Fault-Tolerant Associative Memories Based on c-Partite Graphs

François Leduc-Primeau, *Student Member, IEEE*, Vincent Gripon, *Member, IEEE*, Michael G. Rabbat, *Senior Member, IEEE*, and Warren J. Gross, *Senior Member, IEEE*

***Abstract*—Associative memories allow the retrieval of previously stored messages given a part of their content. In this paper, we are interested in associative memories based on c-partite graphs that were recently introduced. These memories are almost optimal in terms of the amount of storage they require (efficiency) and allow retrieving messages with low complexity. We propose a generic implementation model for the retrieval algorithm that can be readily mapped to an integrated circuit and study the retrieval performance when hardware components are affected by faults. We show using analytical and simulation results that these associative memories can be made resilient to circuit faults with a minor modification of the retrieval algorithm. In one example, the memory retains 88% of its efficiency when 1% of the storage cells are faulty, or 98% when 0.1% of the binary outputs of the retrieval algorithm are faulty. When considering storage faults, the fault tolerance exhibited by the proposed associative memory can be comparable to using a capacity-achieving error correction code for protecting the stored information.**

***Index Terms*—Associative memory, fault tolerant systems, error correction codes, digital integrated circuits.**

I. Introduction

ASSOCIATIVE memories are devices that are able to retrieve previously stored messages given a part of their content. They are used in a variety of applications ranging from CPU caches [2] to database engines [3], and intrusion detection systems [4].

Implementations differ depending on the domain. In electronics, most systems are based on content-addressable memories (CAM) [5]. CAMs can locate data associated with an input (the key) by comparing it concurrently with all the stored elements. They do not provide as much flexibility as general associative memories because the portion of the data that represents the key and the portion that represents the result are pre-determined. Moreover, because all the keys stored in the memory must be accessed in response to every query, they require a lot of power, and in practice they are limited to a moderate number of elements.

In neuroscience, the most celebrated associative memory model is that proposed by Hopfield [6]. Hopfield neural networks allow storing and retrieving binary messages when any fraction of the stored messages is missing as input. However, they provide a limited diversity (number of messages it is possible to store and then retrieve with high probability as a function of the size of the network) [7]. Therefore, they are unpractical when the number of elements to store is large.

Another influential model was proposed by Willshaw [8]. Used as associative memories, Willshaw networks have a much better efficiency (ratio of information stored to the amount of storage required) than Hopfield networks. Several analytical results have been derived to describe their efficiency (see [9] and references therein). Among these, the so-called learning bound is an asymptotic upper bound that is independent of the retrieval algorithm.

Recently, Gripon and Berrou [10], [11] proposed a novel architecture for associative memories (AM) based on c-partite graphs that is almost optimal in terms of storage efficiency [12]. Moreover, the retrieval complexity is limited, even when the number of stored messages is large. Compared to Willshaw networks, c-partite AMs add an error correction mechanism by ensuring that only one vertex in each of the c subsets is associated with a given stored message. This allows them to achieve a greater efficiency. We give examples in the paper of c-partite AMs with an efficiency higher than the learning bound of Willshaw networks.

In this paper, we consider the performance of c-partite AMs when implemented on unreliable hardware. Considering unreliable hardware is motivated by the fact that, as the feature size of integrated circuits decreases, it is becoming increasingly hard to control the variability associated both with the fabrication process and with the circuit's operating conditions [13], [14]. As a result, larger safety margins must be used to maintain yield and performance guarantees. Fault tolerance provided by the algorithm allows reducing these safety margins, thereby increasing the implementation's efficiency.

Willshaw AMs have been studied under faulty data storage. Notably, a lower bound on efficiency is derived in [15], some simulation results are presented in [16], and [17] derived maximum a-posteriori retrieval rules that take into account the probability of faults in the storage cells. An associative memory that

Manuscript received May 28, 2015; revised September 25, 2015; accepted September 29, 2015. Date of publication October 12, 2015; date of current version January 13, 2016. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. John McAllister. This work was supported in part by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013)/ERC grant agreement No. 290901, and by the Natural Science and Engineering Research Council of Canada. A preliminary version of this work was presented at the IEEE ICASSP 2014.

F. Leduc-Primeau, M. G. Rabbat, and W. J. Gross are with the Department of Electrical and Computer Engineering, McGill University, Montreal, QC H3A 0E9, Canada (e-mail: francois.leduc-primeau@mail.mcgill.ca; michael.rabbat@mcgill.ca; warren.gross@mcgill.ca).

V. Gripon is with the Electronics Department, Télécom Bretagne, Brest 29200, France (e-mail: vincent.gripon@telecom-bretagne.eu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2015.2489600

is similar to the c-partite AM was also proposed in [18]. The efficiency of the memory is discussed, but the fault tolerance of the retrieval algorithm is not considered. Some recent work [19] proposes a new neural network model that is able to store $O(a^n)$ messages, where n is the number of vertices and $a > 1$. This model is also analyzed for the case where the retrieval process is affected by internal noise [20]. However, the model uses continuous weights on edges and therefore makes it impossible to evaluate the efficiency of the memory, which is a crucial metric for digital implementations.

The rest of the paper is organized as follows. We start in Section II by reviewing the neural network model proposed by Gripon and Berrou, and describe the modifications that must be made to the retrieval algorithm to make it resilient to faults. In Section III, we propose a realistic model to account for the effect of hardware faults on the algorithm. We define two categories of faults: first those that affect the storage hardware, and second those that affect the computations of the retrieval algorithm. These two types of faults are treated separately throughout the paper. Based on this model, in Section IV we derive some analytical results for the retrieval performance of a faulty implementation expressed as the probability of retrieving a message in terms of the number of messages stored in the memory, and also present simulation results. In Section V, we define the storage efficiency based on the mutual information between the stored messages and the retrieved messages, and explain how our definition relates to similar definitions in the literature on Willshaw networks. We compare the efficiency achieved by c-partite AMs with several other implementation approaches. In particular, we show that the maximum efficiency of an implementation affected by storage faults can be comparable to the efficiency of a hypothetical implementation where the reliability of the data storage is ensured by the use of a capacity-achieving error-correction code. Finally, Section VI concludes the paper.

II. Review of c-Partite Associative Memories

In this section we review the functioning of the associative memories introduced in [10], [11]. More precisely, we start from the improvement proposed in [12], and make minor modifications to improve the fault-tolerance of the decoding algorithm.

Suppose that we want to store a set $\mathcal{M}$ of messages, each composed of c symbols in the alphabet $\mathcal{A} = \{1, 2, 3, \ldots, \ell\}$. We will then be interested in retrieving a message $m \in \mathcal{M}$ given a partial version of it, that is a copy of m where some of the symbols have been replaced by an erasure symbol $\perp$, $\perp \notin \mathcal{A}$. For example, consider an associative memory that stores messages $112, 221$ and 222. The output associated with $1\perp\perp$ is 112, while the output is ambiguous for input $22\perp$ (in some contexts, the AM could output an arbitrary message from the set of matching messages).

A. Network Representation

To describe the AM's internal representation, let us consider an undirected and unweighted[1] graph made of $c \cdot \ell$ vertices. The graph is partitioned into c independent subsets, each containing the same number ℓ of vertices. We will call these subsets *symbol groups*. Let us then index the groups from 1 to c and vertices in each group from 1 to ℓ. Any vertex in the graph is therefore uniquely identified by a pair (i, j), where $1 \leq i \leq c$ and $1 \leq j \leq \ell$.

[1]If vertex a is connected to vertex b, then vertex b is connected to vertex a, and edges are binary: either they exist or not.

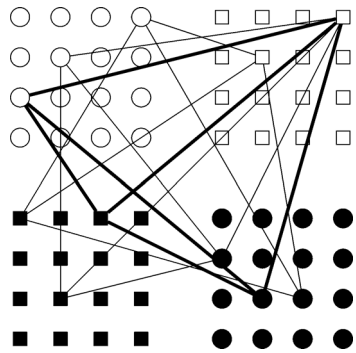

Fig. 1. Storing process illustration reproduced from [10]. Here, messages to store are made of $c = 4$ symbols in the alphabet $\mathcal{A} = \{1, 2, \ldots, 16\}$. The graph is therefore divided into 4 groups (filled circles, filled rectangles, rectangles and circles), each containing 16 vertices. The message currently being stored has added 6 edges to the graph (represented with bold lines), that is all those between the 4 vertices representing the message in the graph.

To represent messages to store in the network, we define a function f that maps a message or a partial message m to a subset of vertices in the graph. The function f is defined as follows:

$$f(m) = \{v_{i,m_i}, 1 \leq i \leq c\}, \tag{1}$$

where $v_{i,j}$ is the j-th vertex in symbol group i and m_i represents the i-th symbol of m. A group corresponding to an erased symbol in the message m is referred to as an "erased group". Note that $f(m)$ does not contain any vertex in erased groups. The function f is invertible, such that to a subset of vertices V_m containing at most one vertex in each symbol group will correspond a unique partial message.

B. Storing Messages

Before storing messages, the graph contains no edges. To store a message $m \in \mathcal{A}^c$ in the graph, we add all the edges between vertices in $f(m)$, with the exception of self-loops. Therefore, $f(m)$ becomes a clique in the graph, that is a subset of vertices fully interconnected. The graph being unweighted, edges that are already in the graph remain unchanged. The storing process is depicted in Fig. 1 using an example graph made of $c = 4$ symbol groups, each containing $\ell = 16$ vertices.

One can test whether a message m has been stored previously by checking if the corresponding clique exists. In a reliable implementation, the probability that a message that was stored previously is not recognized is 0, while the probability of wrongly identifying a message as having been stored is usually very small (see [10], Section V).

C. Retrieving Partially Erased Messages

This associative memory is able to retrieve a previously stored message when only some of its symbols are mapped onto the graph. The retrieval algorithm relies on two basic concepts. Recall that each vertex in a symbol group represents

a possible value of that symbol. First, we say that a vertex is *active* if it has been identified as a possible value of the symbol. Second, we attribute a *score* to each vertex that quantifies the likelihood that it is part of the message being retrieved. When a vertex achieves the maximum score within its symbol group, this indicates that it might correspond to the correct value of the symbol. The retrieval algorithm operates by iteratively computing the scores of vertices, and marking as active the ones that achieve the maximum score, with the hope of converging to a state where only one vertex is active in each symbol group.

