Stochastic Neuromorphic Circuits for Solving MAXCUT

Bradley H. Theilman*, Yipu Wang†, Ojas Parekh†, William Severa*, J. Darby Smith*, and James B. Aimone*

*Neural Exploration and Research Lab †Discrete Math and Optimization
Sandia National Laboratories, Albuquerque, New Mexico
{bhtheil, yipwang, odparek, wmsever, jsmit16, jbaimon}@sandia.gov

Abstract—Finding the maximum cut of a graph (MAXCUT) is a classic optimization problem that has motivated parallel algorithm development. While approximate algorithms to MAXCUT offer attractive theoretical guarantees and demonstrate compelling empirical performance, such approximation approaches can shift the dominant computational cost to the stochastic sampling operations. Neuromorphic computing, which uses the organizing principles of the nervous system to inspire new parallel computing architectures, offers a possible solution. One ubiquitous feature of natural brains is stochasticity: the individual elements of biological neural networks possess an intrinsic randomness that serves as a resource enabling their unique computational capacities. By designing circuits and algorithms that make use of randomness similarly to natural brains, we hypothesize that the intrinsic randomness in microelectronics devices could be turned into a valuable component of a neuromorphic architecture enabling more efficient computations. Here, we present neuromorphic circuits that transform the stochastic behavior of a pool of random devices into useful correlations that drive stochastic solutions to MAXCUT. We show that these circuits perform favorably in comparison to software solvers and argue that this neuromorphic hardware implementation provides a path for scaling advantages. This work demonstrates the utility of combining neuromorphic principles with intrinsic randomness as a computational resource for new computational architectures.

I. INTRODUCTION

Despite the heavy requirements for noise-free operation placed on the components of conventional computers, random numbers play a crucially important role in many parallel computing problems arising in different scientific domains. Because current random number generation occurs largely in software, the required randomness in these systems is plagued by the same memory-processing bottlenecks that limit ordinary computation. Current work in material science and microelectronics is demonstrating the feasibility of constructing stochastic microelectronic devices with controllable statistics that make use of randomness similarly to natural brains, which could emerge from combining the computational principles of natural brains with physical sources of intrinsic randomness. This would allow the natural stochasticity of electronic devices to play a part in large-scale parallel computations, relieving the burden imposed by requiring absolute reliability.

Realizing the potential of probabilistic neural computation requires rethinking conventional parallel algorithms to incorporate stochastic elements from the bottom up. Additionally, techniques for controlling the randomness must be developed so that useful random numbers can be produced efficiently from the desired distributions. In this work, we propose neuromorphic circuits that demonstrate the capacity for intrinsic randomness to solve parallel computing problems and techniques for controlling device randomness to produce useful random numbers.

MAXCUT is a well known, NP-complete problem that has practical applications and serves as a model problem and testbed for both classical and beyond-Moore algorithm development [4], [9], [16], [20]. The problem requires partitioning the vertices of a graph into two disjoint classes such that the number of edges that span classes is maximized. MAXCUT has several stochastic approximation algorithms, which makes it an ideal target for developing new architectures leveraging large-scale parallel stochastic circuit elements for computational benefit.

Stochastic approximation algorithms are evaluated via their approximation ratio, which is the ratio of the expected value
of a stochastically generated solution to the maximum possible value. The stochastic approximation to MAXCUT with the largest known approximation ratio is the Goemans-Williamson algorithm [9]. The Goemans-Williamson algorithm provides the best approximation ratio achievable by any polynomial-time algorithm under the Unique Games Conjecture [20]. To generate solutions, this algorithm requires sampling from a Gaussian distribution with a specific covariance matrix obtained by solving a semi-definite program related to the adjacency matrix of the graph. Our first neural circuit implements this sampling step by using simple, well-known neuron models to transform uniform device randomness into the required distribution. This demonstrates the use of neuromorphic principles to transform an intrinsic source of randomness into a computationally useful distribution.

