and suggests two matrix representations of a device [41]. First is the biadjacency matrix \mathbf{F} , which has a one at the (i, j) position when the memristor at the (i, j) position is in the high-conductance state (i.e., when resistance is 0), and a zero otherwise (when resistance is infinite). An alternative representation that, while often more useful, contains less information, is the reachability matrix \mathbf{D} , which has a one at (i, j) when the *i*th horizontal wire is connected to the *j*th vertical wire through any low-resistance path, and a zero otherwise. This better captures the overall connectivity of the device. Note that

$$\mathbf{D} = \left[\mathbf{F} + \mathbf{F}\mathbf{F}^T\mathbf{F} + (\mathbf{F}\mathbf{F}^T)^2\mathbf{F} + \dots + (\mathbf{F}\mathbf{F}^T)^{k-1}\mathbf{F}\right]_{>0}$$
(2.1)

where $k = \min\{m, n\}$ for an $m \times n$ array. The matrix operator $[\cdot]_{>0}$ sets positive elements of a matrix to 1 and all other elements to 0. Define

$$\mathbf{H} \equiv \mathbf{F} + \mathbf{F}\mathbf{F}^T\mathbf{F} + (\mathbf{F}\mathbf{F}^T)^2\mathbf{F} + \dots + (\mathbf{F}\mathbf{F}^T)^{k-1}\mathbf{F}$$
(2.2)

so that $\mathbf{D} = [\mathbf{H}]_{>0}$. The (i, j)th element of \mathbf{H} gives the number of paths between the *i*th horizontal wire and the *j*th vertical wire. This follows from the fact that the term $(\mathbf{FF}^T)^{i-1}\mathbf{F}$ gives the number of paths of length 2i - 1 between the wires (where the length refers to the number of low-resistance connections). This follows from the graph theoretic properties of the biadjacency matrix: it is a blockwise section of an adjacency matrix, whose powers give the numbers of paths between wires in the device. Note that every path from a horizontal to a vertical wire will have odd length.



Figure 2.2.1: Two representations of the same 16-memristor array. After [2].

As an example, consider the device with m = n = 4 in the state shown in Figure 1.1.1. This circuit is reproduced in Figure 2.2.1a. Figures 2.2.1b and 2.2.1c give the corresponding matrix representations. We see that the only difference in this example is at position (1, 1). As discussed previously, there is a sneak path at this position, which means that the leads to the first row and first column are shorted together, even though the memristor at this location is in the low-conductance state.

We can also verify (2.1) in this case. We have the following products of \mathbf{F} and \mathbf{F}^T . These matrices give the number of paths between the leads of lengths 1, 3, 5, and 7 respectively. We can stop after 7, because we have now reached the paths that include all 8 leads (we don't count the first lead in the path length).

$$\mathbf{F}\mathbf{F}^{T}\mathbf{F} = \begin{bmatrix} 2 & 0 & 3 & 0 \\ 0 & 4 & 0 & 4 \\ 0 & 4 & 0 & 4 \\ 1 & 0 & 2 & 0 \end{bmatrix}$$
(2.3)
$$\mathbf{F}\mathbf{F}^{T})^{2}\mathbf{F} = \begin{bmatrix} 5 & 0 & 8 & 0 \\ 0 & 16 & 0 & 16 \\ 0 & 16 & 0 & 16 \\ 3 & 0 & 5 & 0 \end{bmatrix}$$
(2.4)

$$\left(\mathbf{F}\mathbf{F}^{T}\right)^{3}\mathbf{F} = \begin{bmatrix} 13 & 0 & 21 & 0\\ 0 & 64 & 0 & 64\\ 0 & 64 & 0 & 64\\ 8 & 0 & 13 & 0 \end{bmatrix}$$
(2.5)

For instance, the 1 in the lower-left corner of $\mathbf{FF}^T\mathbf{F}$ corresponds to the length-3 sneak path we have already identified between row 4 and column 1. Many of these paths, especially the longer ones, often double back on themselves. For instance, the 3 in the lower-left corner of $(\mathbf{FF}^T)^2 \mathbf{F}$ indicates the three ways that we can travel along that same sneak-path, backtracking once through a single resistor (one example being row 4 to column 3, back to row 4, back to column 3, to row 1, to column 1).

It should also be noted that the set of \mathbf{F} matrices that map to a matrix \mathbf{D} includes the matrix \mathbf{D} . For this reason, as well as reasons of symmetry, \mathbf{D} is often the best choice when we want to choose a representative of each class of \mathbf{F} matrices. Note that this memristor pattern that has the same \mathbf{F} and \mathbf{D} matrices has no sneak paths (because all of the connections between two nodes are explicitly present as a high-conductance device). In a matrix, this means that if three 1s (high conductances) are the corners of a rectangular submatrix, the entry at the fourth corner must also be a 1. The mathematical implications of this are discussed further by Riguet [63]. It is worth noting that while any binary $m \times n$ matrix can be an \mathbf{F} matrix for a memristor array, only a subset of these matrices are legal \mathbf{D} matrices.

Given the above, it is natural to try to determine which of the \mathbf{D} matrices corresponds to a memristor array. In fact, while sneak-paths will generally prevent us from uniquely identifying the \mathbf{F} matrix of an array, we can always find the \mathbf{D} matrix. The most conceptually simple approach is a "brute-force" test of connectivity. This involves checking the conductance between each pair of input and output nodes. A natural way of doing this is sequentially to drive each input node to some nonzero voltage, letting the others float, and measure the voltages at the output nodes. Those that match the input voltage are shorted to the relevant input node, and those that do not are isolated from it. (In practice, each output should have a small pull-up or pull-down resistor to some other voltage, probably ground, in order to make the voltages at these isolated nodes well-defined.) This will determine a representative \mathbf{D} matrix, which can be mapped to a data pattern using a predetermined encoding scheme.

In short, we cannot reliably store data naïvely in the array, that is, in \mathbf{F} matrix form, but we can store data if we can first map it to a \mathbf{D} matrix.

Identifying D matrices

We already know that sneak paths are present (i.e., our memristor pattern is not in \mathbf{D} matrix form) if exactly three out of four corners of any rectangle of memristors are high-conductance. The following theorem provides an alternative way to check whether a given matrix is a possible \mathbf{D} matrix which will be helpful to our discussion of the theory of binary relations. It is stated without proof by Riguet in section 1.7 of [63].

Theorem 2.2.1. Consider an mn-bit pattern \mathbf{P} , divided into m rows of n bits each, so that $\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_m)$. Then \mathbf{P} is a possible \mathbf{D} matrix for some $m \times n$ memristor array, as defined above, if and only if the bitwise AND of two rows $\mathbf{p}_i \cdot \mathbf{p}_j$ equals 0^n for all (i, j) where $\mathbf{p}_i \neq \mathbf{p}_j$.

Proof. We first prove the forward direction: i.e., we prove that if $\mathbf{p}_i \cdot \mathbf{p}_j = 0^n$ for all (i, j) where $\mathbf{p}_i \neq \mathbf{p}_j$, then **P** is a **D** matrix for at least one **F** matrix. We prove this by constructing such an **F** matrix.

Create bipartite graph **G** with *n* nodes in one bipartition (the "input") and *n* in the other (the "output"). If \mathbf{p}_i has a high bit in the *j*th position, draw an edge between the *i*th input and the *j*th output. Call the **D** matrix for the array corresponding to this graph **P'**. We will show that $\mathbf{P} = \mathbf{P'}$. First, by construction, if $\mathbf{P}_{ij} = 1$, $\mathbf{P}'_{ij} = 1$, where the *ij* index refers to the *j*th element of the *i*th row. It remains to show that if $\mathbf{P}'_{ij} = 1$, then $\mathbf{P}_{ij} = 1$. Suppose toward a contradiction that there exists a pair (i, j) such that $\mathbf{P}'_{ij} = 1$ and $\mathbf{P}_{ij} = 0$. Since $\mathbf{P}_{ij} = 0$, the *i*th input is not directly connected to the *j*th output. This means that the shortest path from the *i*th input to the *j*th output is of the form *i*th input $\rightarrow j'$ th output $\rightarrow i'$ th input \rightarrow path $\Pi \rightarrow j$ th output for some indices *i'* and *j'*, and some (possibly empty) path Π . Because this is the shortest path, the first node in Π (or the *j*th output itself if Π is empty) is adjacent to the *i*th input, but not the *i*th input. This means that $\mathbf{p}_i \neq \mathbf{p}_{i'}$, which means (by assumption) that we must have $\mathbf{p}_i \cdot \mathbf{p}_{i'} = 0^n$. However, since both the *i*th and *i'*th inputs are adjacent to the *j*th output, both \mathbf{p}_i and $\mathbf{p}_{i'}$ have a high bit in the *j*th position, which means that the bitwise AND also has a high bit in this position, which is the desired contradiction.

Now we prove the reverse direction: i.e., we prove that if \mathbf{P} is a legal \mathbf{D} matrix (i.e., it corresponds to some \mathbf{F} matrix), then $\mathbf{p}_i \cdot \mathbf{p}_j = 0^n$ for all (i, j) where $\mathbf{p}_i \neq \mathbf{p}_j$. Suppose toward a contradiction that there exist i and j such that $\mathbf{P}_{ik} \neq \mathbf{P}_{jk}$ for some index k, but $\mathbf{P}_{i\ell} = \mathbf{P}_{j\ell} = 1$ for some index ℓ . This means that the *i*th and *j*th inputs are connected to each other through the ℓ th output, and must therefore be connected to the same set of outputs. This contradicts the fact that only one of them is connected to output k, which completes the proof.

Note that Theorem 2.2.1 is equivalent to stating that a bit pattern is a legal \mathbf{D} matrix if and only if each pair of rows has either all or none of its high bits in common. It is also equivalent to check columns instead of rows, as the array is symmetric in the rows and columns.

2.2.2 Graph representation

It is often convenient when visualizing the array connectivity to think of the matrices \mathbf{D} and \mathbf{F} as the biadjacency matrices for bipartite graphs. That is, we have a bipartite graph with m nodes in one part, corresponding to the inputs, and n nodes in the other part, corresponding to the outputs. A 1 at position (i, j) in binary matrix \mathbf{D} or \mathbf{F} corresponds one-to-one with an edge between the *i*th input and the *j*th output. In practice, we will be more interested in the graph corresponding to the \mathbf{D} matrix.



Figure 2.2.2: A circuit diagram and the corresponding connectivity graph for a sample array. Here, the R nodes correspond to the rows, and the C nodes correspond to the columns. Note that the sneak-path at (4, 1) in the circuit is represented as an explicit connection in the matrix and the graph.

Figure 2.2.2 shows an example of this representation. On the left is a circuit diagram from [64], with black resistors low and white resistors high, and both row and column leads labelled 1 through 4. On the right is the graph representation of the same circuit. The row wires correspond to the R_x nodes on the left, and the column wires correspond to the C_y nodes on the right. Two nodes are connected if and only if the corresponding leads are directly connected by a low-resistance memristor.