Algorithm 1 provides a formal description of the retrieval algorithm, which is adapted from the one described in [12]. The algorithm takes as input a partially erased message $\widetilde{m}$ that is obtained from a message $m \in \mathcal{M}$ by replacing any c_e symbols with $\perp$. Throughout the paper, we use c_e to denote the number of erased symbols, and c_k to denote the number of known symbols. Therefore we have $c = c_e + c_k$. Algorithm 1 is defined in terms of the graphical model. The algorithm is iterative, and its state at iteration $t \in \mathbb{N}$ can be fully described by a set $V_{\widetilde{m}}^{(t)}$, whose initial value is given by $V_{\widetilde{m}}^{(0)} = f(\widetilde{m})$. A vertex $v \in V_{\widetilde{m}}^{(t)}$ is referred to as an *active* vertex at iteration t. We also use the term *spurious* vertex to refer to an active vertex that does not correspond to the correct symbol, that is a vertex v such that $v \in V_{\widetilde{m}}^{(t)}$ and $v \notin f(m)$.

In the algorithm, we denote the subset of vertices that are part of the i-th symbol group as C_i, and $\mathcal{N}(v)$ refers to the set of vertices that are connected to a vertex v. We also introduce a parameter $\gamma \in \mathbb{N}$ which we refer to as the memory effect. The memory effect quantifies the weight given to decisions taken at previous iterations of the decoding process. We fix γ such that $\gamma \geq c$.

At each iteration, we select the vertices that will remain active based on their *score*. The score of a vertex v is denoted n_v. As described on lines 10–11, the score of a given vertex is incremented once for every group in which it has at least one neighboring active vertex. We then find the maximum score achieved in a given group, denoted $n_{\max}$ (line 12). Note that here Alg. 1 differs from the algorithm in [12], where $n_{\max}$ is always $c + \gamma - 1$ and does not need to be computed. A vertex remains active if it achieves the maximum score in its group. In that case, it is added to the set V_{w} (line 15), which is a temporary variable used to construct the next value of $V_{\widetilde{m}}^{(t)}$.

In most applications, we will be interested in a decoder that generates an estimate $\hat{m} \in \mathcal{A}^c$, as opposed to $\{\mathcal{A} \cup \{\perp\}\}^c$. The optimal estimator will depend on the distribution of the messages in $\mathcal{M}$. If we assume that the messages are independent and identically distributed (i.i.d.) with a uniform distribution, we can define an estimator $h(V_{\widetilde{m}}^{(t)})$ that, for each group, selects an arbitrary vertex from $V_{\widetilde{m}}^{(t)}$ and returns the associated symbol value.

Algorithm 1 has a number of interesting properties [21]. First, if the set of active vertices in a group is not empty, it is non-increasing from one iteration to the next; i.e., $\forall i, t$, if $C_i \cap V_{\widetilde{m}}^{(t)} \neq \emptyset$, then $C_i \cap V_{\widetilde{m}}^{(t+1)} \subseteq C_i \cap V_{\widetilde{m}}^{(t)}$. In addition, as long as the partially erased message $\widetilde{m}$ actually corresponds to a message $m \in \mathcal{M}$ with c_e erased symbols, and $c_e < c$ (i.e., not all symbols were erased), then the set of active vertices in every group remains non-empty for every iteration; i.e., $\forall t > 0$, $\forall i$, $C_i \cap V_{\widetilde{m}}^{(t)} \neq \emptyset$. In fact, once a vertex is not in $V_{\widetilde{m}}^{(t)}$ for some t, it will not re-enter the active set at any future iteration, and consequently it can be shown that Algorithm 1 converges for any input message $\widetilde{m}$. Moreover, any stored message $m \in \mathcal{M}$ is a fixed point of Algorithm 1. (The proof of these statements appears in [21].)

input : A partially erased message $\widetilde{m} \in \{\mathcal{A} \cup \{\perp\}\}^c$
output: A message estimate $\hat{m} \in \mathcal{A}^c$

```
1  begin
2      V_m̃ ← f(m̃)
3      PROGRESS ← true
4      while PROGRESS is true do
5          V_w ← ∅
6          for each group C_i do
7              for each vertex v ∈ C_i do
8                  n_v ← 0 if v ∉ V_m̃, γ otherwise
9                  for each group C_k, k ≠ i do
10                     if N(v) ∩ C_k ∩ V_m̃ ≠ ∅ then
11                         n_v ← n_v + 1
12             define n_max = max_{v∈C_i} n_v
13             for each vertex v ∈ C_i do
14                 if n_v = n_max then
15                     add v to V_w
16         if V_w = V_m̃ then PROGRESS ← false
17         V_m̃ ← V_w
18     return h(V_m̃)
```

Algorithm 1: Retrieval of a previously stored message m from a partially erased message $\widetilde{m}$.

III. Implementation on Unreliable Hardware

In this paper, we would like to discuss the implementation of Algorithm 1 in a physical system, and specifically one that might operate unreliably. To do so, we must describe the physical representation of the algorithm in sufficient details to provide a realistic account of the impact of hardware faults on the functioning of the algorithm. We use the term *fault* to refer to the incorrect operation of a physical component, while the term *deviation* refers to a change in the algorithm's behavior as a result of a hardware fault.

A. Data Structures

We start by describing the data structures that will hold the algorithm's input, output, and intermediate data, in such a way that the mapping to physical components is obvious. The following two data objects are needed: first the set $V_{\widetilde{m}}$ of active vertices, and second a description of the graph edges. As suggested in [21], the edges can be represented using the graph's adjacency matrix W, with dimension $c\ell \times c\ell$. However, the graph is constructed such that there is never an edge between two vertices in the same symbol group, and therefore $c\ell^2$ elements of W are known to be zero. A more compact representation of the edges is achieved by using a set of $\ell \times \ell$ bi-adjacency matrices, each representing the edges between a given pair of groups. We denote the matrix representing the edges from group i to group j by $W_{i,j}$. Unlike W, these matrices are not symmetric, but we have that $W_{i,j} = W_{j,i}^{\mathsf{T}}$. Since the graph edges are unweighted, the

elements of $W_{i,j}$ are binary. Therefore, the number of memory bits required for storing $\{W_{i,j}\}$ is at most

$$Q = \ell^2 \binom{c}{2} = \frac{\ell^2}{2}(c^2 - c). \tag{2}$$

To match the organization of the adjacency data, we represent the vertex states separately for each symbol group. We use a set of vectors $\{a_1, a_2, \ldots, a_c\}$, where each $a_i \in \{0,1\}^\ell$ represents the state (active or not) of the vertices in group i. An element j of a_i is denoted $a_i[j]$ and defined as

$$a_i[j] = \begin{cases} 1 & \text{if } v_{i,j} \in V_{\widetilde{m}} \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where $v_{i,j}$ is the j-th vertex in group i.

B. Unreliable Retrieval Algorithm

To study the implementation of the associative memory, we rewrite Algorithm 1 in terms of the data structures introduced in Section III-A. This is described by Algorithm 2. Algorithm 2 is equivalent to Alg. 1, except for the following minor differences. First, only erased groups are decoded. This has no impact on the analysis of a reliable implementation, but is of some importance when considering the effect of deviations. Second, the algorithm terminates as soon as all the erased symbols have been resolved, or after a maximum of L iterations. Furthermore, we allow inactive vertices to become active again, or in other words, we remove the guarantee that $V_{\widetilde{m}}^{(t+1)} \subseteq V_{\widetilde{m}}^{(t)}$. As a result, we are no longer guaranteed that Algorithm 2 will asymptotically converge to a fixed point; however, the algorithm is still guaranteed to terminate because we impose a maximum number of iterations, L, at line 5.

Algorithm 2 uses the following notation: $\underline{0}$ denotes the zero vector, $w(x)$ denotes the Hamming weight of a binary vector x, and the operator $\otimes$ denotes a matrix product where addition is replaced by logical OR, and multiplication by logical AND. The set E defined on line 2 contains the indices of the erased groups. When accounting for deviations, it is convenient to divide the implementation into two components: first the memory storing the adjacency data $W_{i,j}$, and second the computation units responsible for updating the state vectors a_i, along with the state vector memories. For the first component, we use $\widetilde{W}_{i,j}$ to denote the bi-adjacency matrices affected by deviations. For the second component, the deviations occurring in the logic circuits as well as in the state vector memories are taken into account by adding a vector e_i to each state vector. For a reliable implementation of the algorithm, we define $\widetilde{W}_{i,j} = \widetilde{W}_{i,j}$, and $e_i^{(t)} = \underline{0}$.

Algorithm 2 operates as follows. The input is given in terms of the state vectors, which are constructed in a manner equivalent to Alg. 1, by using (3), where $V_{\widetilde{m}} = f(\widetilde{m})$ is the graphical mapping of the partially erased message $\widetilde{m}$. Starting on line 6, each erased group is then processed as it was in Alg. 1. The scores of each vertex in a group i is represented as a vector s_i of length ℓ. On lines 8–9, the score of each vertex is incremented once for each group in which it has an active neighbor. We denote by b_i the state vectors before deviations are taken into account. On lines 10–13, we mark as active all the vertices that achieve the maximum score. Finally, we simulate the deviations occurring in the state vectors on line 15.

input : $\{a_1^{(0)}, a_2^{(0)}, \ldots, a_c^{(0)}\}$
1 **begin**
2 define $E = \{i \,|\, w(a_i^{(0)}) \neq 1\}$
3 $t \leftarrow 0$
4 VALID $\leftarrow E = \emptyset$
5 **while** $t < L$ *and not* VALID **do**
6 **for** *each* $i \in E$ **do**
7 $s_i \leftarrow \underline{0}$
8 **for** *each* $k \in [1, c], k \neq i$ **do**
9 $s_i \leftarrow s_i + a_k^{(t)} \otimes \widetilde{W}_{k,i}$
10 define $n_{\max} = \max_j s_i[j]$
11 **for** j *from* 1 *to* ℓ **do**
12 **if** $s_i[j] = n_{\max}$ **then** $b_i[j] \leftarrow 1$
13 **else** $b_i[j] \leftarrow 0$
14 $t \leftarrow t + 1$
15 **for** *all* i **do** $a_i^{(t)} \leftarrow b_i + e_i^{(t)} \mod 2$
16 **if** $\forall i, w(a_i^{(t)}) = 1$ **then** VALID $\leftarrow$ true
17 return $\{a_1^{(t)}, a_2^{(t)}, \ldots, a_c^{(t)}\}$

Algorithm 2: Unreliable implementation of the message retrieval.

Upon completion, the algorithm outputs the updated state vectors, and we then use the estimator h to obtain the message estimate $\hat{m} = h(\{a_1^{(t)}, a_2^{(t)}, \ldots, a_c^{(t)}\})$. The estimator h is equivalent to the one used in Alg. 1, and is defined such that for each i, $\hat{m}_i = j$, where j is an arbitrary index that satisfies $a_i^{(t)}[j] = 1$.