Another stochastic approximation for MAXCUT is the Trevisan algorithm [29], [31]. Despite having a worse theoretical approximation ratio, in practice this algorithm generates solutions on par with the Goemans-Williamson algorithm [22]. To generate solutions, this algorithm requires computing the minimum eigenvector of the normalized graph adjacency matrix. Our second neuromorphic circuit implements this algorithm using the same circuit motif as above to generate random numbers with specific correlations, but instead of sampling cuts from this distribution, we use these numbers to drive a synaptic plasticity rule (Oja’s rule) inspired by the Hebbian principle in neuroscience [25]. This learning rule can be shown to converge to the desired eigenvector, from which the solution can be sampled. This circuit solves the MAXCUT problem entirely within the circuit, without requiring any external preprocessing, demonstrating the capacity of neuromorphic circuits driven by intrinsic randomness to solve parallel computationally-relevant problems.

Neuromorphic computing is having increasing impacts on non-cognitive problems relevant for parallel computing [1]. Unlike other hardware approaches to MAXCUT, our contributions directly instantiate state-of-the-art MAXCUT approximation algorithms on arbitrary graphs without requiring costly reconfiguration or conversion of the problem to an Ising model with pairwise interactions [10], [11], [32]. Our use of hardware resources is scalable, requiring one neuron and one random device per vertex, and thus more efficient than parallel implementations of MAXCUT using GPUs [8]. These properties make our contributions valuable to the expanding field of beyond-Moore parallel algorithms.

II. MAXCUT ALGORITHMS

A. The Goemans-Williamson MAXCUT Algorithm

Given an n-vertex, m-edge graph $\mathcal{G} = (V, E)$ with vertex set $V$ and edge set $E$, the MAXCUT problem seeks a partition of the vertices into two disjoint subsets, $V = V_+ \cup V_-$ such that the number of edges that cross between the two subsets is maximized. By assigning either of the values $\{-1, 1\}$ to each vertex, the MAXCUT problem is equivalent to maximizing the function

$$\max_v \frac{1}{2} \sum_{ij \in E} A_{ij}(1 - v_i v_j)$$

subject to $v \in \{-1, 1\}^n$. (1)

Here, $A_{ij}$ is the adjacency matrix of the graph $\mathcal{G}$. Let $\text{OPT}(\mathcal{G})$ be the maximum value of this function.

MAXCUT is known to be NP-complete [16]. Goemans and Williamson [9] described a relaxation of the above integer programming problem that yields an efficient approximation to MAXCUT with an approximation ratio of 0.878. The relaxation replaces the integer programming problem with a semidefinite programming problem given by

$$\max_w \frac{1}{2} \sum_{ij} A_{ij}(1 - w_i \cdot w_j)$$

subject to $w_i \in S^{n-1}$, (2)

where $S^{n-1}$ is the $(n-1)$-dimensional unit sphere in $\mathbb{R}^n$. Let $\text{SDP}(\mathcal{G})$ be value of the optimal solution of this semidefinite programming problem. Note that $\text{OPT}(\mathcal{G}) \leq \text{SDP}(\mathcal{G})$.

The solution of this semidefinite programming problem is a set of unit vectors $w_i$, one for each vertex in the graph. Given these vectors, a graph cut is generated by taking a random hyperplane through the origin and assigning the value +1 to vertices with vectors above the plane and −1 to vertices with vectors below the plane.

One can see that the Goemans-Williamson algorithm has two steps: in the first step we solve an SDP, and in the second we round each unit vector $w_i$ to an integer $z_i \in \{-1, +1\}$, where $i \in V$. Bertsimas and Ye [6] observed that the rounding step can be implemented by sampling dependent standard normal random variables, with one variable per vertex. Specifically, suppose for each vertex $i$ we have a random variable $X_i$ following the standard normal distribution, and furthermore for each pair of vertices $i$ and $j$ the covariance between $X_i$ and $X_j$ is $w_i \cdot w_j$, where $w_i$ and $w_j$ are the unit vectors in the solution to the SDP. One can show that such a set of dependent random variables exists. Now define a (random) cut by assigning +1 to vertices $i$ where $X_i$ is positive and assigning −1 to vertices $i$ where $X_i$ is negative. One can show that the resulting cut has the same approximation guarantees as the cut returned by the Goemans-Williamson algorithm. Hence in this paper we will sometimes refer to the rounding step as the sampling step.