We can make a few comments about the resulting graphs. First, as mentioned, the graphs are bipartite, with the rows and columns corresponding to the two parts. This corresponds physically to the fact that every memristor connects a row lead to a column lead. Second, note how the sneak-paths present themselves in this representation. A

sneak-path exists between two nodes in a graph if and only if the nodes are not connected by an edge, but are connected by some longer path.

2.3 Identifying and counting equivalent patterns

When sneak paths are present, several different \mathbf{F} matrices will give the same \mathbf{D} matrix. We can, in fact, partition the set of all \mathbf{F} matrices according to the corresponding \mathbf{D} matrices. We would like to know how many partitions this creates, since this is the number of distinguishable patterns that can be stored in the device.

2.3.1 Stirling numbers

Definition 2.3.1. The Stirling number of the second kind $\binom{n}{k}$ is the number of ways in which n items may be partitioned into k (non-empty) sets.

For example, we have $\binom{4}{2} = 7$, since we may partition a set of 4 elements into 2 subsets in 7 ways. That is, we may partition the set $\{a, b, c, d\}$ into 2 subsets as one of

$$\{\{a\}, \{b, c, d\}\}, \{\{b\}, \{a, c, d\}\}, \{\{c\}, \{a, b, d\}\}, \{\{d\}, \{a, b, c\}\}, \\ \{\{a, b\}, \{c, d\}\}, \{\{a, c\}, \{b, d\}\}, \{\{a, d\}, \{b, c\}\}.$$
(2.6)

2.3.2 Number of distinguishable array patterns

We want to know how many different patterns that might be stored in a memristor array are distinguishable (that is, how many have different **D** matrices). This is a function of the array dimensions m and n (both integers), denoted T(m, n). Sotiriadis derives this value [41].

Theorem 2.3.1. For an $m \times n$ device, there are

$$T(m,n) = \sum_{k=0}^{\min(m,n)} {\binom{n+1}{k+1}} {\binom{m+1}{k+1}} k!$$
(2.7)

distinguishable data patterns.

Sotiriadis provides a rigorous mathematical proof of this result. We will explore an alternative and original proof here. We also provide a mathematical extension from the theory of relations to improve our intuitive understanding of this quantity.

2.3.3 Proof of Theorem 2.3.1 from graph theory

Proof. Consider the graph representation of the array state (in the **D** matrix form). We have a bipartite graph with bipartitions μ and ν with m and n nodes respectively (indexed from 1 to m and 1 to n respectively). Call the number of connected components k. A connected component is a set of nodes that are all reachable from each other and from no nodes outside of the set. Each of these connected components is a complete bipartite subgraph (i.e., each element of one part is connected to every element of the other part). Some nodes in either part may be disconnected, i.e., singletons.

Consider all the array graphs with k non-singleton connected components. Further consider the sets of integers $\mathcal{N} = \{0, 1, 2, ..., n\}$ and $\mathcal{M} = \{0, 1, 2, ..., m\}$. Partition each of \mathcal{M} and \mathcal{N} into k + 1 subsets. There are respectively $\binom{n+1}{k+1}$ and $\binom{m+1}{k+1}$ ways of making these partitions (from the definition of Stirling numbers).

In each of these partitions, we will have a subset containing the integer 0 from \mathcal{M} or \mathcal{N} . We will name these subsets \mathcal{D}_m and \mathcal{D}_n respectively. These represent the disconnected nodes: if $x \in \mathcal{D}_m$, then node $x \in \mu$ is disconnected (and analogously for ν). If \mathcal{D}_m (or \mathcal{D}_n) equals {0} then every node in μ (or ν) is connected to some node in ν (or μ).

This leaves us with k subsets of each of \mathcal{M} and \mathcal{N} , each corresponding to a connected component of the graph. There are k! ways of pairing the connected subsets of \mathcal{M} with the connected subsets of \mathcal{N} , so there are $T_k = {n+1 \\ k+1} {m+1 \\ k+1} k!$ different array graphs with k non-singleton connected components. We can have as few as k = 0 of these components (in the case where every node is a singleton), and as many as $\min(m, n)$ (in the case where one of the parts of the bipartite graph has each node connected to a different set of nodes from the other part). We then sum T_k over this range to obtain the desired result.

2.3.4 Connections to the theory of binary relations

The expression above for T(m, n) also appears in the theory of binary relations as the number of diffunctional relations between sets of m and n elements [65]. This means that we can prove Theorem 2.3.1 by establishing a one-to-one correspondence between **D** matrices and diffunctional relations. To do this, we need the following definitions.

Definition 2.3.2. A relation from a set \mathcal{E} to a set \mathcal{F} is a set of ordered pairs $(e, f) \equiv eRf$, where $e \in \mathcal{E}$ and $f \in \mathcal{F}$. We will use the notation e.R to indicate the image of e, that is $e.R = \{f | eRf\}$, or in other words the set of all ordered pairs in the relation whose first element is e. **Definition 2.3.3.** A relation R from set \mathcal{E} to set \mathcal{F} is diffunctional if and only if the $a \in \mathcal{E}$ and $b \in \mathcal{E}$ are such that $a.R \cap b.R \neq \emptyset$ implies a.R = b.R.

This is one of several equivalent definitions. It is originally due to Everett [66], and is stated in this form by Jaoua et al. (Definition 13.1.4) [67].

Difunctional relations are useful to consider because the number of difunctional relations between a set of m elements and a set of n elements is exactly the expression in (2.7). This means that there is a one-to-one correspondence between difunctional relations and **D** matrices for memristor arrays. One such correspondence in particular is very natural, and we will construct it below.

Theorem 2.3.2. The number of difunctional relations between a set \mathcal{E} of m elements and a set \mathcal{F} of n elements is equal to the number of distinct \mathbf{D} matrices for an $m \times n$ array, as defined above, with m row "inputs" and n column "outputs".

Proof. Without loss of generality, let \mathcal{E} be the set of row indices of the array, labelled $\{1, 2, \ldots, m\}$, and let \mathcal{F} be the set of column indices, labelled $\{1, 2, \ldots, n\}$. We can define a relation R between \mathcal{E} and \mathcal{F} by declaring that for $e \in \mathcal{E}$ and $f \in \mathcal{F}$, we have eRf if e is shorted to f. Let \mathcal{R} be the set of all such relations, and let \mathcal{B} be the set of all diffunctional relations between \mathcal{E} and \mathcal{F} . To prove the theorem, we will show that $\mathcal{R} = \mathcal{B}$.

First, we show that $\mathcal{R} \subseteq \mathcal{B}$. This means that every relation defined as above by shorts in the array is difunctional. This follows from Theorem 2.2.1. First note that there is a one-to-one correspondence between relations R and the matrices \mathbf{D} . In particular, the elements of R are ordered pairs of the indices of the 1s in \mathbf{D} . For an input a, the image a.R is the set of indices of the 1s in row a of \mathbf{D} . By Theorem 2.2.1 this means either $a_1.R = a_2.R$ or $a_1.R \cap a_2.R = \emptyset$, for any distinct inputs a_1 and a_2 , which is exactly the definition of a difunctional relation, so $\mathcal{R} \subseteq \mathcal{B}$, as desired.

Next we show that $\mathcal{B} \subseteq \mathcal{R}$, that is, every diffunctional relation defines a connectivity pattern on an array. We do this by picking an arbitrary $B \in \mathcal{B}$ and constructing an array that corresponds to this relation. For each input a, connect a to all the outputs in a.B. This induces a relation B'. By the above, B' is diffunctional. It remains to show that B' = B.

Take an arbitrary element $a \in \mathcal{E}$. It is sufficient to show that a.B = a.B'. By construction, $a.B \subseteq a.B'$. We need to show that $a.B' \subseteq a.B$. Suppose toward a contradiction that there exists $f \in \mathcal{F}$ such that $f \in a.B'$ and $f \notin a.B$. This means that a is connected to f under relation B', but not under relation B. The argument proceeds analogously to the proof of the converse of Theorem 2.2.1. In short, a is not directly connected to f (by construction), so it must be connected to f through some other input

a' and some other output f'. However, since the relation is difunctional and a and a' are both connected to f' they must be connected to the same set of outputs, which is the desired contradiction.

Therefore the number of **D** matrices for an $m \times n$ array is equal to the number of difunctional relations between a set of size m and a set of size n. Moreover, there is a natural mapping between the two sets.

This view of device connections as "bundles" of connections between two layers of wires will prove to be of particular value in later analysis when we consider multi-layer devices.

2.3.5 Physical array readings

In the physical array, we use the following procedure to distinguish between patterns. Attach a power source to some "input" subset of the wires. Read resulting voltages on some other "output" subset of the wires. Repeat this process until the desired connections have been determined. In practice, it will be easiest to take input wires from one edge of the device, and outputs from a perpendicular edge, since these correspond to voltages across individual memristors.

2.3.6 Capacity and asymptotics

The data storage capacity in an idealized $m \times n$ array is $\log_2 T(m, n)$.² While this is an exact expression, the Stirling numbers make it difficult to use in efficient calculations. Because of this, it is useful to develop an asymptotic scaling for the capacity.

Note that the capacity of arrays of constant perimeter (i.e., m + n fixed) have maximum capacity when they are square, but arrays of constant area (i.e., mn fixed) have minimum capacity when square [41]. The above expression is symmetric in m and n, which was to be expected from the symmetry of the device: since all rows are connected to all columns, there is no way of distinguishing them mathematically.

An approximation of the capacity for large m + n (from (8) in [41]) is

$$(n+m-q)\log q + q\log e \tag{2.8}$$

where

$$q = \frac{n+m}{\log_e(m+n)}.$$
(2.9)

²From now on, all logarithms may be assumed to have base 2 unless otherwise specified

This approximation of the capacity depends on m and n only through m + n and is asymptotic to $(m + n) \log(m + n)$. This is formalized mathematically by Sotiriadis [41]. In particular, Sotiriadis proves the following theorem [41, Theorem 4, proved by analogy with Theorem 3].³

Theorem 2.3.3. The capacity T(m,n) of an $n \times m$ memristor array has the property

$$\lim_{\substack{m,n \to \infty \\ n/m \to a > 0}} \frac{T(m,n)}{(m+n)\log(m+n)} = 1$$
(2.10)

This means that the capacity grows as $p \log p$, where p is the semiperimeter of the array. The constraints in the limit on m and n force the device to have a finite aspect ratio in the limit as the perimeter and area grow. This is necessary, since a $1 \times (m + n)$ device will always have full capacity 2^{mn} . Such a device is physically difficult to manage in most contexts, however. We will also see later that for at least some encoding schemes, the additional cost of peripheral encoding and decoding hardware will make us favour devices of intermediate aspect ratio anyway.

2.3.7 Figures of merit

To determine what choice of parameters makes a memristor array as good as possible, we need to know what makes a memristor array good. There are many desirable physical properties of an array, including its switching speed, ratio of high to low resistances, and lifetime as a number of switching cycles, many of which are surveyed for a large number of experiments and materials in [1]. Power consumption is another important consideration, and it is explored for the basic encoding scheme from [64] described below in section 3.1.