C. Circuit Blocks

Algorithm 2 can be easily implemented as a digital circuit. To show this, we briefly describe the main circuit blocks that are required. The reader can also consult [22] for a more detailed example of a possible architecture. Let us start with the computation of $a_k^{(t)} \otimes \widetilde{W}_{k,i}$ on line 9, which happens $c-1$ times for every erased group. The operation can be viewed as first applying an AND mask to entire rows of $\widetilde{W}_{k,i}$, followed by an OR operation on the columns. However, no AND operations need to be performed in practice. If $\widetilde{W}_{k,i}$ is stored row-by-row in memory, the indices of the non-zero elements in $a_k^{(t)}$ are the memory addresses that need to be accessed. Typically, the number of active vertices $w(a_k^{(t)})$ in a group will be small (for example, it is 1 for non-erased groups), and therefore, only a few rows of $\widetilde{W}_{k,i}$ need to be retrieved. To complete the matrix product, we compute the element-by-element OR of the rows that were retrieved.

We then use the result of the computation above to increment s_i, which can be achieved using ℓ counters. Finally, the maximum score in s_i can be computed using a tree of compare-and-select circuits. This can be simplified if we assume that the maximum will be equal to one of the highest scores, and abort the retrieval otherwise. In that case, the comparators can be replaced with equality checks, implemented using AND gates.

D. Deviation Model

We can distinguish two types of faults affecting circuits. *Permanent* faults are irreversible and can occur during fabrication or later on because of wear caused by the circuit's operation. *Transient* faults are caused by temporary fluctuations in some

operational parameter, or by the interference of external radiation. To model the impact of these faults in a circuit of interest, we consider that the output of a reliable version of the circuit is transmitted through a *deviation channel*. The output of the deviation channel then corresponds to the output of the unreliable version of the circuit. We assume that deviations occur independently on each output bit of storage or computational circuits. Below we show that this is reasonable for the algorithm under consideration.

1) Deviations in the Adjacency Data: The adjacency data is represented by the bi-adjacency matrices $\{W_{i,j}\}$, which must be stored in a memory. Integrated circuit memories such as SRAMs can suffer from both permanent and transient faults [23], [24]. One way to deal with permanent faults is by identifying bad parts of the memory circuit and re-mapping these parts on spare circuits. The testing and re-mapping can be performed independently and in a manner that is transparent to the algorithm, and therefore need not concern the algorithm designer. However, there is a cost for including redundant memory elements and for the logic that performs the re-mapping. Another way to deal with a faulty memory, and which is suitable both for permanent and transient faults, is to encode the data to be stored with an error correction code, which can then be decoded with a faulty decoder [25], [26]. However, this adds complexity and reduces flexibility by imposing a constraint on the number of bits that can be read or written simultaneously, which must be a multiple of the code size. Therefore, providing fault tolerance directly at the algorithm level is likely to be more efficient.

We first consider that the memory storing $\{W_{i,j}\}$ is affected by permanent faults. The associative memory is used in two phases. First, we generate and store the message set $\mathcal{M}$, which fixes $\{W_{i,j}\}$. We model the deviations by saying that $\{W_{i,j}\}$ is transmitted through Q parallel binary deviation channels with output $\{\widetilde{W}_{i,j}\}$, where Q is the number of bits used to represent $\{W_{i,j}\}$, defined in (2). Since we assume that deviations occur independently on each bit, the parallel channels are independent. Once $\mathcal{M}$ has been stored, and with $\{\widetilde{W}_{i,j}\}$ fixed, we perform a number of retrieval operations.

A common deviation model for permanent faults, which was found to be accurate for more than 50% of SRAM defects [23], is the "stuck-at" model, where a defective 1-bit circuit has either a fixed output of 0 (stuck-at-0) or 1 (stuck-at-1). Each binary deviation channel is therefore a channel with input X and output Y such that

$$Y = \begin{cases} 0 & \text{with probability } \psi_0, \\ 1 & \text{with probability } \psi_1, \\ X & \text{with probability } 1 - \psi_0 - \psi_1. \end{cases} \tag{4}$$

Since faults are permanent, consecutive uses of a given binary channel in a given fabricated device are not independent. However, if we assume that messages are independent and uniformly distributed, the knowledge of deviations occurring in the decoding of one message does not affect the probability of successfully retrieving another message. Therefore it is equivalent to consider that any message to be retrieved is the first message to be retrieved, and that each of the Q binary channels is only used once. If the stuck-at-0 and stuck-at-1 events are equiprobable and the channel is only used once, the transmission can be equivalently modeled using a binary symmetric channel (BSC) defined as

$$Y = \begin{cases} (1 - X) & \text{with probability } \psi, \\ X & \text{with probability } 1 - \psi, \end{cases} \tag{5}$$

where $\psi = \psi_0 = \psi_1$.

If the Q bits of $\{W_{i,j}\}$ are transmitted using (5), we obtain

$$\widetilde{W}_{i,j} = \begin{cases} W_{i,j} + D_{i,j} \bmod 2 & \text{if } i < j \\ \widetilde{W}_{j,i}^{\mathsf{T}} & \text{if } i > j \end{cases}, \tag{6}$$

where $D_{i,j}$ is an indicator matrix of deviation events, defined as a random matrix of size $\ell \times \ell$, where each element $d_{x,y}$ is an independent Bernoulli random variable with $\mathbb{P}(d_{x,y} = 1) = \psi$. For each (i, j) with $i < j$, we only store one of $W_{i,j}$ or $W_{j,i}$. Therefore, when considering faults, the identity $\widetilde{W}_{i,j} = \widetilde{W}_{j,i}^{\mathsf{T}}$ remains valid.

The impact of transient faults occurring in the memory is similar to permanent faults if we assume that a transient fault on a bit cell permanently changes the value of that cell. The only difference is that transient faults can occur at any point in time (but note that this is also the case of permanent faults that are due to aging). To simplify, we assume that the faults occur only between invocations of the retrieval algorithm. In that case, the new deviations can be modeled by re-transmitting $\{\widetilde{W}_{i,j}\}$ through a BSC before every message retrieval operation, with every channel use independent of the previous ones. However, it is easily shown that successive transmissions through a BSC can be reduced to a single transmission through a BSC (with a different cross-over probability). Therefore, the deviation model for transient faults is the same as for permanent faults, although ψ increases over time when faults are transient.

Under this model, the performance of the associative memory is a function of two random variables, namely $\{\widetilde{W}_{i,j}\}$ and the partially erased message $\widetilde{m}$. We expect several messages to be retrieved from the memory, and we are interested in the average performance over all messages retrieved. It follows that the performance should be defined as an expectation over $\widetilde{m}$. On the other hand, $\{\widetilde{W}_{i,j}\}$ is only sampled once, and we are therefore interested in the performance for a particular realization of $\{\widetilde{W}_{i,j}\}$. Nonetheless, the approach taken for the analysis in Section IV is to consider the expected performance over $\{\widetilde{W}_{i,j}\}$. This is motivated by simulation results that show that the performance for various realizations of $\{\widetilde{W}_{i,j}\}$ is very concentrated around the mean.

2) Deviations in the State Vectors: We also consider the possibility of observing deviations on the state vectors $\{a_1^{(t)}, a_2^{(t)}, \ldots, a_c^{(t)}\}$. In addition to modeling faults occurring in the memory storing the vectors, we argue that this can also serve as an approximate model for faults occurring in the logic. To maintain a simple model, we would like to assume that individual bits are being affected independently by deviations. Fortunately, most of the operations in Algorithm 2 do generate each bit of the new state vectors independently. The only exception is the max operation on line 10. If a deviation occurs on $n_{\max}$ for some group, it could in turn cause multiple correlated deviations in the state vector of that group. Therefore, our model is only valid if the max operations are implemented reliably.

We also limit ourselves to considering transient deviations, that are sampled at every iteration t. In addition to modeling transient faults, transient deviations can also represent the effect of permanent variations in the logic, which may or may not be *sensitized*, depending on the signal transitions taking place. For example, if as a result of manufacturing variations, a logic gate reacts too slowly to changes in its inputs [27], this defect will only cause a deviation when the inputs are changing. If they are stable, no deviation occurs. However, permanent faults in the state vector memory have to be excluded.

Recall that the deviations on the state vectors are represented by the vectors $e_i^{(t)}$, $i = \{1, 2, \ldots, c\}$. Each vector $e_i^{(t)}$ is an indicator of deviation events on individual bits of state vector i, at iteration t, and each element $e_i^{(t)}[j]$ is obtained by sampling an independent Bernoulli random variable, such that $\mathbb{P}(e_i^{(t)}[j] = 1) = \phi$.

IV. Retrieval Performance

When the network is used as an associative memory, we are interested in retrieving a previously stored message given a partially erased version $\widetilde{m}$, where c_e of the c symbols have been erased. To assess the retrieval performance, we will consider the probability $P_s^{(t)}$ that an erased symbol is retrieved correctly after t decoding iterations, and also the probability $P_m^{(t)}$ that the entire message is retrieved correctly after t decoding iterations.

Throughout the analysis, we assume that the stored messages are i.i.d. uniform. The uniform distribution is of particular interest for the analysis of an implementation with permanent storage faults because it yields the worst retrieval performance. This is because a non-uniform message distribution implies that the probability that a particular graph edge is present is also non-uniform. Therefore, given the knowledge of which storage elements are faulty (which can be obtained easily using a standard memory built-in self-test), it could be possible to change the mapping of memory elements to graph edges such that stuck-at-1 elements are associated with edges that are more likely to be present, and stuck-at-0 elements with edges that are more likely to be absent, thereby improving the retrieval performance.

A. Deviation-Free Case

Denote by $M = |\mathcal{M}|$ the number of messages to store. Note that during the storing process, each message to store adds—or does not modify if it already exists—one edge between every pair of symbol groups.

Let us fix a pair of vertices from distinct groups. The probability that a message obtained using i.i.d. uniform random variables adds the edge between them while being stored is trivially $1/\ell^2$.

Thus, the probability that this edge is added to the graph after storing M i.i.d. uniform messages is[2]:

$$d = 1 - \left(1 - \frac{1}{\ell^2}\right)^M. \tag{7}$$

The existence of edges is not independent since a message adds multiple correlated edges while being stored. However, for $M \ll \ell^3$ and $M^2 \ll \ell^c$, we can assume that they are independent, as motivated by simulation results and by the following theorem.

1) Theorem 1: We assume that $\mathcal{M}$ contains uniform i.i.d. messages such that $M = o(\ell^3)$ and $M^2 = o(\ell^c)$. Select any two distinct pairs of vertices in the graph. Denoting $\mathbb{P}(E_2)$ the probability that an edge exists for both pairs, we have that

$$\mathbb{P}(E_2) \underset{\ell \to \infty}{\sim} d^2 \tag{8}$$

and therefore the existence of two distinct edges can be asymptotically regarded as independent.