B. The Trevisan (Simple Spectral) Algorithm

The Trevisan algorithm is another random approximation algorithm for MAXCUT [31]. Though it has a worse theoretical approximation ratio (0.631) [29] than the Goemans-Williamson algorithm, in practice it can perform just as well and has speed advantages [22]. Here we consider a slight modification of the full Trevisan algorithm we refer to as the Trevisan Simple Spectral algorithm [22].

Given a graph $G = (V, E)$ with adjacency matrix $A$ and diagonal degree matrix $D$, we compute the normalized

780

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
adjacency matrix \( A = D^{-1/2}AD^{1/2} \). Next, the eigenvector corresponding to the minimum eigenvalue of the matrix \( I + A \) is computed. The graph cut is obtained by thresholding the values of this eigenvector by sign. If \( u \) is the minimum eigenvector of \( A \), then the graph cut is given by

\[
v_i = \begin{cases} 
-1 & u_i \leq 0 \\
1 & u_i > 0 
\end{cases}
\]  

(5)

III. NEUROMORPHIC CONCEPTS

A. Stochastic Devices

Physical microelectronics display intrinsic stochasticity due to the physics behind their operation. Typically this stochastic behavior is under active research to develop devices with tunable statistics for probabilistic computing applications [5], [23], [26]. In our work, we idealize stochastic devices as analogous to “coin flips” such that at any given time, the device can be in one of two states (“heads” or “tails”; “0” or “1”) with a specific probability. In our circuits, we assume random devices behave as fair coins. That is, each state has a probability of 0.5. Thus, a random device is modeled as a source for a random bit stream with equal probabilities of 0 or 1. Magnetic tunnel junctions [19], [26] and tunnel diodes [5] are examples of two classes of devices actively being developed to meet these requirements.

B. Leaky Integrate and Fire Neurons

The leaky integrate and fire (LIF) neuron is a simplified model of biological neurons, readily implemented in hardware [14], that captures a biological neuron’s capacity for temporal integration of synaptic inputs along with discontinuous spiking. An individual LIF neuron numerically integrates weighted input events into a state variable called the membrane potential. When the integrated membrane potential reaches some specified threshold, a spike event is emitted and the membrane potential is reset to some defined value. In between spike events, the membrane potential evolves according to the first order differential equation

\[
C \frac{dV}{dt} = -\frac{V}{R} + I_{\text{tot}}.
\]  

(6)

Here, \( V \) is the membrane potential, \( C \) and \( R \) are the membrane capacitance and leak resistance, which determine the temporal response properties of the neuron. \( I_{\text{tot}} \) is the total synaptic input current, given by a weighted sum of the individual synaptic inputs onto a neuron

\[
I_{\text{tot}} = \sum_{\alpha} W_{i\alpha} s_\alpha
\]  

(7)

The variables \( s_\alpha \) are binary variables indicating the presence (1) or absence (0) of a presynaptic spike event at any given time step. The weights \( W_{i\alpha} \) give the magnitude of the current injected into neuron \( i \) due to a presynaptic spike event from input \( \alpha \).

When a single LIF neuron receives large numbers of independent stochastic input events, the membrane potential approximates a one-dimensional random walk [21]. The mean and variance of \( V \) under these dynamics are given by

\[
\langle V \rangle = R \langle I_{\text{tot}} \rangle
\]  

(8)

and

\[
\text{Var}(V) = \frac{R}{C} \text{Var}(I_{\text{tot}}).
\]  

(9)

Thus, LIF neurons can be seen as a means to transform random inputs into a Gaussian process with specified statistics.

C. LIF Covariances

Shared input between two LIF neurons induces correlations between their membrane potentials. The expression for the covariance between the membrane potentials of neurons \( i \) and \( j \)

\[
\text{Cov}(V_i, V_j) = \frac{R}{C} \sum_{\alpha \beta} W_{i\alpha} W_{j\beta} \text{Cov}(s_\alpha, s_\beta).
\]  

(10)

In other words, the LIF membrane covariances are a linear transformation of the covariances of the random inputs. Thus, the LIF neuron population transforms random input events into a set of Gaussian processes with covariance proportional to the Gram matrix of the weight vectors. In what follows, choosing the weights appropriately allows this circuit motif to supply random samples with the appropriate covariances for the stochastic MAXCUT approximation algorithms.