The main figure of merit that we will consider here is the information density of a device. In its simplest form, this is $\rho = [\log T(m, n)]/(mn)$. However, as seen in [41], this gives somewhat unsatisfactory results. For a given area A = mn, the densest results are given when m = 1 or n = 1, since this gives a $1 \times A$ array which, by construction, cannot include any sneak paths. Such a geometry will be impractical for applications involving large A, so we will update our figure of merit as follows to reflect this.

One of the factors that makes the $1 \times A$ arrays unappealing is that they have maximal perimeter, and therefore maximal peripheral hardware for reading and writing. This hardware will also consume some area, which in general may be non-negligible as it may still be implemented in standard silicon transistors, which are comparatively large.

³The notation and terminology have been altered slightly for consistency.

The exact area of the peripheral hardware will vary based on the chosen encoding scheme. However, since every read or write will find or change a binary symbol (high or low resistance), it is reasonable to assume that we will need at least one read or write per bit, and consequently peripheral hardware of logarithmic depth. That is, our hardware will have area $A_h(m, n) = C(m \log m + n \log n)$ for some constant C.

We can now update our density figure of merit to

$$\rho_C(m,n) = \frac{\log T(m,n)}{A(m,n) + A_h(m,n)} = \frac{\log T(m,n)}{mn + C(m\log m + n\log n)}$$
(2.11)

Figure 2.3.3 gives the values of ρ_C for a few fixed areas A = mn as a function of m for both C = 1 and C = 10. We see that with the addition of peripheral hardware, even in the case of small C, the square arrays are the densest. Because of this, from now on we will focus on square arrays, for which we can consider how T changes as a function of the array side length.

We also see that larger areas are less dense at their maxima, which makes sense, since larger areas have more opportunity for sneak paths. We also see that, as expected, the curves are symmetric (in a multiplicative sense) around $m = n = \sqrt{A}$. This is a result of the symmetry of the arrays.

If we plot a heatmap of this density for array sizes up to 40×40 , which is shown in figure 2.3.4a, we discover a potential issue. By a large margin, the highest density occurs at the m = n = 1 case, since this sets the logarithms to zero. This is unrealistic for many encoding schemes, as even a single-cell memristor will usually need peripherals (we will examine one encoding scheme where this is not the case later). To adjust this, we can consider a different approach where the area of peripherals scales with the perimeter, making $A_h(m,n) = C(m+n)$, with technology-dependent constants C. In this case, when C = 10, we see that an intermediate array is optimal (the 5 × 40 array is the best array with both dimensions at most 40). The dimensions of the optimal array depend on the value of C. This scheme adds the necessary overhead for small arrays, but loses the relationship with the number of bits in the system.

In general, we may expect many encoding schemes to have a combination of these schemes, with $A_h(m,n) = C_1(m+n) + C_2(m\log m + n\log n)$, with constants C_1 and C_2 . Another alternative is to have an area expression of the form $A_h(m,n) = C_1 + C_2(m\log m + n\log n)$.

It is also possible, as described in [64], to focus on one area of an array by grounding a subset of the wires. This is another way to increase the effective density of an array, by making a large array behave like a series of small ones. Since there is no obvious choice for the best device density calculation, and since the different options vary so widely, in the following discussion, we will often look an the alternative figure of merit for encoding schemes: the number of patterns in the range of the scheme as a fraction of the total number of patterns that may be stored in an array of that size.



Figure 2.3.3: Density ρ_C for C = 1 and varying areas. Note the difference in scales.



(a) Density of peripherals when $A_h(m,n) = C(m \log m + n \log n)$ with C = 10Density for log-depth peripherals



(b) Density of peripherals when $A_h(m,n) = C(m+n)$ with C = 10

Chapter 3

Encoding schemes for single-layer devices

3.1 Existing encoding scheme

A key source for current research on the information theory of memristor arrays is [64]. There, Cassuto et al. propose a scheme for encoding information into memristor arrays in a way that is asymptotically optimal, only maps to distinguishable patterns of memristors, and in most cases is reasonably easy to encode and decode.

We will be examining the third of three encoding and decoding schemes proposed in [64] (section IV.C). In terms of speed and fraction of capacity achieved, the other two schemes scale less well, since they require calculations that scale superpolynomially in the device dimensions. This scheme does, however, have encoding issues that will be discussed below. The scheme is optimized over a parameter L satisfying $1 \le L \le \min\{m, n\}$. It encodes $(m + n - L) \log L$ bits of information, meaning that practically, L should be a power of 2. The bits are arranged into m + n - L binary vectors of length log L labelled $(\mathbf{r}_1, \ldots, \mathbf{r}_m, \mathbf{c}_1, \ldots, \mathbf{c}_{n-L})$.

We pick a function f that maps $\log L$ binary vectors to integers between 1 and L bijectively.¹ A natural choice is to think of the binary vector as a binary number, convert it to decimal, and add 1, although any f satisfying these criteria will work equally well. Now we construct the $m \times n$ array **A** according to Algorithm 1, with indexed elements indicated by a_{ii} (from [64]).

That is, the $m \times n$ array is divided into an $m \times L$ "row data" section and a $m \times (n-L)$ "column data" section. The row section has exactly one 1 in each row, whose index in that

¹Bijectivity is not specified in [64], but is necessary, as will become apparent.

	Algorithm	1	An	encoding	algorithm	from	[64
--	-----------	---	----	----------	-----------	------	-----

for i = 1, ..., m do $a_{i,f(r_i)} = 1$ end for for j = 1, ..., n - L do $a_{1:m,j+L} = a_{1:m,\psi(c_j)}$ end for for i = 1, ..., m do for j = 1, ..., n do if $a_{ij} \neq 1$ then $a_{ij} = 0$ end if end for end for end for

row encodes $\log L$ bits of information. The column data section duplicates columns from the row data section, which are (usually) distinct, and encodes information in the index of the column that has been duplicated. The distinctness is, however, not guaranteed. This is due to the encoding issue mentioned briefly, which will be analyzed below.

Assuming distinctness, the rate of this system is $(m + n - L) \log L$. The value of L that maximizes this is denoted L^* and grows asymptotically as $(m + n)/\log(m + n)$, which means the overall fraction of distinct patterns that can be encoded approaches capacity.

3.1.1 Example

Consider an array with m = 4, and n = 6, which gives $L^* = 4$. We can store $(m + n - L^*) \log L^* = 12$ bits of information in this array using this scheme. As an example, we will store the bitstring 011110001001.

First, divide this bitstring into the **r** and **c** vectors of length $\log L^* = 2$. This gives the vectors **v** listed in table 3.1, with corresponding values $f(\mathbf{v})$, using the binary conversion plus 1 for the function f.

Now we perform step 1, by entering the **r** vectors into the first L^* columns of an $m \times n$ array. For vector \mathbf{r}_i , we set the $f(\mathbf{r}_i)$ th element of the *i*th row to 1. This gives the following matrix, where "?" indicates a memristor whose value has not yet been set,

Vector ${\bf v}$	Name	$f(\mathbf{v})$
01	\mathbf{r}_1	2
11	\mathbf{r}_2	4
10	\mathbf{r}_3	3
00	\mathbf{r}_4	1
10	\mathbf{c}_1	3
01	\mathbf{c}_2	2

Table 3.1: Vectors of bits, with names and f-values, for a sample bitstring and m = 4, n = 6.

and the bar divides the row and column sections:

$$\begin{bmatrix} ? & 1 & ? & ? & ? & ? \\ ? & ? & 1 & ? & ? \\ ? & ? & 1 & ? & ? & ? \\ 1 & ? & ? & ? & ? & ? \end{bmatrix}$$
(3.1)

Proceeding to step 2, we duplicate column $f(\mathbf{c}_j)$ into the $j + L^*$ th column for j = 1and j = 2. That is, we duplicate the third column into the fifth, and the second column into the sixth, giving

$$\begin{bmatrix} ? & 1 & ? & ? & 1 \\ ? & ? & 1 & ? & ? \\ ? & ? & 1 & ? & 1 & ? \\ 1 & ? & ? & ? & ? & ? \end{bmatrix}$$
(3.2)

We have now set all the 1s in the array. Finally, step 3 fills in the remaining unknown memristors to zeros, giving

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
(3.3)

Note that, as desired, there are no sneak-paths in this array. We also have no rectangles of ones in the row-data section. The row-data section has four rows, each of which encodes $\log 4 = 2$ bits of information. The column-data section has two columns, each of which also encodes $\log 4 = 2$ bits of information.



Figure 3.1.1: The number of bits as a fraction of the maximum value T(m, n) as a function of p = m + n, for the optimal choice of L and two array geometries

3.1.2 Comments

The existing row/column scheme from [64] discussed above encodes $(m + n - L) \log L$ bits in mn memristors. The choice of L that maximizes capacity is asymptotically

$$L^* = \frac{m+n}{\log(m+n)} \tag{3.4}$$

Note that L^* depends on m and n only through the semi-perimeter p = m+n. Moreover, the maximum number of bits with this scheme approaches the overall maximum quickly. This is seen in Figure 3.1.1 for the cases n = L (i.e., no c_j data in the original scheme) and m = n (square arrays). Note that for constant p = m + n changing m and n affects T(m, n) but not the maximum number of bits that can be encoded using this scheme. In these calculations, L and log L were constrained to the integers, since they correspond to integer numbers of devices. Similarly, the minimum value for p is 3, since smaller values give arrays of devices with nonpositive dimensions.

It is also worth noting that the optimum value $L^* = p/\log p$ is an asymptotic approximation. In a physical device, L and $\log L$ are integers, meaning that L is a power of 2. However, even the approximation that L is the smallest power of 2 larger than $p/\log p$, i.e., $2^{\lceil \log(p/\log p) \rceil}$ is occasionally incorrect. A comparison between the true power-of-2 op-



Figure 3.1.2: The optimum value of L as a function of p, and two approximations, when using the third encoding scheme from [64].

timum (found by brute-force iteration) and these two approximations for p up to 200 is given in Figure 3.1.2.

3.1.3 Error probability

In a small subset of cases this scheme from [64] cannot be decoded uniquely. This happens when two of the columns in the row data are identical. Because there is only ever one 1 per row in the row data section, these identical columns will always be columns of zeros.

Since each row of the $m \times L$ row data section contains exactly one 1, the probability of any one element of this section being 1 is $P_1 = L^{-1}$, meaning the probability of any one element of this section being 0 is $P_0 = 1 - P_1 = 1 - L^{-1}$. We are interested in columns of zeros. The probability that a given column is all zero is $P_c = P_0^m = (1 - L^{-1})^m$, assuming that the 1 in a row is equally likely to occur in each position. The probability that no column is all zero is then $P_{\text{none}} = (1 - P_c)^L$ and the probability that a given column is the only column of all zeros is $P_{\text{only}} = P_c(1 - P_c)^{L-1}$.

We want the probability that at least two columns are all zeros. This is the complement of the probability that either no column is all zeros, or exactly one column is all zeros. Since there are L columns, this is $1 - (P_{\text{none}} + LP_{\text{only}})$. For optimal L and square arrays, this probability is between 0.05 and 0.15 for m between 10 and 500. Moreover, this probability for this geometry is an increasing function of m, meaning that larger square arrays are more prone to error than small ones.