Proof: See Appendix. ∎

Unfortunately, Theorem 1 does not extend in a straightforward way to cover the mutual independence of the existence of more than two edges, but it nonetheless serves to provide additional support for the assumption.

Let us now fix a message m in the set of stored messages, and erase $c_e > 0$ symbols uniformly at random to obtain a partial message $\widetilde{m}$. Note that when no symbols are erased (i.e., $c_e = 0$), the input $\widetilde{m} \in \mathcal{M}$ is a fixed point of the algorithm iterations, and so $P_s = P_m = 1$. The probability of retrieving m correctly using Algorithm 1 can be bounded from below by considering the case where the algorithm is used for a single iteration. This is clearly a lower bound since $V_{\widetilde{m}}^{(t+1)} \subseteq V_{\widetilde{m}}^{(t)}$.

Let us first consider the probability $P_s^{(1)}$ that a given erased symbol is successfully retrieved after one iteration. Recall that we generate an estimate of the message using $\hat{m} = h(\{a_1^{(t)}, a_2^{(t)}, \ldots, a_c^{(t)}\})$, where h is a function that, for each symbol group i, selects an arbitrary vertex from the set of active vertices, and returns the associated symbol value. Therefore, we have that $P_s^{(t)} = \mathbb{E}[1/A]$, where A represents the number of active vertices in the group. By definition, all active vertices achieve the maximum score $n_{\max}$, which in the first iteration is $n_{\max} = c_k = c - c_e$. Also, since correct vertices—that is those corresponding to $f(m)$—will remain in $V_{\widetilde{m}}$, we can write $A = 1 + A_s$, where $A_s \in \mathbb{N}$ represents the number of spurious vertices that remain activated. Since non-erased groups contain a single active vertex, and since edges are assumed to be independent, a spurious vertex remains activated with probability d^{c_k}. Therefore, $P_s^{(1)}$ is given by

$$\begin{aligned} P_s^{(1)} &= \sum_{k=0}^{\ell-1} \frac{1}{k+1} \mathbb{P}(A_s = k) \\ &= \sum_{k=0}^{\ell-1} \frac{1}{k+1} \binom{\ell-1}{k} d^{kc_k} (1 - d^{c_k})^{\ell-1-k}. \end{aligned} \tag{9}$$

Because edges are assumed independent and subsets of edges that target each erased symbol in the first iteration are disjoint, the probability that the complete message is retrieved successfully after the first iteration is simply

$$P_m^{(1)} = \left(P_s^{(1)}\right)^{c_e}. \tag{10}$$

The expression for $P_m^{(1)}$ was confirmed to match simulation results. The curve for $\psi = 0$ and 1 iteration in Fig. 2 provides an example for the case of $c = 8$, $\ell = 256$, and $c_e = c/2$.

B. With Deviations in the Adjacency Data

We now consider the effect of deviations on the matrices $W_{i,j}$ on the retrieval performance, as described by (6). Because we

[2]See details in proof of Theorem 1.

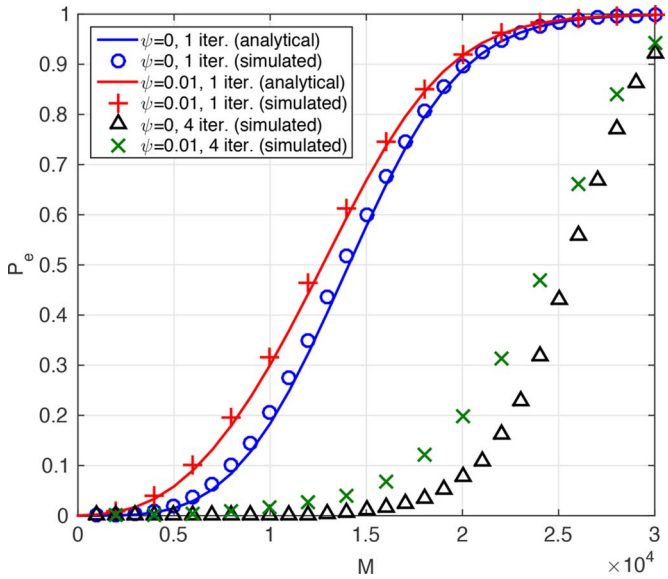

Fig. 2. Message error rate as a function of the number M of stored messages, for a network with $c = 8$, $\ell = 256, c_e = \frac{c}{2}$, when the bi-adjacency matrices are affected by deviation with prob. $\psi \in \{0, \frac{1}{100}\}$. Solid curves represent analytical results.

have $\widetilde{W}_{i,j} = \widetilde{W}_{j,i}^{\mathsf{T}}$, we can equivalently model the deviations in terms of the graph model. For each pair of vertices, if an edge exists, we remove it with probability ψ, and if no edge exists, we add one with probability ψ.

As mentioned in Section III.D1, we simplify the analysis by considering the expected performance over all realizations of $\{\widetilde{W}_{i,j}\}$. In other words, we study a typical faulty memory. Let $P_s^{(t)}(D)$ and $P_m^{(t)}(D)$ represent respectively the symbol and message retrieval probability for a particular realization D of the storage deviations. To motivate our approach, we performed Monte-Carlo simulations to measure the mean μ and standard deviation σ of $P_s^{(t)}(D)$ and $P_m^{(t)}(D)$. For every data point shown in Fig. 2, we sampled 200 realizations of the storage deviations. For each of these realizations, the error rate was measured by averaging over 100,000 message retrieval attempts. The results show that the performance is very concentrated. In fact, for all the data points, $\sigma < 0.002$, such that the interval $[\mu - 3\sigma, \mu + 3\sigma]$ is too small to be visible at the scale of the Figure. Therefore, we allow ourselves to write $P_s^{(t)}$ instead of $\mathbb{E}_D[P_s^{(t)}(D)]$ and $P_m^{(t)}$ instead of $\mathbb{E}_D[P_m^{(t)}(D)]$.

As in Section IV-A, we consider the probability $P_s^{(1)}$ that a given erased symbol is retrieved after one iteration. Because of the deviations on the edges, the correct vertex is not guaranteed to achieve the maximum score. Let n_{v_0} be the score achieved by the correct vertex v_0, and let F denote the event that at least one incorrect vertex in the group achieves a score higher than n_{v_0}. Note that if F occurs, the probability of identifying the correct vertex is 0. Whereas in the reliable case we had $P_s^{(1)} = \mathbb{E}[1/A]$, we now have

$$P_s^{(1)} = \sum_{n_0=0}^{c_k} \mathbb{E}\left[\frac{1}{A}\middle| \neg F, n_{v_0} = n_0\right] \cdot \mathbb{P}\left(\neg F \mid n_{v_0} = n_0\right) \cdot \mathbb{P}\left(n_{v_0} = n_0\right), \quad (11)$$

where $\neg F$ denotes the negation of event F.

Let us now expand each of the three expressions in (11). First, the probability that the correct vertex v_0 achieves a score of n_0 for $\psi > 0$ is given by

$$\mathbb{P}(n_{v_0} = n_0) = \binom{c_k}{n_0} (1-\psi)^{n_0} \psi^{c_k - n_0}. \quad (12)$$

On the other hand, using the edge independence assumption, the probability that any other vertex is connected to the active vertex of a non-erased group is $P_+ = \psi(1-d) + (1-\psi)d$, and the probability mass function of the score n_v of incorrect vertices is therefore

$$\mathbb{P}(n_v = x) = \binom{c_k}{x} {P_+}^x (1-P_+)^{c_k - x}, 0 \le x \le c_k. \quad (13)$$

The scores of vertices are independent because each vertex is associated with a distinct set of messages. The probability that the correct vertex is in the active set when $n_{v_0} = n_0$ is then

$$\mathbb{P}\left(\neg F \mid n_{v_0} = n_0\right) = \left(\sum_{x=0}^{n_0} \mathbb{P}(n_v = x)\right)^{\ell - 1}. \quad (14)$$

Finally, the probability of selecting the correct vertex from the active set when the retrieval has not failed and when $n_{v_0} = n_0$ is given by

$$\mathbb{E}\left[\frac{1}{A}\middle| \neg F, n_{v_0} = n_0\right] = \sum_{k=0}^{\ell-1} \frac{1}{k+1} \binom{\ell-1}{k} \pi^k (1-\pi)^{\ell-1-k}, \quad (15)$$

where π is the probability that a spurious vertex remains in the active set when $\neg F$ and $n_{v_0} = n_0$ occur, given by

$$\pi = \mathbb{P}\left(n_v = n_{v_0} \mid \neg F, n_{v_0} = n_0\right) = \frac{\mathbb{P}\left(n_v = n_0\right)}{\mathbb{P}\left(n_v \le n_0\right)}. \quad (16)$$

Note that (16) can be evaluated easily based on (13).

As previously, the message retrieval probability $P_m^{(1)}$ can be obtained using (10).

Fig. 2 shows the message retrieval performance when messages are composed of 8 symbols of 8 bits each (i.e., $\ell = 256$), and half the symbols are erased. Simulation results for a single iteration confirm the analytical expressions. Even when the deviation probability is as high as 1%, the retrieval performance remains reasonably close to that of a reliable implementation. At $\psi = 10^{-3}$, the difference in performance with a reliable memory becomes negligible (for this reason the curve is not shown in the figure).

C. With Deviations in the State Vectors

In order for the retrieval algorithm to output the correct message with certainty, it is necessary that only one vertex be active in each erased group, and that these active vertices correspond to the correct symbols. Therefore, it could seem that the algorithm is very sensitive to deviations in the state vectors, since any deviation in the state vector of any erased group will prevent from retrieving the message with certainty. However, starting from a state vector that maps to the correct symbol, most deviations on the vector are detectable. Recall that vectors b_i represent the state vectors before taking deviations into account. Assuming $w(b_i) = 1$, a deviation is undetectable if and only if

$w(b_i + e_i \bmod 2) = 1$, which implies that a deviation occurs on the active bit, while exactly one other deviation occurs on one of the inactive bits. We call this event I, which happens with probability $\mathbb{P}(I) = (\ell - 1)\phi^2(1-\phi)^{\ell-2}$. Usually, this will be much smaller than the symbol error probability $(1 - P_s)$. For example, $\mathbb{P}(I) < 0.0022$ for $\ell = 256$.

Therefore, some simple modifications to the algorithm can increase its fault tolerance when deviations occur in the state vectors. First, we can "freeze" state vectors that achieve a weight of one. This amounts to repeating line 2 of Alg. 2 at every iteration of the loop. Such a modification is reasonable in the context of an unreliable implementation because it simply corresponds to outputting some symbols before all the symbols have been retrieved. Second, we have the option of increasing the number of iterations.