D. Synaptic Plasticity: Oja’s Rule

In neuroscience, synaptic plasticity refers to the ability of the connection strengths between neurons to change over time as a function of neural activity. The most commonly applied synaptic plasticity principle is the Hebbian learning...
principle [13], which is summarized succinctly with the adage “neurons that fire together, wire together.” Synaptic plasticity rules are defined by differential equations relating the change in weights to neuronal activity state variables such as the membrane potential. If $\mathbf{w}$ is the weight vector between presynaptic neuron activity $\mathbf{x}$ and postsynaptic neuron activity $y$, the simplest instantiation of the Hebbian principle is given the formula

$$\Delta \mathbf{w} = \mathbf{y} \mathbf{x}$$  \hspace{1cm} (11)$$

As stated, this rule is unstable, meaning that the magnitude of the weights will diverge over time. Oja presented a modification to this plasticity rule that preserved the Hebbian principle but enforced weight stability [24]. Oja’s rule is given by the formula

$$\Delta \mathbf{w} = \mathbf{y}(\mathbf{x} - \mathbf{y} \mathbf{w})$$  \hspace{1cm} (12)$$

Oja proved that under mild assumptions this rule forces the weight vector to converge to the first principle component of the covariance matrix of the inputs, or equivalently, the eigenvector corresponding to the largest eigenvalue.

By considering anti-Hebbian plasticity, Oja derived a related, stabilized learning rule that converges to the minimum eigenvector of the covariance matrix [25]

$$\Delta \mathbf{w} = -\mathbf{y} \mathbf{x} + (y^2 + 1 - \mathbf{w}^T \mathbf{w}) \mathbf{w}$$  \hspace{1cm} (13)$$

By providing inputs with covariance proportional to the adjacency matrix of the graph as used in Trevisan’s algorithm, Oja’s anti-Hebbian rule can find the minimum eigenvector of this matrix, yielding an approximate solution to MAXCUT.

IV. CIRCUITS

A. LIF-Goemans-Williamson

Figure 1 shows a neural circuit that implements the sampling step of the Goemans-Williamson algorithm. The requirement is to generate binarized samples from a Gaussian distribution with specified covariance matrix $C$. We refer to this circuit with the abbreviation LIF-GW.

For a graph $G$, the Goemans-Williamson SDP is solved to yield a set of $n$ unit vectors in $r$ dimensions, where $r$ is the rank of the solution and $n$ is the number of vertices. These vectors can be combined into the $n$ by $r$ dimensional matrix $W_{GW}$.

The circuit consists of a pool of $r$ random devices connected to $n$ LIF neurons. The synaptic weights between the devices and the neurons are chosen proportional to the corresponding entries in $W_{GW}$. The precise magnitudes of these weights are not critical; what matter are their relative values, as these ratios determine the LIF covariances as in equation (10). This allows the circuit to be adapted to specific hardware implementations imposing constraints on the range of available weights. For our tests, we used a fixed rank of 4 for all graphs.

Choosing the weights proportional to the solution to the SDP yields membrane covariances proportional to those required by the Goemans-Williamson algorithm. The spiking threshold of the LIF neurons implements a rounding and sampling operation that we map to graph cuts. Neurons that spike together on a given time step map to vertices on one side of the cut, and neurons that are collectively silent on a given time step map to vertices on the other side of the cut.

B. LIF-Trevisan

The second neural circuit (Figure 2) implements the simple spectral modification of Trevisan’s algorithm [22], [31] and we refer to it as either LIF-Trevisan or LIF-TR. Like the LIF-GW circuit, the first stage consists of a population of LIF neurons, one for each vertex in the graph, driven by a pool of random devices. Next, the output of the LIF population is fed onto a single LIF neuron. The output of this Stage-2 neuron is discarded; what matters is the weight vector linking the Stage-1 LIF population to Stage-2. This weight vector is controlled by Oja’s anti-Hebbian plasticity rule. This forces the weight vector $w$ to converge onto the minimum eigenvector of the LIF covariance matrix.

The LIF covariance matrix is determined by the connection weights between the random devices and the LIF population. These are set proportional to the Trevisan matrix, which is the sum $I + D^{-1/2}AD^{-1/2}$ of the identity plus the normalized adjacency matrix of the graph. In this way, the LIF-Trevisan circuit does not require solving an SDP offline.