In order to avoid these errors without materially changing the encoding scheme we need to ensure that each column of memristors contains at least one 1 (in fact, we need only ensure that all but one of the columns have this property, but it will be easier for the moment to maintain the rule for all columns). There are a few ways in which we can do this. Each of them involves either a reduction in the amount of information stored, or constraints on the geometry of the device.

If we have control over the geometry of the array, we can avoid this problem by choosing $n = L^*$ (since L^* depends on m and n only through p). This eliminates the column data section of the original scheme, and therefore the problems it can cause.

In this case, we have $m = p - n = p - L^*$. Since L^* , and so n, is asymptotic to $p/\log p$, this gives arrays that have a fairly skewed aspect ratio, which may not be desirable in all architectures. Moreover, this solution does assume control over the geometry of the system which may not be available in all contexts.

3.2 Simple "identity matrix" encoding

Perhaps the simplest way of guaranteeing that each column contains a 1 is to have the first L^* rows of the "row data" include all the possibilities for a single 1 in order, that is, to set $a_{i,i} = 1$ for $i \in \{1, 2, ..., L^*\}$ and $a_{i,j} = 0$ for $i \in \{1, 2, ..., L^*\}$ when $j \neq i$. This this section of the memristor array the appearance of an identity matrix in linear algebra.

The rate of this method is reduced by $L \log L$ bits, since we are setting the first L of m rows of the "row data" section, which originally encoded $m \log L$ bits. This gives an overall capacity of $(m + n - 2L) \log L$, or $(p - L) \log L$ bits, compared with $(p - L) \log L$ for the initial scheme. The number of reading and writing operations required to read and change data is the same as in the original scheme. This is included in Figure 3.3.3.

3.3 Permuted identity matrix encoding

We can improve on this rate somewhat by permuting the rows of the identity matrix to store additional information. There are L^* ! such permutations, so we recover $\log L^*$! bits by doing this. This increases the capacity to $(p-L^*) \log L^* + \log L^*$! at the cost of making it more complicated to read or alter the data stored in the permutation section of the array. In particular, this process will require a more sophisticated external computing unit to determine the specific permutation present.

The rates of these two alternatives are compared against the rates of the (error-prone) original scheme in Figure 3.3.3. The distinct "bumps" in the curves are a result of the discreteness of the optimal values of L^* , which must be powers of 2 in order to produce integer-dimensioned arrays of both data bits and memristors.



Figure 3.3.3: Comparison of the schemes discussed (see text)

3.4 Generalized header rows

Both of these schemes fix the flaws in the original by adding several header rows in the "row data" section of the memristor array. If we want to maintain the split between the "row data" and "column data" sections, these header rows, and, in general, all the rows in the row data section, must each contain at most one 1, since we may otherwise have indistinguishable columns in the row data section, which is the same issue as in the original scheme.

It may be worthwhile to consider a scheme where we store all data row-wise, where each row is a copy of one of a set of header rows. In particular, in an $m \times n$ array with h header rows, where $h \leq n$, we have m - h data rows, each of which can have one of h values, for a total of $(m - h) \log h$ bits. For square arrays, the optimal value of h is plotted in Figure 3.4.4.



Figure 3.4.4: Optimal number of header rows for square arrays as a function of array dimension

If we examine the number of bits that can be achieved with this header row scheme as a fraction of the maximum, we obtain the results in figure 3.4.5. That is, this graph shows

$$\max_{h} \frac{(m-h)\log h}{\log T(m,m)} \tag{3.5}$$

We see that this is a fairly small fraction, and appears to approach a value less than 0.5 asymptotically. Note moreover that this is an upper bound, as we would practically need to truncate h to powers of 2 so that the relevant numbers of bits are integers. We may add a few bits by permuting the header rows, as above, but this makes a negligible difference as m increases, with benefits well below 1% when m is greater than 80.

3.5 One-hot rows

Probably the simplest scheme for reading and writing is a "one-hot" scheme. In such a scheme, each row contains exactly one 1. This is simple to read, as a single read operation extracts one bit. For instance, if the array has $n = 2^k$ columns for some integer k, we



Figure 3.4.5: Maximum fraction of available bits using basic header row scheme

can store $\log n = k$ bits in a row, and extract one bit per read by checking whether the 1 appears in one of the corresponding columns. For example, if we want the third most significant bit, we test all columns whose index has a 1 in the third most significant bit of its binary representation, that is, columns 4, 5, 6, 7, 12, 13, 14, 15,

Writing is similarly straightforward. If the pattern to be overwritten is known a priori, overwriting involves only flipping those memristors corresponding to the old and new 1s. If the existing pattern is not known or the data is being initialized, the new pattern will be written in k-bit words corresponding to rows of n memristors.

This means that writing is significantly more efficient when the data is known, so in contexts where write speed is a priority we may want to read the current pattern before writing, as this will take $k = \log n$ read and 2 write operations, instead of $2^k = n$ write operations. This choice will depend on the specifics of the array size, read and write times, and the relative wear on the devices of reads and writes, which is a subject for future work.

Each of the *m* rows of this scheme contains $\log n$ bits of information, meaning that the entire array contains $m \log n$ bits of information. This is less than the $(m+n) \log(m+n)$ asymptotic capacity, meaning that for most rectangular geometries this is a scheme to use when read-write complexity is prioritized over storage density.

3.5.1 Geometric dependence

The above expression, $m \log n$, is a larger fraction of the asymptotically capacity-achieving $(m + n) \log(m + n)$ for some values of m and n than others. In particular, if we fix L = m + n and optimize m (equivalently n) to maximize $m \log n$ we obtain results that are numerically very close to those for the $(m + n) \log(m + n) = L \log L$ case. We need to bear in mind that this asymptotic result is only valid when m/n approaches a finite positive value as L increases.

3.6 Optimizing one-hot rows with peripherals

Encoding and decoding for the one-hot scheme are straightforward to implement in peripheral transistor hardware. The total area of this hardware will be proportional to $m \log m + n \log n$. This means that if we let a memristor have unit area, the total area including peripherals of an $m \times n$ device is

$$A_C(m,n) = C(m\log m + n\log n) + mn, \qquad (3.6)$$

where C is the proportionality constant, which depends on the transistor technology. The terms multiplying the C come from the fact that we need to be able to select any of the n rows and any of the m columns. Since this device stores $m \log n$ bits of information, we can consider the area information density, which is

$$\rho_C(m,n) = \frac{m\log n}{A_C(m,n)} = \frac{m\log n}{C(m\log m + n\log n) + mn}.$$
(3.7)

We would like to know the values of m and n that maximize this density for a given parameter C. This gives us the most information-dense array, which can then be repeated over the area to give maximally information-dense devices, assuming that it is possible to route the inputs and outputs to such a device efficiently.

Figure 3.6.6 gives the pairs (m^*, n^*) that maximize $\rho_C(m, n)$ as a function of C. Both appear to grow roughly linearly in C, with some variance above roughly C = 20, which is possibly due to numerical artifacts resulting from the difficulty of solving a large discrete optimization problem.

We see that, as might be expected, m^* is consistently larger than n^* . There are also some integer values of m that do not appear to be optimal for any C.



Figure 3.6.6: The values of m and n that maximize $\rho_C(m, n)$ as a function of C.

3.7 Final notes

In this chapter we have discussed a variety of encoding schemes here. In general, there is often a tradeoff between the number of bits that an encoding scheme can store and the complexity of implementing it. We have also seen a brief analysis of the variation in information density, including peripheral encoding/decoding hardware, for the one scheme with a natural encoding/decoding algorithm (the one-hot encoding scheme).

Chapter 4

Capacity of multi-layer devices

One of the attractive features of memristor arrays is the fact that they are composed of two-terminal devices. This contrasts with transistor-based memories such as flash, as transistors are three-terminal devices, and therefore more both more complicated in terms of connectivity and usually larger. This is a useful feature because it reduces the number of connectors needed, and particularly means that all connectors can be placed around the edges of the arrays. This means that memristor arrays can be stacked vertically, producing a very dense device. To maximize density, we will consider devices that alternate layers of wires and memristors without any separating layers.

4.1 Device representation

We generalize our graph and matrix representations for single-layer architectures to the multi-layer devices. We identify a device architecture by the number of layers of memristors, rather than the number of layers of wires, so that the architecture examined in previous chapters is a single-layer architecture, and the architecture examined in this chapter are multi-layer devices.

4.1.1 Matrix representation

In the single-layer case, we used biadjacency matrices \mathbf{D} and \mathbf{F} to represent the locations of high-conductance memristors and the overall connectivity of the architecture, respectively. In cases with more than two layers of wires, a biadjacency matrix is not a valid representation, so we can generalize to adjacency matrices \mathbf{A} and \mathbf{B} .

Toward a new representation, consider the adjacency matrices for the single-layer case. Define the adjacency matrix \mathbf{A} , corresponding to \mathbf{F} , as follows. Matrix \mathbf{A} is an

 $(m+n) \times (m+n)$ matrix with the first *m* indices corresponding to a layer of *m* wires, and the last *n* indices corresponding to a layer of *n* wires. The value $\mathbf{A}_{ij} = 1$ if there is a high-conductance memristor connecting the wire corresponding to index *i* to the wire corresponding to index *j*, and $\mathbf{A}_{ij} = 0$ otherwise. Note that this means

$$\mathbf{A} = \begin{bmatrix} \mathbf{0}_{m \times m} & \mathbf{F} \\ \mathbf{F}^T & \mathbf{0}_{n \times n} \end{bmatrix}$$
(4.1)

The reachability matrix \mathbf{B} , corresponding to \mathbf{D} , describes the overall connectivity of the device. We can define \mathbf{B} analogously to (2.1) as follows.

Definition 4.1.1. The reachability matrix is

$$\mathbf{B} \equiv \left[\sum_{k=1}^{m+n} \mathbf{A}^k\right]_{>0} \tag{4.2}$$

where the terms in the sum correspond to paths of different lengths.

Now consider the ℓ -layer case, with $\ell + 1$ layers of wires, where there are n_k wires in the kth layer, $1 \le k \le \ell + 1$. We define the adjacency matrix as follows.

Definition 4.1.2. The matrix \mathbf{A} is a square matrix of dimension $\sum_k \ell_k$. Each index corresponds to a wire in the device. As in the single-layer case, \mathbf{A}_{ij} is 1 when the wires corresponding to indices i and j are directly connected by a high-conductance memristor.

For example, consider a four-layer case. We can think of this as four single-layer cases that share a sets of wires in between. These four single-layer cases will have **D** matrices \mathbf{D}_1 , \mathbf{D}_2 , \mathbf{D}_3 , and \mathbf{D}_4 . The **A** matrix will then have the form

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{D}_{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{D}_{1}^{T} & \mathbf{0} & \mathbf{D}_{2} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_{2}^{T} & \mathbf{0} & \mathbf{D}_{3} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{D}_{3}^{T} & \mathbf{0} & \mathbf{D}_{4} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{D}_{4}^{T} & \mathbf{0} \end{bmatrix}$$
(4.3)

with the zero matrices of the appropriate dimensions. In general, a layer of wires can only be directly connected to the layers immediately above or below it, which means that, in this block form, there are zero matrices everywhere except in the blocks immediately above and below the main diagonal. Note also that since this \mathbf{A} an undirected adjacency matrix, it is symmetric.