We can obtain an approximation for the message retrieval probability by considering that we use a reliable implementation until the termination condition becomes true at iteration $t = r$, and that state vector deviations can occur only for iterations $t \geq r$. We also assume that $w(b_i) = 1, \forall i, \forall t \geq r$.

We first consider the probability $P_{s,\text{dev}}^{(\tau)}$ that a given symbol is retrieved within τ iterations after the reliable implementation terminates ($\tau \geq 0$), such that the total number of decoding iterations is $t + \tau$. A symbol is retrieved successfully in $t + \tau$ iterations if its state vector converges to the correct symbol within τ additional iterations (event "GC"), or if the state vector does not converge within τ iterations (event "NC"), but the estimator makes the right guess (event "EST"):

$$P_{s,\text{dev}}^{(\tau)} = \mathbb{P}(\text{GC}) + \mathbb{P}(\text{EST}\,|\,\text{NC})\mathbb{P}(\text{NC}). \tag{17}$$

At each iteration there are three possible outcomes: incorrect convergence (event I), correct convergence (event S), or no convergence (event N). $\mathbb{P}(I)$ was stated above, $\mathbb{P}(S) = (1-\phi)^{\ell}$, and $\mathbb{P}(N) = 1 - \mathbb{P}(S) - \mathbb{P}(I)$. In the case the decoder does not converge, we must consider the number A of active vertices in the group. Since we assume that $w(b_i) = 1$, the fact that $A \neq 1$ can only be due to deviations in the current iteration. The event probabilities in (17) are given by

$$\begin{aligned}
\mathbb{P}(\text{GC}) &= \frac{\mathbb{P}(S)(1 - \mathbb{P}(N)^{\tau+1})}{1 - \mathbb{P}(N)} \\
\mathbb{P}(\text{NC}) &= \mathbb{P}(N)^{\tau+1} \\
\mathbb{P}(\text{EST}\,|\,\text{NC}) &= \mathbb{E}\left[\frac{1}{A} \mid A > 1\right],
\end{aligned}$$

where

$$\mathbb{E}\left[\frac{1}{A} \mid A > 1\right] = \sum_{x=2}^{\ell} \frac{1}{x} \binom{\ell - 2}{x - 2} \phi^{x-2}(1-\phi)^{\ell-x+2}. \tag{18}$$

Since deviations are independent, the probability of retrieving a message in τ additional iterations is $P_{m,\text{dev}}^{(\tau)} = \left(P_{s,\text{dev}}^{(\tau)}\right)^{c_e}$. Therefore, denoting by T the number of iterations before a reliable implementation terminates, the probability $\widetilde{P}_m^{(t)}$ of successfully retrieving a message in the presence of state vector deviations is approximated by

$$\widetilde{P}_m^{(t)} \approx \sum_{r=1}^{t} \mathbb{P}(T = r) P_{m,\text{dev}}^{(t-r)}. \tag{19}$$

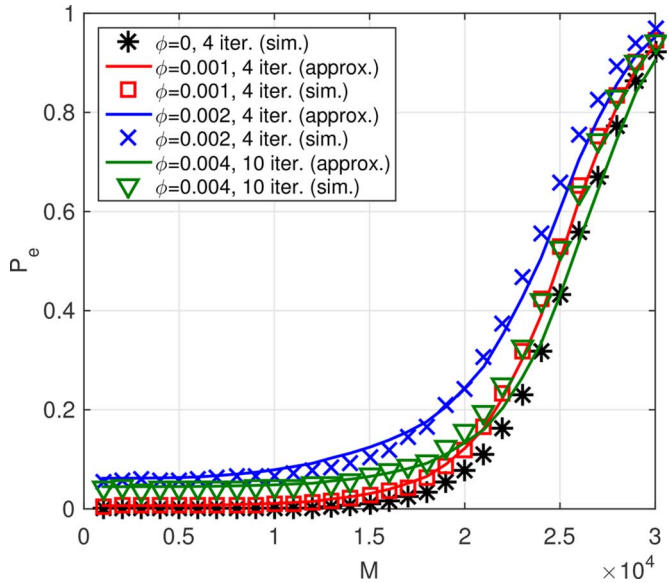

Fig. 3. Message error rate as a function of the number M of stored messages, for a network with $c = 8, \ell = 256, c_e = \frac{c}{2}$, when the state vectors are affected by deviations. The error rates are averaged over 10^4 retrieval attempts. The solid curves represent the semi-analytical evaluation of (19) based on simulations of a reliable implementation.

For $r = 1$, $\mathbb{P}(T = 1) = P_m^{(1)}$ is given by (9) and (10). We rely on simulations to evaluate $\mathbb{P}(T = r)$ for $r > 1$.

Fig. 3 shows an example of the message retrieval performance for $c = 8, \ell = 256$, and half the symbols erased. We first note that the analytical approximation is fairly accurate. When $\phi = 10^{-3}$, the performance loss is minimal, even when using the same number of iterations as the reliable implementation. When the iteration limit is fixed at 4, the performance degrades rapidly for ϕ values above 10^{-3}. This is expected due to the simplicity of the fault tolerance mechanism. However, it is important to note that this simplicity also means that no complexity is added to the implementation, and that therefore the fault tolerance shown here comes for free. Finally, as demonstrated by the curve for $\phi = 4 \cdot 10^{-3}$ and 10 iterations, the memory can tolerate slightly larger deviation rates by increasing the number of iterations.

V. Storage Efficiency

A Figure of merit that is often quoted for an associative memory is its *capacity*, that is the amount of information that can be stored in the network as defined by the entropy, and given in terms of the number of vertices in the graph. The neural network model considered in this paper can be shown to allow the storage of $\Theta(\ell^2)$ uniformly distributed messages when $c = \Theta(\log(\ell))$. However, in practice, it is more important to compare the information that the memory can store and retrieve with the amount of storage material required to represent it, which we refer to as *storage efficiency*. Capacity alone is not a proper measure of efficiency, since messages are stored by the graph edges, and not by the vertices. From (2), the storage required for the proposed memory is $\Theta(\ell^2 c^2)$ bits. Since the entropy of ℓ^k uniformly generated messages is[3]

[3]Assuming that no repetition occurs in $\mathcal{M}$, which is true with high probability (see also details in proof of Theorem 1). We use this assumption throughout the section.

$c\ell^k \log_2(\ell) = \Omega(\ell^k)$ bits, it is not possible to store $O(\ell^k)$ messages with $k > 2$ without increasing the storage requirement. Doing so would imply that the information is represented with fewer bits than the entropy, which by Shannon's source coding theorem cannot be done losslessly.

The capacity remains a useful property of the memory, since a greater capacity allows storing more messages without increasing the size of the network. We note that the model used in this paper can be used to store any number of messages by adding a pre-processing step before storing the messages. Suppose that one wants to store $O(\lambda^k)$ messages of χ symbols in an alphabet of size λ. We can pre-process messages by grouping tuples of $\alpha = \lfloor k/2 \rfloor$ symbols, to obtain messages made of $\lceil \chi/\alpha \rceil$ symbols in an alphabet of size λ^α. The network is then able to store a quadratic number of such messages, that is $\Theta(\lambda^{2\alpha}) = O(\lambda^k)$. In order to allow the retrieval of partially erased messages in the initial space, the retrieval process described in this paper must be extended to allow the initial set of active vertices to contain more than one active vertex per symbol group.

A. Efficiency Definitions

The information provided by the memory when retrieving a message $m \in \mathcal{M}$ can be expressed in information-theoretic terms as the mutual information between the memory output $\hat{m}$ and the stored message m, which we denote $I(m, \hat{m})$. A message is retrieved from the memory by providing a partially erased message $\widetilde{m}$, which provides information $I(m, \widetilde{m})$ on the stored message. As is commonly done in the literature on Willshaw networks (see e.g., [28] and references therein), we base our efficiency definition on the new information provided by the memory, given by $I(m, \hat{m}) - I(m, \widetilde{m})$.

If we assume that symbol errors in the retrieval occur independently and that an incorrectly retrieved symbol can take any of the $\ell - 1$ incorrect values with equal probability, then the retrieval of an erased symbol m_i can be modeled as transmitting m_i through a non-binary symmetric channel with an alphabet of size ℓ. For a symbol error probability p, the capacity $C(p)$ of this channel is given by

$$C(p) = \log_2(\ell) + (1-p)\log_2(1-p) + p\log_2(p) - p\log_2(\ell - 1) \tag{20}$$

in bits per channel use, and this capacity is achieved with uniformly distributed inputs [29].

If we consider the retrieval of all stored messages, the new information provided by the memory can be expressed as $I(\mathcal{M}, \widehat{\mathcal{M}}) - I(\mathcal{M}, \widetilde{\mathcal{M}})$, where $I(\mathcal{M}, \widehat{\mathcal{M}})$ is the mutual information between the set of stored messages and the set of retrieved messages, and similarly $I(\mathcal{M}, \widetilde{\mathcal{M}})$ is the mutual information between the set of stored messages and the set of partially-erased messages. The information in the partially-erased messages is simply

$$I(\mathcal{M}, \widetilde{\mathcal{M}}) = Mc_k \log_2(\ell), \tag{21}$$

and since the location of the erased symbols is known at retrieval time, we have

$$I(\mathcal{M}, \widehat{\mathcal{M}}) = M \cdot (c_k \log_2(\ell) + c_e C(1 - P_s^{(t)})), \tag{22}$$

where as previously $P_s^{(t)}$ denotes the probability of retrieving an erased symbol in t iterations.

We then define the storage efficiency η for $c_e > 0$ as

$$\eta = \frac{c \cdot (I(\mathcal{M}, \widehat{\mathcal{M}}) - I(\mathcal{M}, \widetilde{\mathcal{M}}))}{c_e Q} = \frac{Mc\,C(1 - P_s^{(t)})}{Q}, \tag{23}$$

where Q is given by (2). An upper bound can be easily derived for η. Since messages in $\mathcal{M}$ are uniformly distributed, the entropy $H(\mathcal{M})$ of the set is $H(\mathcal{M}) = Mc\log_2(\ell)$. Clearly, it is only possible to retrieve messages correctly with a high probability if $H(\mathcal{M}) \leq Q$. Since we also have $C(p) \leq \log_2(\ell)$, it follows that $\eta \leq 1$. The multiplication by c/c_e in (23) serves to make the upper bound independent of c_e/c to allow comparing memories with different fractions of erased symbols. Note that the typical efficiency definition used for Willshaw networks does not include this normalization.

Another efficiency definition that has been proposed recently [30]normalizes the information provided by the memory to the entropy of the network representation $H(Q)$. However, such a definition implies the use of a compression mechanism when storing the graph representation, which increases retrieval complexity. Furthermore, when the storage device is unreliable, using compression without error correction coding would likely decrease the robustness of the implementation. Therefore, for our purposes the definition in (23) is more appropriate.