V. RESULTS

We simulated these circuits by using standard techniques from computational neuroscience, namely, numerically integrating (6) and (13) using the forward Euler method. We then quantified their ability to generate graph cuts by interpreting

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{LIF-Trevisan circuit implementing a stochastic approximation to MAXCUT by combining hardware randomness with anti-Hebbian synaptic plasticity. The connection weights between the random device pool (left) and the LIF neurons are set proportional to the adjacency matrix of the graph. The activity of the LIF neurons drives synaptic plasticity on the weight vector onto an output neuron. The solution is sampled by thresholding this weight vector by sign: excitatory, positive weights correspond to one side of the cut and inhibitory, negative weights correspond to the other side. The output of the output neuron is ignored.}
\end{figure}
Figure 3. Maximum cut weight relative to software Goemans-Williamson solver (green triangle curve) as a function of the number of samples for Erdős-Rényi random graphs. Rows correspond to fixed numbers of vertices $n$ and columns correspond to fixed connection probabilities $p$. Panel title gives graph parameters $G(n,p)$. Error bars correspond to standard error of the mean over 10 independently generated graphs from each graph class. Blue circles: LIF-GW circuit; orange squares: LIF-TR circuit; green triangles: software solver; red X’s: random graph cuts. Blue and green curves overlap.
the spiking output (LIF-GW circuit) or the output weight vector (LIF-TR circuit) as a graph cut. Following [22] we evaluated the circuits on Erdős-Rényi random graphs with a number of vertices $n$ in $\{50, 100, 200, 350, 500\}$ and a connection probability $p$ in $\{0.1, 0.25, 0.5, 0.75\}$. We generated 10 distinct random graphs per $(n, p)$ combination, yielding 200 total graphs. We generated $2^{20}$ graph cuts per circuit, per graph. We compared the circuit-generated cut weights to cut weights generated by a generic SDP solver (PyManOpt [30]) and cut weights generated by a purely random assignment of vertices to sides of a cut. As described, circuits were driven by a simulated pool of random devices. Each device was assumed to have two states, and have a probability of 0.5 of being in any given state at each time step.

Figure 3 shows that, as expected, the LIF-GW circuit matches the performance of the generic solver. This also validates the proposed circuit motif using LIF neurons to translate hardware randomness into Gaussian processes with desired covariances. The LIF-Trevisan circuit shows performance that increases over time, approaching the performance of the solver, due to the on-line learning of the solution through Oja’s rule. In all cases, the LIF-Trevisan circuit eventually outperforms the random algorithm. The trajectory of the LIF-Trevisan circuit’s performance suggests that the rate of convergence of the plasticity depends on the graph parameters, but there is no indication that the performance of this circuit is saturated within the number of samples considered here.

We next evaluated our circuits on empirical graphs taken from the Network Repository [27]. We picked the same graphs tested in [22]. Figure 4 shows the performance of our circuits compared to SDP-solver and random cuts. Consistent with our results on Erdős-Rényi graphs, we found that the LIF-GW sampling circuit matched the performance of the software solver. We found that the LIF-Trevisan circuit was able to outperform randomly-generated cuts and, occasionally, exactly match the solver-generated cuts, after evolving the circuit with synaptic plasticity for sufficiently many samples. This is consistent with the results of [22], who found that in some cases the simplified approximation algorithms to MAXCUT matched or outperformed cuts generated by the full Goemans-Williamson algorithm on empirical graphs, even though the simplified algorithms have worse approximation guarantees. The maximum cut values for each circuit for each graph are presented in Table I, which agree with the maximum cut sizes found in [22] (rightmost column).

VI. DISCUSSION

We have presented two neuromorphic circuits that transform the activity of a pool of random devices into useful distributions that solve a computational problem, in this case, MAXCUT. Our approach combines insights from theoretical computer science, neuroscience, and materials science to show that probabilistic neural computing is a viable path to new computational architectures. Our circuits display competitive performance with traditional software solvers. Consistent with prior work [22], we find that the simple spectral Trevisan algorithm performs in practice nearly as well as the gold-standard Goemans-Williamson algorithm. Our results for the Trevisan circuit suggest that its performance can be expected to improve beyond the $2^{20}$ samples considered here. While in this work we considered only a single example problem, MAXCUT is a special case of a larger class of problems known as constraint satisfaction problems, which include problems like maximum directed cut (MAXDICUT) and maximum 2-satisfiability (MAX2SAT). As such, our circuits may extend to more general probabilistic neural approaches for solving discrete optimization problems. For instance, using results due to Goemans and Williamson [9], our LIF-GW circuit can implement sampling steps for algorithms for MAXDICUT and MAX2SAT that yield approximation ratios of 0.796 and 0.878, respectively.