4.1.2 Graph representation

The graph representation used previously generalizes nicely to multi-layer devices. Instead of using a bipartite graph, we have an $(\ell + 1)$ -partite graph, with each part corresponding to one layer of wires in the array.

4.2 Identifying sneak paths

In the single-layer case, we can go from the \mathbf{F} matrix to the \mathbf{D} matrix, or equivalently the \mathbf{A} matrix to the \mathbf{B} matrix by identifying "Ls" in the matrix, that is, rectangles of entries which have three corners equal to 1 and the fourth equal to 0. We cannot use the same process in the multi-layer case. Consider the \mathbf{A} matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & | 1 & 0 & | 0 & 0 \\ 0 & 0 & | 1 & 0 & | 0 & 0 \\ \hline 1 & 1 & 0 & 0 & | 1 & 1 \\ 0 & 0 & 0 & 0 & | 1 & 1 \\ \hline 0 & 0 & | 1 & 1 & | 0 & 0 \\ 0 & 0 & | 1 & 1 & | 0 & 0 \end{bmatrix}$$
(4.4)

which has corresponding **B** matrix

where the interior lines divide the matrix into the regions corresponding to each layer of wires (i.e., this is a $3 \times 3 \times 3$ array). We see that there are many sneak paths, despite there being no "Ls" in the **A** matrix.

This happens because we can have sneak paths from layer to layer in the multi-layer case (as between wire 1 and wire 4 in this case). The "L" rule works inside the block corresponding to a single layer, but since these blocks are distributed above and below the main blockwise diagonal, with zeros everywhere else, they are not caught by the "L" rule.

We can modify **A** so that we can still use the "L" rule by adding an identity matrix

of the same size, which will generate the necessary "L"s. First let us examine how this changes \mathbf{B} .

Theorem 4.2.1. Define $\mathbf{C} \equiv \mathbf{A} + \mathbf{I}$ and $\mathbf{G} \equiv \left[\sum_{k=1}^{m+n} \mathbf{C}^k\right]_{>0}$. It follows that $\mathbf{B} = \mathbf{G}$ except possibly along the diagonal. Moreover, the matrices differ at diagonal entry (i, i) if and only if row (or column) i of \mathbf{A} is zero.

Proof. Call **G** the *reachability matrix with self-loops* (this proof will show why this name is appropriate).

First examine $\mathbf{C}^k \equiv (\mathbf{A}+\mathbf{I})^k$. Since \mathbf{A} is symmetric, this expands to $\mathbf{C}^k = \sum_{j=0}^k {k \choose j} \mathbf{A}^j$, where, as usual, $\mathbf{A}^0 = \mathbf{I}$. Note that $[q\mathbf{M}]_{>0} = [\mathbf{M}]_{>0}$ for any positive scalar q.

Combining these facts implies that

$$\mathbf{G} = \left[\sum_{k=1}^{m+n} \sum_{j=0}^{k} \binom{k}{j} \mathbf{A}^{j}\right]_{>0}$$
(4.6)

$$= \left[\sum_{k=1}^{m+n} \sum_{j=0}^{k} \mathbf{A}^{j}\right]_{>0} = \left[\sum_{k=0}^{m+n} \mathbf{A}^{j}\right]_{>0}$$
(4.7)

$$= \left[\mathbf{I} + \sum_{k=1}^{m+n} \mathbf{A}^k \right]_{>0} \tag{4.8}$$

The final expression above differs from the expression for **B** only in the identity, which gives that $\mathbf{B} = \mathbf{G}$ except possibly along the diagonal, as desired. It remains to prove that the matrices differ at diagonal entry (i, i) if and only if row i of **A** is zero.

Examine entries where **G** and **B** differ. These will be entries on the diagonal where all powers \mathbf{A}^k are zero. The entry \mathbf{A}_{ii} is zero if and only if there are no paths from wire *i* back to itself, which means that wire *i* has no neighbours, or equivalently row *i* of **A** is all zero. This completes the proof.

Graphically, we are making each graph node self-adjacent, which will not affect the subsequent analysis, since we cannot have a sneak path from a node to itself. Because of this, we may use the "L" rule to identify sneak paths as before. It is also still equivalent to saying that every pair of rows (or columns) in **G** must have either all 1s in common or no 1s in common by applying the arguments from Theorem 2.2.1.

It remains to show that we can find all sneak-paths using the "L" rule if we start from \mathbf{G} instead of from \mathbf{B} . First we need to introduce some definitions.

Definition 4.2.1. The single-step rectangular closure of a binary matrix M is the matrix

 $\mathcal{C}_1(\mathbf{M})$ defined by

$$\mathcal{C}_{1}(\mathbf{M})[r,c] = \begin{cases} 1 & \text{if } \mathbf{M}[r,c] = 1\\ 1 & \text{if } \exists r',c'\text{s.t.}\mathbf{M}[r,c'] = \mathbf{M}[r',c'] = \mathbf{M}[r',c] = 1\\ 0 & \text{otherwise} \end{cases}$$
(4.9)

Definition 4.2.2. The k-step rectangular closure of a binary matrix \mathbf{M} is the result of taking the single-step rectangular closure k times. It is denoted $C_k(\mathbf{M})$.

Definition 4.2.3. The rectangular closure of a binary matrix M is

$$\mathcal{C}(\mathbf{M}) \equiv \lim_{k \to \infty} \mathcal{C}_k(\mathbf{M}). \tag{4.10}$$

Note that this limit will exist, since by repeatedly taking rectangular closures we will always reach a point where no more closures can be taken, even if this means filling the entire matrix with 1s. In particular, we will need to take no more than $\prod_{i=0}^{\ell} n_i n_{i+1}$ closures, which is the number of memristors in the device.

We can now state the following theorem.

Theorem 4.2.2. For a given connectivity matrix \mathbf{A} , the reachability matrix equals the rectangular closure of the connectivity matrix with self-loops, that is

$$\mathbf{G} = \mathcal{C}(\mathbf{A} + \mathbf{I}) \tag{4.11}$$

Proof. We will show first that any reachability matrix is a rectangular closure matrix, and then that any rectangular closure matrix is a reachability matrix.

Assume that matrix **G** has a 1 at some index, that is, there exist r and c such that $\mathbf{G}(r, c) = 1$. We want to show that $\mathcal{C}(\mathbf{A} + \mathbf{I})(r, c) = 1$, which means there exists some k such that $\mathcal{C}_k(\mathbf{A} + \mathbf{I})(r, c) = 1$.

The assumption that $\mathbf{G}(r,c) = 1$ means that there exists some minimal $k \ge 0$ such that $[\mathbf{A}^k(r,c)]_{>0} = 1$. Remembering that \mathbf{A} is an adjacency matrix, this means that there is some path of minimal length of connections between wire r and wire c which we can write as $\pi = [p_0, p_1, \ldots, p_k]$ where $p_0 = r$ and $p_k = c$. This path is a list of all the wires we traverse to get from r to c. This means that $\mathbf{A}(p_{i-1}, p_i) = 1$ for $1 \le i \le k$ when k > 0. When k = 0, then $\mathbf{A}^0(p_0, p_0) = \mathbf{I}(p_0, p_0)$ trivially.

We proceed by induction on k. We have already proved the k = 0 case, which is sufficient for a base case. To gain intuition, consider the k = 1 and k = 2 cases. When k = 1, our path is $\pi = [p_0, p_1] = [r, c]$. We know that $\mathbf{A}(p_0, p_1) = 1$, so we have

$$\mathcal{C}(\mathbf{A} + \mathbf{I})(p_0, p_1) = 1 \tag{4.12}$$

since the closure has 1s wherever A has 1s.

When k = 2 we have $\pi = (p_0, p_1, p_2)$. We know that

$$(\mathbf{A} + \mathbf{I})(p_0, p_1) = (\mathbf{A} + \mathbf{I})(p_1, p_1) = (\mathbf{A} + \mathbf{I})(p_1, p_2) = 1$$
 (4.13)

so by definition, $C_1(\mathbf{A} + \mathbf{I})(p_0, p_2) = 1$, so $C(\mathbf{A} + \mathbf{I})(p_0, p_2) = 1$.

Now assume that $C(\mathbf{A} + \mathbf{I})(r', c') = 1$ for any path π_k of length k beginning at some r' and ending at some c'. We want to show that for any path π_{k+1} of length k + 1 it is true that $C(\mathbf{A} + \mathbf{I})(p_0, p_{k+1}) = 1$. The first k elements of π_{k+1} form a path of length k from r to some p_k , so we have $C_i(\mathbf{A} + \mathbf{I})(r, p_k) = 1$ for some i. We also know that $(\mathbf{A} + \mathbf{I})(p_k, p_{k+1}) = 1$. Thus,

$$\mathcal{C}_i(\mathbf{A} + \mathbf{I})(r, p_k) = \mathcal{C}_i(\mathbf{A} + \mathbf{I})(p_k, p_{k+1}) = \mathcal{C}_i(\mathbf{A} + \mathbf{I})(p_k, p_k) = 1.$$
(4.14)

This means that $C_{i+1}(\mathbf{A} + \mathbf{I})(r, c) = 1$, so $C(\mathbf{A} + \mathbf{I})(r, c) = 1$ as desired.

We have shown that if the reachability matrix has a 1 at some index, the rectangular closure has a 1 at the same location. To complete the proof that these matrices are equal we will show that if the rectangular closure has a 1 at some location then so does the reachability matrix.

Assume that $C(\mathbf{A} + \mathbf{I})(r, c) = 1$ for some indices (r, c). If $(\mathbf{A} + \mathbf{I})(r, c) = 1$, then $\mathbf{G}(r, c) = 1$ and we are done. Otherwise, there exists some k such that $C_{k-1}(\mathbf{A} + \mathbf{I})(r, c) = 0$ and $C_k(\mathbf{A} + \mathbf{I})(r, c) = 1$, where we define $C_0(\mathbf{A} + \mathbf{I}) = \mathbf{A} + \mathbf{I}$.

Again, we proceed by induction on k. If k = 1 then $(\mathbf{A} + \mathbf{I})(r, c) = 0$ but

$$\mathcal{C}_1(\mathbf{A} + \mathbf{I})(r, c) = 1 \tag{4.15}$$

meaning that there exists a pair (r', c') such that

$$(\mathbf{A} + \mathbf{I})(r, c') = (\mathbf{A} + \mathbf{I})(r', c') = (\mathbf{A} + \mathbf{I})(r', c) = 1.$$
 (4.16)

This means that [r, c', r', c] forms a path in $\mathbf{A} + \mathbf{I}$, so $\mathbf{G}(r, c) = 1$ as desired.