In some applications, it might be required to discard retrieved messages that are not entirely correct. Assuming that there exists an application-specific way of identifying messages that have been incorrectly retrieved, such a situation corresponds to transmitting the messages through an erasure channel with erasure probability $1 - P_m^{(t)}$. In this case, we have $I(\mathcal{M}, \widehat{\mathcal{M}}) = M \log_2(\ell)(c_k + c_e P_m^{(t)})$, and we call the resulting efficiency the *message-wise* efficiency, defined as

$$\eta_m = \frac{Mc\log_2(\ell)P_m^{(t)}}{Q}, \tag{24}$$

where the information was scaled by c/c_e as in (23), so that $\eta_m \leq 1$.

We consider the maximum efficiency that can be achieved by the memory. For a fixed c and ℓ, the efficiency varies in terms of M. We denote the maximum efficiency in M by $\eta^* = \max_M(\eta)$, and similarly the maximum message-wise efficiency by $\eta_m^* = \max_M(\eta_m)$.

B. Efficiency With Adjacency Deviations

We first consider the storage efficiency when deviations occur in the adjacency data with probability ψ. The maximum efficiency depends on ψ and we write it as $\eta^*(\psi)$ and $\eta_m^*(\psi)$. For a single iteration of the algorithm, we can evaluate $\eta^*(\psi)$ by combining (11) and (23) and optimizing numerically. Doing so reveals that for a fixed ℓ, the efficiency varies in terms of c. Therefore we can look for the value of c that maximizes the efficiency when the proportion of erased symbols remains $c_e = c/2$:

$$\eta^{**}(\psi) = \max_c \eta^*(\psi)|_{c_e = c/2}. \tag{25}$$

Note that because of the way c_e is defined, we restrict c to even values. The results for three values of ℓ are shown in Fig. 4. We see that efficiency increases with ℓ, and that the value c^* of c that maximizes the expression tends to increase slightly with ψ. We can compare the efficiency obtained by the c-partite AM at $\psi = 0$ with that of auto-associative Willshaw networks,

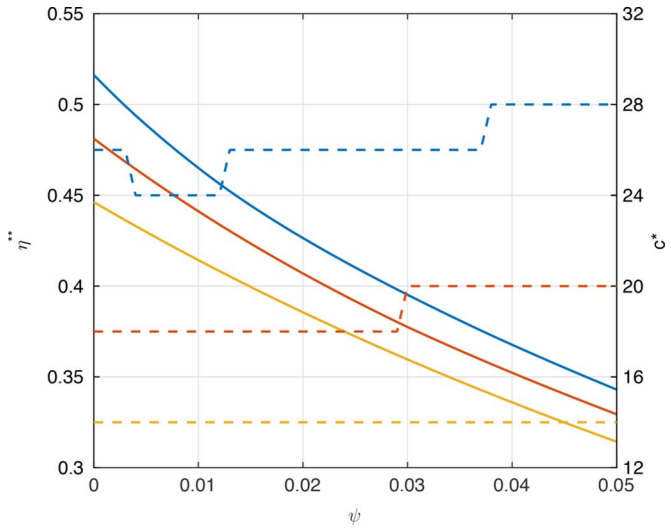

Fig. 4. The solid curves show the storage efficiency for a single iteration of the retrieval algorithm, maximized in terms of M and of c, with $c_e = c/2$, and c even. The dashed curves show the value of c that maximizes η^*. From top to bottom, the curves correspond to $\ell = 2048$, 256, and 64.

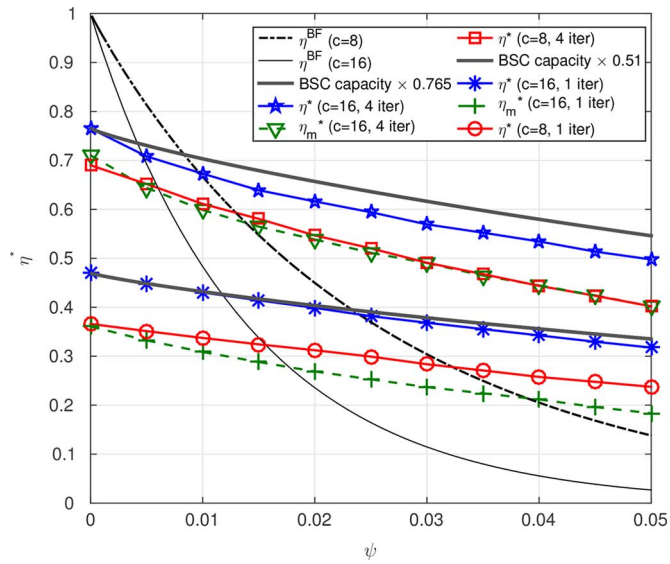

Fig. 5. η^* (solid curves) or η_m^* (dashed curves) as a function of the deviation parameter ψ affecting the bi-adjacency matrices. For all curves, $c_e = c/2$ and $\ell = 256$. The argument M^* of η^* and η_m^* is found using a resolution of 100 messages, and retrieval error rates are averaged over 10^4 trials. Also shown is the efficiency of a brute-force memory and the capacity of the binary symmetric channel scaled by some constants.

taking into account the scaling factor that we introduce in the definition of η. For all the examples shown in Fig. 4, the c-partite AM with a single retrieval iteration has a larger efficiency than the asymptotic efficiency $\ln(2)/4 \cdot c/c_e \approx 0.347$ of Willshaw networks with single-iteration retrieval [9], and also a larger efficiency than the best result reported in [28] using multiple-iteration retrieval $(0.19 \cdot c/c_e = 0.38)$.

The efficiency can be greatly improved by using more than one decoding iteration. In that case, we evaluate $\eta^*(\psi)$ based on simulation results. Results for $c \in \{8, 16\}$, $\ell = 256$, and half the symbols erased are shown in Fig. 5. Note that the efficiency at $c = 16$ for a single retrieval iteration is close to the optimum. Comparing with Willshaw networks, we see that the efficiency of a reliable implementation of a c-partite AM with $c = 16$, $\ell = 256$ and 4 iterations is larger than the so-called learning bound for the efficiency of Willshaw networks, given by $\ln(2)/2 \cdot c/c_e \approx 0.693$ [17], which is independent of the retrieval algorithm. For this example, increasing the number of iterations beyond 4 has little effect.

For the case of $c = 16$ and 4 iterations, the argument M^* of η^* at $\psi = 0$ is $M^* = 49162$, and the corresponding symbol retrieval probability is $P_s^{(4)*} = 0.976$. Both M^* and $P_s^{(4)*}$ decrease as ψ increases, a trend observed in all the examples.

As can be expected, the efficiency is lower if we require complete messages. For clarity, these $\eta_m^*(\psi)$ curves are shown for $c = 16$ only. Under unreliable storage, we see that large deviation rates can be tolerated at the expense of a small reduction in the maximum efficiency. For example, when ψ goes from 0 to 0.04, the memory with $c = 16$, $\ell = 256$ looses 30% of its efficiency (or 36% when measured message-wise). On the other hand, for a 65 nm CMOS process, tolerating an error rate of 4% in an SRAM can reduce energy consumption by close to one order of magnitude [31]. As process variations increase in smaller technologies, the ratio of energy savings to error rate becomes even more favorable.

Associative memories can also be built using a standard memory array that represents $\mathcal{M}$ directly, equipped with a brute-force search to perform retrieval. This alternative implementation approach, which we refer to as the brute-force (BF) memory, provides a point of comparison for the fault tolerance of the proposed memory. Contrary to the graph-based memory, under a reliable implementation the retrieval performance of the BF memory is only limited by potential ambiguities in $\mathcal{M}$, that is, by the possibility that there are multiple messages that have the same non-erased symbols as the probe $\widetilde{m}$. This event is very unlikely for the values of c_k, ℓ, and M considered here. Therefore we assume that a reliable implementation can retrieve all stored messages perfectly. If individual bits from the memory are flipped with probability ψ, an erased symbol can only be retrieved if no deviations occur in storing the known part of the message. If that is the case, a retrieved symbol provides $C_{\text{dev}}(\psi)\log_2(\ell)$ bits of information about the stored symbol, where $C_{\text{dev}}(\psi) = 1 + \psi\log_2(\ell) + (1-\psi)\log_2(1-\psi)$ is the capacity of the binary deviation channel. Therefore, and since the BF memory stores a direct representation of the messages, its efficiency is

$$\eta^{\text{BF}} = (1-\psi)^{c_k \log_2(\ell)} C_{\text{dev}}(\psi), \tag{26}$$

where as in (23) and (24) the new information provided by the memory is scaled by c/c_e.

We see in Fig. 5 that the proposed associative memory maintains a good efficiency as ψ increases, whereas the efficiency of the brute-force memory quickly goes to 0. Of course, this lackluster fault tolerance of the brute-force memory is not its only drawback, the main one being that a brute-force retrieval involves many more memory accesses than a graph-based retrieval method such as Algorithm 2. The brute-force memory could use an error correction code to increase its fault tolerance, but in addition to increasing the retrieval complexity, this can pose the problem of also providing fault tolerance in the error correction decoder.

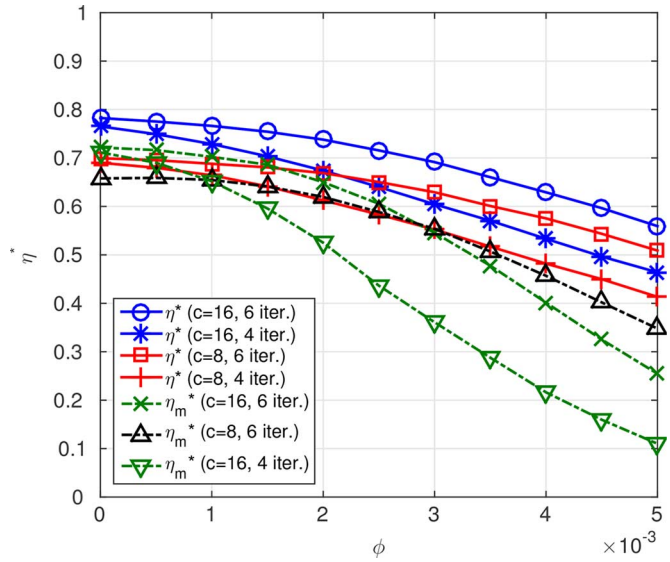

Fig. 6. η^* (solid curves) or η_m^* (dashed curves) as a function of the deviation parameter ϕ affecting the state vectors. For all curves, $c_e = c/2$ and $\ell = 256$. The argument M^* of η^* and η_m^* is found using a resolution of 100 messages, and retrieval error rates are averaged over 10^4 trials.