Now assume that if $C_k(\mathbf{A} + \mathbf{I})(r, c) = 1$ then $\mathbf{G}(r, c) = 1$ for some k. We want to show that if $C_{k+1}(\mathbf{A} + \mathbf{I})(r, c) = 1$ when $C_k(\mathbf{A} + \mathbf{I})(r, c) = 0$ then $\mathbf{G}(r, c) = 1$. By definition, if $C_{k+1}(\mathbf{A} + \mathbf{I})(r, c) = 1$, there must exist (r', c') such that

$$\mathcal{C}_k(\mathbf{A} + \mathbf{I})(r, c') = \mathcal{C}_k(\mathbf{A} + \mathbf{I})(r', c') = \mathcal{C}_k(\mathbf{A} + \mathbf{I})(r', c) = 1.$$
(4.17)

However, by the inductive assumption, this means that $\mathbf{G}(r, c') = \mathbf{G}(r', c'), \mathbf{G}(r', c) = 1$. Hence, [r, c', r', c] forms a path in the graph corresponding to $\mathbf{A} + \mathbf{I}$. By construction, the endpoints of any path in this graph correspond to a 1 in \mathbf{G} , so $\mathbf{G}(r, c) = 1$ as desired.

We have shown that **G** has a 1 at some location if and only if $C(\mathbf{A} + \mathbf{I})$ has a 1 at that location. Since these are binary matrices, they must be equal.

4.3 Counting data patterns

As in the single-layer case, in order to understand a multi-level memristor architecture in an information theoretic context it will be convenient to understand how many distinguishable data patterns may be stored in it. Unlike in the single-layer case, however, it is not true that any rectangular closure matrix of the appropriate dimensions will correspond to a physical device. This can be seen with the help of the following example. Consider the adjacency matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
(4.18)

which has rectangular closure

$$C(\mathbf{A} + \mathbf{I}) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$
 (4.19)

This is a matrix with no "L"s, but it is not physically realizable as it includes connections from the first layer to the third layer that do not pass through the second. This is physically impossible. Indeed, the number of rectangularly closed matrices for a set $\{n_i\}_{i=1}^{L}$ of wire layer dimensions gives us a fairly loose upper bound on the number of data patterns which may be physically stored in a device with these dimensions.

To find an exact expression for the *L*-layer case, we will need the following definitions.

Definition 4.3.1. A (k, s) pattern is a pattern in which the n nodes of the top layer are partitioned into k non-trivial connected components, and s disconnected nodes.

In general, there is more than one (k, s) device for a given pair (k, s). In fact, there are $\binom{n}{s}$ ways of choosing the disconnected nodes and $\binom{n-s}{k}$ ways of partitioning the remaining nodes. This gives us the following alternative expression for $T_2(n_0, n_1)$.

Theorem 4.3.1. The number of distinguishable single-layer memristor patterns is

$$T_1(n_0, n_1) = \sum_{s=0}^{n_1} \sum_{k=0}^{\min(n_0, n_1 - s)} {n_0 + 1 \\ k + 1} {n_1 - s \\ k} {n_1 - s \\ k} {n_1 \choose s} k!.$$
(4.20)

Equation 4.20 follows either from the previous remarks, or from the Stirling number identity

$$\binom{n+1}{k+1} = \sum_{j=k}^{n} \binom{n}{j} \binom{j}{k}.$$

$$(4.21)$$

Now let us consider the two-layer case. In particular, let us see how many two-layer devices have a bottom layer with a (k_1, s_1) architecture.

For example, suppose we had $n_0 = 7$, $n_1 = 6$, and $n_2 = 7$, with $s_1 = 3$, and $k_1 = 3$. One possible device with these parameters is shown in Figure 4.3.1.

In our example, in the base single-layer device, $n_1 - s_1 = 3$ of the wires in layer 1 are partitioned into $k_1 = 2$ connected components, and the remaining $s_1 = 3$ are not connected to lower layers. We will consider these two classes of components separately when adding the new layer. In particular, we can divide the new upper layer of wires (layer 2) into 3 types of components as follows. There will be k'_2 connected components that correspond to a subset of the previous layer's k_1 connected components. There will be a further k''_2 connected components corresponding to some subset of the s_1 unconnected wires in the previous layer. Finally, there will be s_2 disconnected wires. In our example, $k'_2 = 1$, $k''_2 = 1$, and $s_2 = 3$.

This means that there are $k'_2 + k''_2$ connected components that include wires in wire layers 1 and 2. In wire layer 1, there are $\binom{k_1}{k'_2}$ ways to choose the k'_2 components and $\binom{s_1+1}{k''_2+1}$ ways to create the k''_2 components. In wire layer 2 there are $\binom{n_2}{s_2}$ ways to choose the s_2 wires that will not be used, and $\binom{n_2-s_2}{k'_2+k''_2}$ ways to partition the remaining wires



Figure 4.3.1: A (non-unique) legal connection pattern for which $n_0 = 7$, $n_1 = 6$, and $n_2 = 7$, with $s_1 = 3$, $s_2 = 3$, $k_1 = 2$, $k_2 = 2$, and $j_2 = 1$. In this notation, all nodes in pairs of boxes connected by lines with dots are connected to each other.

into $k'_2 + k''_2$ connected components. Finally, there are $(k'_2 + k''_2)!$ ways of pairing the components thus created.

Combining all of this gives us the following theorem.

Theorem 4.3.2. The number of distinguishable 2-layer memristor patterns with a given (k_1, s_1) device as the lower layer is

$$\sum_{s_2=0}^{n_2} \sum_{k'_2=0}^{k_1} \sum_{k''_2=0}^{\min(s_1,n_2)} \binom{k_1}{k'_2} \binom{s_1+1}{k''_2+1} \binom{n_2-s_2}{k'_2+k''_2} \binom{n_2}{s_2} (k'_2+k''_2)!$$
(4.22)

This can be simplified if we define $k_2 = k'_2 + k''_2$ and $j_2 = k'_2$. Such a simplification gives the expression

$$\sum_{s_2=0}^{n_2} \sum_{j_2=0}^{k_1} \sum_{k_2=j_2}^{\min(j_2+s_1,n_2-s_2)} \binom{k_1}{j_2} \binom{s_1+1}{k_2-j_2+1} \binom{n_2-s_2}{k_2} \binom{n_2}{s_2} k_2!$$
(4.23)

For ease of notation, let us define the following quantity.

Definition 4.3.2. Define $N_{n_2}(s_1, s_2, k_1, k_2, j_2)$ to be the number of devices corresponding to this set of parameters, namely

$$\binom{k_1}{j_2} \binom{s_1+1}{k_2-j_2+1} \binom{n_2-s_2}{k_2} \binom{n_2}{s_2} k_2!$$
(4.24)

Finally, this means that we now can write an expression for the number of distinguishable 2-layer patterns.

Theorem 4.3.3. The number of distinguishable 2-layer patterns $T_2(n_0, n_1, n_2)$ is

$$\sum_{s_1=0}^{n_1} \sum_{k_1=0}^{\min(n_0,n_1-s_1)} \sum_{s_2=0}^{n_2} \sum_{j_2=0}^{k_1} \sum_{k_2=j_2}^{\min(j_2+s_1,n_2-s_2)} \left(\binom{n_0+1}{k_1+1} \binom{n_1-s_1}{k_1} \binom{n_1}{s_1} k_! \right) N_{n_2}(s_1,s_2,k_1,k_2,j_2)$$

$$(4.25)$$

Note that the term in brackets looks a bit like N_{n1} . In fact, if we define $s_0 \equiv n_0$, $k_0 \equiv 0$ and $j_1 \equiv 0$, we can write $\binom{n_0+1}{k_1+1} = \binom{k_0}{j_1} \binom{s_0+1}{k_1-j_1+1}$, which means that the term in brackets is exactly $N_{n_1}(s_0, s_1, k_0, k_1, j_1)$. This means that we can write the above expression somewhat more compactly as follows.

Theorem 4.3.4. The number of distinguishable 2-layer patterns $T_2(n_0, n_1, n_2)$ is

$$\sum_{s_1=0}^{n_1} \sum_{k_1=0}^{\min(j_1+s_0,n_1-s_1)} \sum_{s_2=0}^{n_2} \sum_{j_2=0}^{k_1} \sum_{k_2=j_2}^{\min(j_2+s_1,n_2-s_2)} N_{n_1}(s_0,s_1,k_0,k_1,j_1) N_{n_2}(s_1,s_2,k_1,k_2,j_2)$$
(4.26)

This motivates the following theorem.

Theorem 4.3.5. The number of distinguishable L-layer patterns $T_L(\{n_i\}_{i=0}^L)$ is

$$\sum_{s_1=0}^{n_1} \sum_{j_1=0}^{k_0} \sum_{k_1=j_1}^{\min(j_1+s_0,n_1-s_1)} \cdots \sum_{s_L=0}^{n_L} \sum_{j_L=0}^{k_{L-1}} \sum_{k_L=j_L}^{\min(j_L+s_{L-1},n_L-s_L)} \prod_{i=1}^L N_{n_i}(s_{i-1},s_i,k_{i-1},k_i,j_i) \quad (4.27)$$

Proof. The proof is by induction. We have proved the base cases L = 1 and L = 2 above. Now assume this is true for L = m. We want to show it is true for L = m + 1.

Consider the bottom m layers of an m + 1-layer device as an m-layer device. This is a (k_m, s_m) pattern, and by the inductive hypothesis, we know how many such patterns there are. As in the proof for the 2-layer case, we will add a new wire layer with two categories of connected components: k_{m+1} connected components that connect to some lower wires, and s_{m+1} disconnected wires. Of the k_{m+1} connected components, j_{m+1} of

them connect to subsets of the s_m wires in wire layer m that are disconnected from all lower wires.

We want to find out how many ways we can create k_{m+1} connected components at wire layers m and m+1. At layer m, j_{m+1} of the components are chosen from the existing k_m connected components, and the other $k_{m+1} - j_{m+1}$ are formed from a partition of the remaining s_m wires, for a total of $\binom{k_m}{j_{m+1}} \begin{Bmatrix} s_m+1 \\ k_{m+1}-j_{m+1}+1 \end{Bmatrix}$ ways of creating k_{m+1} connected components. At layer m+1, there are s_{m+1} disconnected wires, and the remaining n_{m+1} s_{m+1} wires are partitioned into k_{m+1} components. This can be done in $\binom{n_{m+1}-s_{m+1}}{k_{m+1}}\binom{n_{m+1}}{s_{m+1}}$ ways. The k_{m+1} connected components in the two wire layers can be paired in k_{m+1} ! different ways. These factors all multiply to form $N_{n_{m+1}}(s_m, s_{m+1}, k_m, k_{m+1}, j_{m+1})$.

Since $T_m(\{n_i\}_{i=0}^m)$ equals

$$\sum_{s_1=0}^{n_1} \sum_{j_1=0}^{k_0} \sum_{k_1=j_1}^{\min(j_1+s_0,n_1-s_1)} \cdots \sum_{s_m=0}^{n_m} \sum_{j_m=0}^{k_{m-1}} \sum_{k_m=j_m}^{\min(j_m+s_{m-1},n_m-s_m)} \prod_{i=1}^m N_{n_i}(s_{i-1},s_i,k_{i-1},k_i,j_i) \quad (4.28)$$

there are

$$\sum_{s_1=0}^{n_1} \sum_{j_1=0}^{k_0} \sum_{k_1=j_1}^{\min(j_1+s_0,n_1-s_1)} \cdots \sum_{s_m=0}^{n_m} \sum_{j_m=0}^{k_{m-1}} \sum_{k_m=j_m}^{\min(j_m+s_{m-1},n_m-s_m)} \sum_{j_{m+1}=0}^{k_m} \prod_{i=1}^{m+1} N_{n_i}(s_{i-1},s_i,k_{i-1},k_i,j_i)$$

$$(4.29)$$

different (k_{m+1}, s_{m+1}) patterns, where we have multiplied the summand by

$$N_{n_{m+1}}(s_m, s_{m+1}, k_m, k_{m+1}, j_{m+1})$$
(4.30)

and summed over j_{m+1} . Summing over k_{m+1} , and s_{m+1} gives

$$\sum_{s_1=0}^{n_1} \sum_{j_1=0}^{k_0} \sum_{k_1=j_1}^{\min(j_1+s_0,n_1-s_1)} \cdots \sum_{s_{m+1}=0}^{n_{m+1}} \sum_{j_{m+1}=0}^{k_m} \sum_{k_{m+1}=j_{m+1}}^{\min(j_{m+1}+s_m,n_{m+1}-s_{m+1})} \prod_{i=1}^{m+1} N_{n_i}(s_{i-1},s_i,k_{i-1},k_i,j_i),$$
(4.31)
the desired result.

the desired result.

This approach to counting tells us that we can count the different arrays by first classifying all L - 1-layer patterns as (k_{L-1}, s_{L-1}) devices, and then summing over all possible ways that each of these device types can lead to each of the (k_L, s_L) device types.

This gives us a general expression for the number of distinguishable patterns that we can store in a multilayer memristor array. The logarithm (base 2) of this value gives us the capacity of such an architecture in bits.

4.4 Analysis

We can see that layering memristors in this way will give us more capacity per unit area than if we were to layer discrete single-layer devices alternating with layers of insulator. This is because layering single-layer devices in this way is equivalent to setting all memristors on every second layer in the high-resistance state to the connected multi-layer case.

The expression for $T_L(\{n_i\}_{i=0}^L)$ grows very quickly in L, as it includes two more Stirling number factors, two more binomial factors, and two more factorial factors for every unit increase in L. All of these factors are roughly exponential. For this reason, we can only analyze fairly small multi-layer arrays exactly. However, even for such small arrays, we can already see the benefits of layering compared to stacking single-layer devices. Figure 4.4.2 shows this for devices having every dimension equal to 5. We see that $\log T_L$ grows much more quickly than $\lfloor L/2 \rfloor \log T_1$ for these small arrays, as expected. In the general case when comparing across area we would develop a density metric as explored in Section 2.3.7. However, in this case plotted, the dimensions of all layers are the same, which means that the devices will have similar footprints when calculating densities.



Figure 4.4.2: Comparison between capacity of multi-layer device and layered single-layer devices, where $n_i = 5$ for all *i*.

Chapter 5

Conclusions and future work

5.1 Single-layer devices

Single-layer memristor storage devices were well modelled in the existing literature prior to our work. The present work provides an alternative proof for the capacity of these devices, and links their study to the study of difunctional relations, which is useful when extending the analysis to the multi-layer case. Moreover, we explored some improvements to existing encoding schemes, which both avoid a flaw in one of the existing schemes, and, in the case of the one-hot encoder, lend themselves well to straightforward encoding and decoding.

There are many possible extensions to this work. For instance, in the present model, we have been assuming that all resistances are either zero or infinite, and all applied voltages are equal. If we generalize either of these assumptions to allow intermediate resistances or voltages, we introduce an interesting new set of challenges, as well as a variety of new device parameters. In particular, we can start to consider the important consideration of the power consumed by a device, especially a multi-layer device, in which the physical problem of heat dissipation will be of particular concern. This is discussed to some extent by Cassuto in [2], and is important to consider when implementing a device physically.

In addition to having non-ideal resistances, it may also be interesting to consider modelling memristors that may be reliably programmed to more than two values. Although this programming is often unstable physically, some promising work in the materials science of such a device has been done in, for instance, [39].

5.2 Multi-layer devices

Our discussion of multi-layer develops novel combinatorial results describing the capacity of such devices. Moreover, this discussion suggests that if such a device can be constructed it would be worthwhile to consider as a memory device. In any case, many of the discussions here about single-layer devices bear consideration in the multi-layer case. In particular, it would be useful to develop encoding and decoding algorithms for these devices.

Bibliography

- F. Pan, S. Gao, C. Chen, C. Song, and F. Zeng, "Recent progress in resistive random access memories: Materials, switching mechanisms, and performance," *Materials Science and Engineering: R: Reports*, vol. 83, pp. 1–59, Sep. 2014.
- Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Sneak-path constraints in memristor crossbar arrays," in 2013 IEEE International Symposium on Information Theory, Jul. 2013, pp. 156–160.
- [3] L. O. Chua, "Memristor—the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. CT-18, no. 5, pp. 507–519, Sep. 1971.
- [4] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, May 2008.
- [5] S. Long, L. Perniola, C. Cagli, J. Buckley, X. Lian, E. Miranda, F. Pan, M. Liu, and J. Suñé, "Voltage and power-controlled regimes in the progressive unipolar reset transition of HfO₂-based RRAM," *Scientific Reports*, vol. 3, p. 2929, Oct. 2013.
- [6] D. Berco and T.-Y. Tseng, "A comprehensive study of bipolar operation in resistive switching memory devices," *Journal of Computational Electronics*, vol. 15, no. 2, pp. 577–585, Jun. 2016.
- [7] S. Ambrogio, S. Balatti, D. C. Gilmer, and D. Ielmini, "Analytical modeling of oxide-based bipolar resistive memories and complementary resistive switches," *IEEE Transactions on Electron Devices*, vol. 61, no. 7, pp. 2378–2386, Jul. 2014.
- [8] K. M. Kim, B. J. Choi, M. H. Lee, G. H. Kim, S. J. Song, J. Y. Seok, J. H. Yoon, S. Han, and C. S. Hwang, "A detailed understanding of the electronic bipolar resistance switching behavior in Pt/TiO₂/Pt structure," *Nanotechnology*, vol. 22, no. 25, p. 254 010, Jun. 2011.

- [9] K. Nagashima, T. Yanagida, K. Oka, M. Kanai, A. Klamchuen, J.-S. Kim,
 B. H. Park, and T. Kawai, "Intrinsic mechanisms of memristive switching," *Nano Letters*, vol. 11, no. 5, pp. 2114–2118, May 2011.
- [10] S. Kosta, Y. Kosta, M. Bhatele, Y. Dubey, A. Gaur, S. Kosta, J. Gupta, A. Patel, and B. Patel, "Human blood liquid memristor," *International Journal of Medical Engineering and Informatics*, vol. 3, pp. 16–29, 1 2011.
- [11] E. Gale, A. Adamatzky, and B. de Lacy Costello, "Slime mould memristors," *BioNanoScience*, vol. 5, no. 1, pp. 1–8, Mar. 2015.
- [12] L. Goux and S. Spiga, "Unipolar resistive-switching mechanisms," in *Resistive Switching*. Wiley-VCH Verlag, Jan. 2016, pp. 363–394.
- P. Huang, X. Y. Liu, W. H. Li, Y. X. Deng, B. Chen, Y. Lu, B. Gao, L. Zeng,
 K. L. Wei, G. Du, X. Zhang, and J. F. Kang, "A physical based analytic model of RRAM operation for circuit simulation," in 2012 International Electron Devices Meeting, Dec. 2012, pp. 26.6.1–26.6.4.
- S. Long, C. Cagli, D. Ielmini, M. Liu, and J. Suñé, "Analysis and modeling of resistive switching statistics," *Journal of Applied Physics*, vol. 111, no. 7, p. 074508, Apr. 2012.
- [15] B. Gao, B. Sun, H. Zhang, L. Liu, X. Liu, R. Han, J. Kang, and B. Yu, "Unified physical model of bipolar oxide-based resistive switching memory," *IEEE Electron Device Letters*, vol. 30, no. 12, pp. 1326–1328, Dec. 2009.
- [16] S. Balatti, S. Ambrogio, D. Ielmini, and D. C. Gilmer, "Variability and failure of set process in HfO₂ RRAM," in 2013 5th IEEE International Memory Workshop, May 2013, pp. 38–41.
- [17] D. Ielmini, S. Balatti, Z. Q. Wang, and S. Ambrogio, "Variability and cycling endurance in nanoscale resistive switching memory," in 2015 IEEE 15th International Conference on Nanotechnology (IEEE-NANO), Jul. 2015, pp. 124–127.
- [18] A. Ghofrani, M.-A. Lastras-Montaño, S. Gaba, M. Payvand, W. Lu,
 L. Theogarajan, and K.-T. Cheng, "A low-power variation-aware adaptive write scheme for access-transistor-free memristive memory," *J. Emerg. Technol. Comput. Syst.*, vol. 12, no. 1, 3:1–3:18, Aug. 2015.

- [19] S. Ambrogio, S. Balatti, A. Cubeta, A. Calderoni, N. Ramaswamy, and D. Ielmini, "Understanding switching variability and random telegraph noise in resistive RAM," in 2013 IEEE International Electron Devices Meeting, Dec. 2013, pp. 31.5.1–31.5.4.
- [20] B. Chen, Q. Y. Jun, B. Gao, F. F. Zhang, K. L. Wei, Y. S. Chen, L. F. Liu, X. Y. Liu, J. F. Kang, and R. Q. Han, "A compact model of resistive switching devices," in 10th IEEE International Conference on Solid-State and Integrated Circuit Technology, Nov. 2010, pp. 1829–1831.
- [21] M. P. Sah, C. Yang, H. Kim, B. Muthuswamy, J. Jevtic, and L. Chua, "A generic model of memristors with parasitic components," *IEEE Transactions on Circuits* and Systems I: Regular Papers, vol. 62, no. 3, pp. 891–898, Mar. 2015.
- [22] A. Ascoli, F. Corinto, and R. Tetzlaff, "Generalized boundary condition memristor model," *International Journal of Circuit Theory and Applications*, vol. 44, no. 1, pp. 60–84, Jan. 2016.
- [23] Y. Lu, B. Gao, Y. Fu, B. Chen, L. Liu, X. Liu, and J. Kang, "A simplified model for resistive switching of oxide-based resistive random access memory devices," *IEEE Electron Device Letters*, vol. 33, no. 3, pp. 306–308, Mar. 2012.
- [24] F. Corinto, P. P. Civalleri, and L. O. Chua, "A theoretical approach to memristor systems," in 2014 14th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA), Jul. 2014, pp. 1–2.
- [25] C. Ma, S. Xie, Y. Jia, and G. Lin, "Macromodeling of the memristor using piecewise volterra series," *Microelectronics Journal*, vol. 45, no. 3, pp. 325–329, Mar. 2014.
- [26] T. Wang and J. S. Roychowdhury, "Well-posed models of memristive devices," CoRR, vol. abs/1605.04897, May 2016.
- [27] S. Sadeque and P. Rahman, "Modeling memristive behavior using drude model," in 2012 7th IEEE Conference on Industrial Electronics and Applications (ICIEA), Jul. 2012, pp. 1201–1206.
- [28] H. Li, Z. Jiang, P. Huang, Y. Wu, H. Y. Chen, B. Gao, X. Y. Liu, J. F. Kang, and H. S. P. Wong, "Variation-aware, reliability-emphasized design and optimization of RRAM using SPICE model," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2015, pp. 1425–1430.
- [29] S. Benderli and T. A. Wey, "On SPICE macromodelling of TiO₂ memristors," *Electronics Letters*, vol. 45, no. 7, pp. 377–379, Mar. 2009.