Another point of comparison for the fault tolerance of the proposed memory is the following scenario. Ignoring any complexity aspect, a simple way to tolerate storage faults is to protect the data to be stored using an error-correction code, thus abstracting away the unreliability of the data storage from the retrieval algorithm. Let us consider an ideal scenario where we are able to use a code that achieves the capacity $C_{\text{dev}}(\psi)$ of the deviation channel. We first decode the code, and then use the now reliable implementation of the retrieval algorithm to obtain the erased symbols. The efficiency $\eta^{\text{C}}(\psi)$ of such a scheme is then at most $\eta^{\text{C}}(\psi) = C_{\text{dev}}(\psi) \cdot \eta^*(0)$, and similarly in the message-wise case. It is important to note that this scenario does not necessarily represent the optimal efficiency because a reliable implementation does not perform optimal information storage (i.e., $\eta^*(0) < 1$). As shown in Fig. 5, the efficiency of the proposed associative memory is close to this ideal benchmark, indicating that it is very robust to storage faults.

C. Efficiency With State Vector Deviations

Fig. 6 shows the maximum storage efficiency achieved by the memory in terms of the state vector deviation channel parameter. All curves use $\ell = 256$. We first see that as can be expected, the message-wise efficiency (η_m^*) for $c = 16$ degrades faster than for $c = 8$, which is due to the fact that a larger number of erased symbols increases the likelihood that deviations will prevent the algorithm from converging. This dependency on c_e is removed when we consider η^*.

For $\ell = 256$, the memory can function with almost no efficiency degradation up to $\phi = 10^{-3}$. As discussed earlier, a few additional iterations allow achieving slightly larger ϕ values at a given efficiency. For example, at $\eta = 0.7$, the memory with $c = 16$ and $\ell = 256$ tolerates a deviation rate of $\phi = 1.55 \cdot 10^{-3}$ when using 4 iterations, but can reach $\phi = 2.8 \cdot 10^{-3}$ when using 6 iterations. We also observe that the number of stored messages that maximizes the efficiency and the corresponding retrieval probability both decrease as ϕ increases, similarly to the case of adjacency deviations.

VI. Conclusion

We studied the fault-tolerance of an associative memory based on a previously introduced c-partite graph model [10]. We presented a detailed implementation strategy and a model describing how circuit faults can introduce *deviations* in the algorithm. These deviations can be grouped into those affecting the representation of the graph's adjacency relationships, and those affecting the state of the retrieval algorithm.

Based on a performance analysis of the faulty algorithm, we showed how structural parameters can be chosen to optimize the storage efficiency of the memory in the presence of deviations on the adjacency data. We also showed that there is little degradation in the storage efficiency of the memory even when 1% of the adjacency storage is affected by deviations. Furthermore the algorithm can tolerate deviation probabilities on the order of 10^{-3} on the algorithm state for some realistic structural parameters. This fault tolerance is achieved without adding any complexity to the retrieval algorithm.

Our results therefore show that these associative memories can lend themselves to efficient circuit implementations with reduced safety margins.

Appendix

Proof of Theorem 1: We assume the set of stored messages $\mathcal{M}$ to be uniform i.i.d. In order to simplify this technical proof, we first point out that it is asymptotically equivalent to sample M messages uniformly at random with or without replacement. As a matter of fact, the probability not to have a repetition when drawing M messages i.i.d. uniform at random among ℓ^c possible ones is $\prod_{i=1}^{M-1} 1 - \frac{i}{\ell^c} \geq (1 - \frac{M-1}{\ell^c})^{M-1} \sim \exp(-\frac{(M-1)^2}{\ell^c}) \sim 1$ since $M^2 = o(\ell^c)$.

Let us denote by $W_{(i,j)(i',j')}$ the adjacency matrix element indicating whether an edge exists between the j-th vertex of group i and the j'-th vertex of group i'. Fix $i, i' \in [0; c-1], i \neq i'$. By construction of W, we have $\forall j, j'$:

$$\begin{aligned}
&\mathbb{P}(W_{(i,j)(i',j')} = 1) \\
&\quad = \mathbb{P}(\exists m \in \mathcal{M}, m_i = j \wedge m_{i'} = j') \\
&\quad = 1 - \mathbb{P}(\forall m \in \mathcal{M}, m_i \neq j \vee m_{i'} \neq j') \\
&\quad = 1 - \mathbb{P}(m_i \neq j \vee m_{i'} \neq j', m \in \mathcal{M})^M \\
&\qquad \text{due to } \mathcal{M} \text{ i.i.d.} \\
&\quad = 1 - (1 - \mathbb{P}(m_i = j \wedge m_{i'} = j', m \in \mathcal{M}))^M \\
&\quad = 1 - \left(1 - \frac{1}{\ell^2}\right)^M \quad \text{due to } \mathcal{M} \text{ uniform.}
\end{aligned}$$

The expression above represents the edge density, denoted by d. Let us now fix two edges $W_{(i_0,j_0)(i_1,j_1)}$ and $W_{(i_2,j_2)(i_3,j_3)}$ where $i_0 \neq i_1$, $i_2 \neq i_3$. We thus have: $\mathbb{P}(W_{(i_0,j_0)(i_1,j_1)} = 1)\mathbb{P}(W_{(i_2,j_2)(i_3,j_3)} = 1) = [1 - (1 - \frac{1}{\ell^2})^M]^2 = d^2$.

Denote by E_2 the joint event "$W_{(i_0,j_0)(i_1,j_1)} = 1 \wedge W_{(i_2,j_2)(i_3,j_3)} = 1$". We first consider the case where i_0, i_1, i_2 and i_3 are mutually distinct. We have:

$$\begin{aligned}
\mathbb{P}(E_2) = \mathbb{P}(W_{(i_0,j_0)(i_1,j_1)} = 1 \mid W_{(i_2,j_2)(i_3,j_3)} = 1) \\
\cdot \mathbb{P}(W_{(i_2,j_2)(i_3,j_3)} = 1). \quad (27)
\end{aligned}$$

Since the stored messages are i.i.d., we directly obtain: $\mathbb{P}(E_2) = d^2$.

Let us now consider the case where two edges originate from the same group, that is $i_0 = i_3$, with i_0, i_1, i_2 mutually distinct. Note that the other configurations can be obtained from this one by reordering (or relabeling) the indices. Introduce E the event "$\exists m \in \mathcal{M}, m_{i_0} = j_0 \wedge m_{i_1} = j_1 \wedge m_{i_2} = j_2$". We obtain:

$$\begin{aligned}
&\mathbb{P}(E_2)\\
&\quad = \mathbb{P}(\exists m \in \mathcal{M}, m_{i_0} = j_0 \wedge m_{i_1} = j_1 \wedge\\
&\quad \exists m' \in \mathcal{M}, {m'}_{i_0} = j_0 \wedge {m'}_{i_2} = j_2)\\
&\quad = \mathbb{P}(E \vee \neg E \wedge \exists m \in \mathcal{M}, m_{i_0} = j_0 \wedge m_{i_1} = j_1 \wedge\\
&\quad \exists m' \in \mathcal{M}, {m'}_{i_0} = j_0 \wedge {m'}_{i_2} = j_2)\\
&\text{(since the two events are disjoint :)}\\
&\quad = \mathbb{P}(E) + \mathbb{P}(\neg E \wedge \exists m \in \mathcal{M}, m_{i_0} = j_0 \wedge m_{i_1} = j_1 \wedge\\
&\quad \exists m' \in \mathcal{M}, {m'}_{i_0} = j_0 \wedge {m'}_{i_2} = j_2)\\
&\quad = \mathbb{P}(E) + \mathbb{P}(\neg E \wedge \exists m \in \mathcal{M}, m_{i_0} = j_0 \wedge m_{i_1} = j_1 \wedge\\
&\quad \exists m' \in \mathcal{M}, m' \neq m, {m'}_{i_2} = j_2 \wedge {m'}_{i_3} = j_3)\\
&\quad \leq \mathbb{P}(E) + \mathbb{P}(\exists m \in \mathcal{M}, m_{i_0} = j_0 \wedge m_{i_1} = j_1 \wedge\\
&\quad \exists m' \in \mathcal{M}, m' \neq m, {m'}_{i_2} = j_2 \wedge {m'}_{i_3} = j_3)\\
&\quad \leq \mathbb{P}(E) + d^2.
\end{aligned}$$

By considering similar reasoning to that which lead to the formula for d, we obtain that $\mathbb{P}(E) = 1 - (1 - \frac{1}{\ell^3})^M$. Under the assumption $M = o(\ell^3)$ and $M = \omega(\ell)$, we obtain: $\mathbb{P}(E) = o(d^2)$. To prove this result, we consider two cases: a) $M = o(\ell^2)$ and b) $M = \Omega(\ell^2)$. In case a), we obtain easily that $d \sim M/\ell^2$ and $\mathbb{P}(E) \sim M/\ell^3$. Thus $d^2/\mathbb{P}(E) \sim M/\ell = \omega(1)$. In case b), d is eventually larger than a non-zero constant, and since $\mathbb{P}(E)$ tends to zero, we conclude that $\mathbb{P}(E) = o(d^2)$. And thus $\mathbb{P}(E_2) \leq_{\ell \to \infty} d^2$.

For the lower bound, we use (27) and introduce $\mathcal{M}_0 = \{m \in \mathcal{M}, m_{i_0} = j_0 \wedge m_{i_2} = j_2\}$:

$$\begin{aligned}
&\mathbb{P}(W_{(i_0,j_0)(i_1,j_1)} = 1 \mid W_{(i_0,j_0)(i_2,j_2)} = 1)\\
&\quad \cdot \mathbb{P}(W_{(i_0,j_0)(i_2,j_2)} = 1)\\
&\qquad = \mathbb{P}(\exists m \in \mathcal{M}, m_{i_0} = j_0 \wedge m_{i_1} = j_1\\
&\qquad\quad \mid \exists m' \in \mathcal{M}, {m'}_{i_0} = j_0 \wedge {m'}_{i_2} = j_2) \cdot d\\
&\qquad = \mathbb{P}(\exists m' \in \mathcal{M}_0, {m'}_{i_1} = j_1 \vee \exists m \in \mathcal{M} - \mathcal{M}_0,\\
&\qquad\quad m_{i_0} = j_0 \wedge m_{i_1} = j_1 \mid \#(\mathcal{M}_0) \geq 1) \cdot d\\
&\qquad = d \sum_{n_0=1}^{M} \mathbb{P}(\exists m' \in \mathcal{M}_0, {m'}_{i_1} = j_1 \vee \exists m \in \mathcal{M} - \mathcal{M}_0,\\
&\qquad\quad m_{i_0} = j_0 \wedge m_{i_1} = j_1 \mid \#(\mathcal{M}_0) = n_0)\\
&\qquad\quad \cdot \mathbb{P}(\#(\mathcal{M}_0) = n_0 \mid \#(\mathcal{M}_0) \geq 1)\\
&\qquad = d \sum_{n_0=1}^{M} \left(1 - \left(1 - \frac{1}{\ell}\right)^{n_0}\right) \left(1 - \left(1 - \frac{1}{\ell^2}\right)^{M-n_0}\right)\\
&\qquad\quad \cdot \mathbb{P}(\#(\mathcal{M}_0) = n_0 \mid \#(\mathcal{M}_0) \geq 1)\\
&\qquad \geq d \sum_{n_0=1}^{M} \left(1 - \left(1 - \frac{1}{\ell^2}\right)^{M}\right) \mathbb{P}(\#(\mathcal{M}_0) = n_0\\
&\qquad\quad \mid \#(\mathcal{M}_0) \geq 1)\\
&\qquad = d^2
\end{aligned}$$

We conclude that for all cases:

$$\begin{aligned}
\mathbb{P}(E_2) &\underset{\ell \to \infty}{\sim} d^2\\
&= \mathbb{P}(W_{(i_0,j_0)(i_1,j_1)} = 1)\mathbb{P}(W_{(i_2,j_2)(i_3,j_3)} = 1).
\end{aligned}$$

Therefore, the existence of two distinct edges can be asymptotically regarded as independent.