- [30] A. G. Radwan, M. A. Zidan, and K. N. Salama, "On the mathematical modeling of memristors," in 2010 International Conference on Microelectronics, Dec. 2010, pp. 284–287.
- [31] M. Mahvash and A. C. Parker, "A memristor SPICE model for designing memristor circuits," in 2010 53rd IEEE International Midwest Symposium on Circuits and Systems, Aug. 2010, pp. 989–992.
- [32] D. Batas and H. Fiedler, "A memristor SPICE implementation and a new approach for magnetic flux-controlled memristor modeling," *IEEE Transactions* on Nanotechnology, vol. 10, no. 2, pp. 250–255, Mar. 2011.
- [33] X. Fang, X. Yang, J. Wu, and X. Yi, "A compact SPICE model of unipolar memristive devices," *IEEE Transactions on Nanotechnology*, vol. 12, no. 5, pp. 843–850, Sep. 2013.
- [34] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Generalized memristive device SPICE model and its application in circuit design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 8, pp. 1201–1214, Aug. 2013.
- [35] K. Xu, Y. Zhang, L. Wang, M. Yuan, W. T. Joines, and Q. H. Liu, "Nano-scale memristor SPICE implementation using ideal operational amplifier model," vol. 8911, Aug. 2013, 89110H–89110H-7.
- [36] F. García-Redondo, R. P. Gowers, A. Crespo-Yepes, M. López-Vallejo, and L. Jiang, "SPICE compact modeling of bipolar/unipolar memristor switching governed by electrical thresholds," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 8, pp. 1255–1264, Aug. 2016.
- [37] H. Li, P. Huang, B. Gao, B. Chen, X. Liu, and J. Kang, "A SPICE model of resistive random access memory for large-scale memory array simulation," *IEEE Electron Device Letters*, vol. 35, no. 2, pp. 211–213, Feb. 2014.
- [38] D. Ielmini, F. Nardi, and C. Cagli, "Universal reset characteristics of unipolar and bipolar metal-oxide RRAM," *IEEE Transactions on Electron Devices*, vol. 58, no. 10, pp. 3246–3253, Oct. 2011.
- [39] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu, "A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications," *Nano letters*, vol. 12, no. 1, pp. 389–395, Jan. 2011.

- [40] S. Stathopoulos, A. Khiat, M. Trapatseli, S. Cortese, A. Serb, I. Valov, and T. Prodromakis, "Multibit memory operation of metal-oxide bi-layer memristors," *Scientific Reports*, vol. 7, p. 17532, 1 Dec. 2017.
- [41] P. P. Sotiriadis, "Information capacity of nanowire crossbar switching networks," *IEEE Transactions on Information Theory*, vol. 52, no. 7, pp. 3019–3032, Jul. 2006.
- [42] F. T. Chen, Y.-S. Chen, H.-Y. Lee, W.-S. Chen, P.-Y. Gu, T.-Y. Wu, C.-H. Tsai, Y.-Y. Liao, P.-S. Chen, S.-S. Sheu, P.-F. Chiu, W.-P. Lin, C.-H. Lin, M.-J. Tsai, and T.-K. Ku, "Access strategies for resistive random access memory (RRAM)," *ECS Transactions*, vol. 44, no. 1, pp. 73–78, Mar. 2012.
- [43] A. Ghofrani, M. A. Lastras-Montaño, and K. T. Cheng, "Toward large-scale access-transistor-free memristive crossbars," in 20th Asia and South Pacific Design Automation Conference, Jan. 2015, pp. 563–568.
- [44] K. Eshraghian, K. R. Cho, O. Kavehei, S. K. Kang, D. Abbott, and S. M. S. Kang, "Memristor MOS content addressable memory (MCAM): Hybrid architecture for future high performance search engines," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 8, pp. 1407–1417, Aug. 2011.
- [45] P. Junsangsri and F. Lombardi, "A memristor-based memory cell using ambipolar operation," in 2011 IEEE 29th International Conference on Computer Design (ICCD), Oct. 2011, pp. 148–153.
- [46] B. Mohammad, D. Homouz, O. A. Rayahi, H. Elgabra, and A. S. A. Hosani, "Hybrid memristor-CMOS memory cell: Modeling and design," in *ICM 2011 Proceeding*, Dec. 2011, pp. 1–6.
- [47] X. Li, Z. Chen, Z. Fang, A. Kamath, X. Wang, N. Singh, G.-Q. Lo, and D.-L. Kwong, "Integration of resistive switching memory cell with vertical nanowire transistor," *International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering*, vol. 6, no. 9, pp. 918–920, 2012.
- [48] V. S. Baghel and S. Akashe, "Low power memristor based 7T SRAM using MTCMOS technique," in 2015 Fifth International Conference on Advanced Computing Communication Technologies, Feb. 2015, pp. 222–226.
- [49] P. Junsangsri, J. Han, and F. Lombardi, "A memristor-based memory cell with no refresh," in 14th IEEE International Conference on Nanotechnology, Aug. 2014, pp. 947–950.

- [50] C. Yakopcic, T. M. Taha, and R. Hasan, "Hybrid crossbar architecture for a memristor based memory," in NAECON 2014 - IEEE National Aerospace and Electronics Conference, Jun. 2014, pp. 237–242.
- [51] M.-J. Lee, S. Seo, D.-C. Kim, S.-E. Ahn, D. H. Seo, I.-K. Yoo, I.-G. Baek, D.-S. Kim, I.-S. Byun, S.-H. Kim, I.-R. Hwang, J.-S. Kim, S.-H. Jeon, and B. H. Park, "A low-temperature-grown oxide diode as a new switch element for high-density, nonvolatile memories," *Advanced Materials*, vol. 19, no. 1, pp. 73–76, 2007.
- Y. C. Shin, J. Song, K. M. Kim, B. J. Choi, S. Choi, H. J. Lee, G. H. Kim, T. Eom, and C. S. Hwang, "(In, Sn)₂O₃/TiO₂/Pt Schottky-type diode switch for the TiO₂ resistive switching memory array," *Applied Physics Letters*, vol. 92, no. 16, p. 162 904, Apr. 2008.
- [53] W. Y. Park, G. H. Kim, J. Y. Seok, K. M. Kim, S. J. Song, M. H. Lee, and C. S. Hwang, "A Pt/TiO₂/Ti Schottky-type selection diode for alleviating the sneak current in resistance switching memory arrays," *Nanotechnology*, vol. 21, no. 19, p. 195 201, 2010.
- [54] M. J. Lee, Y. Park, B. S. Kang, S. E. Ahn, C. Lee, K. Kim, W. Xianyu,
 G. Stefanovich, J. H. Lee, S. J. Chung, Y. H. Kim, C. S. Lee, J. B. Park,
 I. G. Baek, and I. K. Yoo, "2-stack 1D-1R cross-point structure with oxide diodes as switch elements for high density resistance RAM applications," in 2007 IEEE International Electron Devices Meeting, Dec. 2007, pp. 771–774.
- [55] K. Zhang, K. Sun, F. Wang, Y. Han, Z. Jiang, J. Zhao, B. Wang, H. Zhang, X. Jian, and H. S. P. Wong, "Ultra-low power Ni/HfO2/TiOx/TiN resistive random access memory with sub-30-nA reset current," *IEEE Electron Device Letters*, vol. 36, no. 10, pp. 1018–1020, Oct. 2015.
- [56] P. O. Vontobel, W. Robinett, P. J. Kuekes, D. R. Stewart, J. Straznicky, and R. S. Williams, "Writing to and reading from a nano-scale crossbar memory based on memristors," *Nanotechnology*, vol. 20, no. 42, p. 425 204, 2009.
- [57] J. Liang and H. S. P. Wong, "Cross-point memory array without cell selectors—device characteristics and data storage pattern dependencies," *IEEE Transactions on Electron Devices*, vol. 57, no. 10, pp. 2531–2538, Oct. 2010.
- [58] O. Kavehei, S. Al-Sarawi, K. R. Cho, K. Eshraghian, and D. Abbott, "An analytical approach for memristive nanoarchitectures," *IEEE Transactions on Nanotechnology*, vol. 11, no. 2, pp. 374–385, Mar. 2012.

- [59] E. Linn, R. Rosezin, C. Kugeler, and R. Waser, "Complementary resistive switches for passive nanocrossbar memories," *Nature Materials*, vol. 9, no. 5, pp. 403–406, May 2010.
- [60] Q. Xia, M. D. Pickett, J. J. Yang, X. Li, W. Wu, G. Medeiros-Ribeiro, and R. S. Williams, "Two- and three-terminal resistive switches: Nanometer-scale memristors and memistors," *Advanced Functional Materials*, vol. 21, no. 14, pp. 2660–2665, Jul. 2011.
- [61] I. Vourkas, D. Stathis, G. C. Sirakoulis, and S. Hamdioui, "Alternative architectures toward reliable memristive crossbar memories," *IEEE Transactions* on Very Large Scale Integration (VLSI) Systems, vol. 24, no. 1, pp. 206–217, Jan. 2016.
- [62] M. A. Zidan, A. M. Eltawil, F. Kurdahi, H. A. H. Fahmy, and K. N. Salama, "Memristor multiport readout: A closed-form solution for sneak paths," *IEEE Transactions on Nanotechnology*, vol. 13, no. 2, pp. 274–282, 2014.
- [63] J. Riguet, "Relations binaires, fermetures, correspondances de Galois [Binary relations, closures, Galois connections]," Bulletin de la Société Mathématique de France, vol. 76, pp. 114–155, 1948.
- [64] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Information-theoretic sneak-path mitigation in memristor crossbar arrays," *IEEE Transactions on Information Theory*, vol. 62, no. 9, pp. 4801–4813, Sep. 2016.
- [65] OEIS Foundation Inc., "The On-Line Encyclopedia of Integer Sequences," 2017.[Online]. Available: www.http://oeis.org/A265417.
- [66] C. J. Everett, "Closure operators and Galois theory in lattices," Transactions of the American Mathematical Society, vol. 55, no. 3, pp. 514–525, May 1944.
- [67] A. Jaoua, N. Belkiter, H. Ounalli, and T. Moukam, "Databases," in *Relational Methods in Computer Science*, C. Brink, W. Kahl, and G. Schmidt, Eds., Vienna: Springer-Verlag, 1997, ch. 13, pp. 197–210.