REFERENCES

[1] F. Leduc-Primeau, V. Gripon, M. G. Rabbat, and W. J. Gross, "Cluster-based associative memories built from unreliable storage," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, May 2014.

[2] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *Proc. 17th Ann. Int. Symp. Comput. Architect.*, 1990, pp. 364–373.

[3] C. S. Lin, D. C. P. Smith, and J. M. Smith, "The design of a rotating associative memory for relational database applications," *ACM Trans. Database Syst.*, vol. 1, pp. 53–65, Mar. 1976.

[4] A. Papadogiannakis, M. Polychronakis, and E. P. Markatos, "Improving the accuracy of network intrusion detection systems under load using selective packet discarding," in *Proc. 3rd Eur. Workshop on Syst. Secur.*, New York, NY, USA, 2010, vol. ser. EUROSEC'10, pp. 15–21.

[5] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006.

[6] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," in *Proc. Nat. Acad. Sci.*, Apr. 1982, vol. 79, pp. 457–464.

[7] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. Venkatesh, "The capacity of the Hopfield associative memory," *IEEE Trans. Inf. Theory*, vol. 33, no. 4, pp. 461–482, Jul. 1987.

[8] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins, "Non-holographic associative memory," *Nature* vol. 222, no. 5197, pp. 960–962, 06, 1969 [Online]. Available: http://dx.doi.org/10.1038/222960a0

[9] G. Palm, "Neural associative memories and sparse coding," *Neural Netw.*, vol. 37, no. 0, pp. 165–171, 2013.

[10] V. Gripon and C. Berrou, "Sparse neural networks with large learning diversity," *IEEE Trans. Neural Netw.*, vol. 22, no. 7, pp. 1087–1096, 2011.

[11] V. Gripon and C. Berrou, "A simple and efficient way to store many messages using neural cliques," in *Proc. IEEE Symp. Computat. Intell., Cogn. Algorithms, Mind, Brain*, Paris, France, Apr. 2011, pp. 54–58.

[12] V. Gripon and C. Berrou, "Nearly-optimal associative memories based on distributed constant weight codes," in *Proc. Inf. Theory Appl. Workshop*, San Diego, CA, USA, Feb. 2012, pp. 269–273.

[13] S. Ghosh and K. Roy, "Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era," *Proc. IEEE*, vol. 98, no. 10, pp. 1718–1751, Oct. 2010.

[14] S. S. Sapatnekar, "Overcoming variations in nanometer-scale technologies," *IEEE J. Emerg. Select. Topics Circuits Syst.*, pp. 5–18, Mar. 2011.

[15] G. Palm, "English On the storage capacity of an associative memory with randomly distributed storage elements," *English Biolog. Cybern.*, vol. 39, no. 2, pp. 125–127, 1981.

[16] G. Bolt, J. Austin, and G. Morgan, "Operational fault tolerance of the ADAM neural network system," in *Proc. 2nd Int. Conf. Artif. Neural Netw.*, Nov. 1991, pp. 285–289.

[17] F. Sommer and P. Dayan, "Bayesian retrieval in associative memories with storage errors," *IEEE Trans. Neural Netw.*, vol. 9, no. 4, pp. 705–713, Jul. 1998.

[18] V. Kryzhanovsky, B. Kryzhanovsky, and A. Fonarev, "Application of potts-model perceptron for binary patterns identification," in *Proc. 18th Int. Conf. Artif. Neural Netw. Part I*, Berlin, Heidelberg, Germany, 2008, pp. 553–561, ser. ICANN'08, Springer-Verlag.

[19] A. Karbasi, A. H. Salavati, and A. Shokrollahi, "Iterative learning and denoising in convolutional neural associative memories," in *Proc. 30th Int. Conf. Mach. Learn.*, Jun. 2013.

[20] A. Karbasi, A. H. Salavati, A. Shokrollahi, and L. R. Varshney, "Noise facilitation in associative memories of exponential capacity," *Neural Computat.*, vol. 26, no. 11, pp. 2493–2526, Nov. 2014.

[21] Z. Yao, V. Gripon, and M. G. Rabbat, "A massively parallel associative memory based on sparse neural networks," [Online]. Available: arXiv:1303.7032v2. Mar. 2013

[22] H. Jarollahi, N. Onizawa, V. Gripon, and W. J. Gross, "Reduced-complexity binary-weight-coded associative memories," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, 2013, pp. 2523–2527.
[23] R. Dekker, F. Beenker, and L. Thijssen, "A realistic fault model and test algorithms for static random access memories," *IEEE Trans. Comput. -Aided Design Integr. Circuits Syst.*, vol. 9, no. 6, pp. 567–572, 1990.
[24] T. Karnik and P. Hazucha, "Characterization of soft errors caused by single event upsets in CMOS processes," *IEEE Trans. Depend. Secure Comput.*, vol. 1, no. 2, pp. 128–143, Apr.–Jun. 2004.
[25] L. Varshney, "Performance of LDPC codes under faulty iterative decoding," *IEEE Trans. Inf. Theory*, vol. 57, no. 7, pp. 4427–4444, Jul. 2011.
[26] F. Leduc-Primeau and W. J. Gross, "Faulty Gallager-B decoding with optimal message repetition," in *Proc. 50th Allerton Conf. Commun., Contr., Comput.*, Oct. 2012.
[27] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proc. Design Autom. Conf.*, 2003, pp. 338–342.
[28] F. Schwenker, F. T. Sommer, and G. Palm, "Iterative retrieval of sparsely coded associative memory patterns," *Neural Netw.*, vol. 9, no. 3, 1996.
[29] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. New York, NY, USA: Wiley-Intersci., 2006.
[30] A. Knoblauch, G. Palm, and F. T. Sommer, "Memory capacities for synaptic and structural plasticity," *Neural Computat.*, vol. 22, no. 2, pp. 289–341, Feb. 2010.
[31] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming Moore's law through energy efficient integrated circuits," *Proc. IEEE*, vol. 98, no. 2, pp. 253–266, Feb. 2010.

François Leduc-Primeau (S'09) received the B.Eng. and M.Eng. degrees in computer engineering from McGill University, Montréal, Canada, in 2007 and 2009, respectively.

He is currently finishing the Ph.D. degree at McGill University. His research interests include error-correction codes, algorithms and systems for telecommunications and signal processing, and novel approaches for improving the energy efficiency of digital systems.

Vincent Gripon (S'10–M'12) received the M.S. degree from École Normale Supérieure of Cachan and the Ph.D. degree from Télécom Bretagne.

He is a permanent researcher with Télécom Bretagne (Institut Mines-Télécom), Brest, France. His research interests include information theory, neuroscience, and theoretical and applied computer science. His intent is to propose models of neural networks inspired by information theory principles, what could be called informational neurosciences. He is also the co-creator and organizer of an online programming contest named TaupIC which targets French top undergraduate students.

Michael G. Rabbat (S'02–M'07–SM'15) received the B.Sc. degree from the University of Illinois, Urbana-Champaign, in 2001, the M.Sc. degree from Rice University, Houston, TX, USA, in 2003, and the Ph.D. degree from the University of Wisconsin, Madison, USA, in 2006, all in electrical engineering.

He joined McGill University, Montréal, QC, Canada, in 2007, and he is currently an Associate Professor. During the 2013–2014 academic year, he held visiting positions at Télécom Bretegne, Brest, France; the Inria Bretagne-Atlantique Reserch Centre, Rennes, France; and KTH Royal Institute of Technology, Stockholm, Sweden. He was a Visiting Researcher at Applied Signal Technology, Inc., Sunnyvale, CA, USA, during summer 2003. His research interests include distributed algorithms for optimization and inference, consensus algorithms, and network modelling and analysis, with applications in distributed sensor systems, large-scale machine learning, statistical signal processing, and social networks.

Dr. Rabbat coauthored the paper which received the Best Paper Award (Signal Processing and Information Theory Track) at the 2010 IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS). He currently serves as Senior Area Editor for the IEEE SIGNAL PROCESSING LETTERS and as Associate Editor for IEEE TRANSACTIONS ON SIGNAL AND INFORMATION PROCESSING OVER NETWORKS and IEEE TRANSACTIONS ON CONTROL OF NETWORK SYSTEMS.

Warren J. Gross (S'92–M'04–SM'10) received the B.A.Sc. degree in electrical engineering from the University of Waterloo, Waterloo, Ontario, Canada, in 1996, and the M.A.Sc. and Ph.D. degrees from the University of Toronto, Toronto, Ontario, Canada, in 1999 and 2003, respectively.

Currently, he is an Associate Professor with the Department of Electrical and Computer Engineering, McGill University, Montréal, Québec, Canada. His research interests are in the design and implementation of signal processing systems and custom computer architectures.

Dr. Gross is currently Chair of the IEEE Signal Processing Society Technical Committee on Design and Implementation of Signal Processing Systems. He has served as Technical Program Co-Chair of the IEEE Workshop on Signal Processing Systems (SiPS 2012) and as Chair of the IEEE ICC 2012 Workshop on Emerging Data Storage Technologies. He served as an Associate Editor for the IEEE TRANSACTIONS ON SIGNAL PROCESSING. He has served on the Program Committees of the IEEE Workshop on Signal Processing Systems, the IEEE Symposium on Field-Programmable Custom Computing Machines, the International Conference on Field-Programmable Logic and Applications, and as the General Chair of the 6th Annual Analog Decoding Workshop. He is a licensed Professional Engineer in the Province of Ontario.