Our circuits present a trade-off in neuromorphic implementations of combinatorial optimization. The LIF-GW circuit requires fewer random devices and delivers superb solution quality rapidly, but requires a substantial commitment of offline resources (i.e. solving a semi-definite program) to initialize. Conversely, the LIF-Trevisan circuit requires as many random devices as vertices in the graph, and takes many more samples to reach comparable performance. However, this circuit avoids offline computations, which are a significant fraction of the running time of these algorithms [22]. This prior work also used far fewer samples (100) to compare the algorithms than our $2^{20}$. While the number of samples required suggests a disadvantage, at hardware speeds the greater number of samples required will likely be a trivial increase in the running time compared to a software implementation. Current hardware implementations of LIF neurons operate with time constants on the order of 1 nanosecond [7], [12], and recent results suggest that it is possible to operate stochastic MTJs on the order of 10s of nanoseconds [26], but the ultimate temporal limits are an open research question. The convergence rate of the synaptic plasticity in our LIF-TR circuit depends on both the circuit parameters and graph structure in complicated ways, but this dependence could be formalized or optimized in future work. Furthermore, the requirement for greater numbers of random devices is likely not a limitation, as the current trajectory for implementing the stochastic devices required for these circuits shows promising scaling advantages [23].

Our circuits differ from other neuromorphic approaches to MAXCUT. Several prior implementations are based on Ising models [10], [11], [32]. While in principle, solving Ising models is equivalent to solving MAXCUT, in practice the required circuits may be difficult to instantiate in hardware due to geometric constraints on the interactions between neurons. To map arbitrary combinatorial optimization problems onto Ising models requires arbitrary pairwise interactions, while the geometry of 2D device layout strongly favors nearest-neighbor interactions [8]. This limits the general applicability of Ising model approaches, and these constraints also necessarily limit the ultimate speed and power efficiency of these implementations. In contrast, our circuits implement well-established algorithms with theoretical performance guarantees using
Figure 4. Maximum cut relative to solver as a function of the number of samples for empirical graphs taken from the Network Repository. Each panel represents a single graph, thus there are no error bars. Panel title identifies the graph dataset. Blue circles: LIF-GW circuit; orange squares: LIF-TR circuit; green triangles: software solver; red X’s: random graph cuts. Blue and green curves overlap.
Table I
MAXIMUM CUT VALUES FOR THE DIFFERENT NEUROMORPHIC CIRCUITS APPLIED TO EMPIRICAL GRAPHS FROM THE NETWORK REPOSITORY [27].

| Graph              | LIF-GW | LIF-TR | Solver | Random [22] |
|--------------------|--------|--------|--------|-------------|
| hamming6-2         | 992    | 972    | 992    | 957         | 992         |
| soc-dolphins       | 122    | 122    | 122    | 107         | 121         |
| inf-USAir97        | 107    | 97     | 107    | 89          | 107         |
| road-chesapeake    | 126    | 125    | 126    | 120         | 125         |
| johnson16-2-4      | 3036   | 2987   | 3036   | 2858        | 3036        |
| p-hat700-1         | 33350  | 31369  | 33351  | 31002       | 33050       |
| ia-infect-dublin   | 1751   | 1600   | 1750   | 1494        | 1664        |
| ca-netscience      | 635    | 579    | 634    | 522         | 611         |
| dwt-209            | 554    | 534    | 554    | 441         | 540         |
| dwt-503            | 1937   | 1740   | 1937   | 1493        | 1921        |
| ia-infect-hyper    | 1277   | 1262   | 1277   | 1182        | 1233        |
| email-enron-only   | 425    | 394    | 425    | 367         | 413         |
| Erdos991           | 1027   | 920    | 1027   | 791         | 934         |
| eco-stmarks        | 1765   | 1764   | 1765   | 1747        | 1190        |
| DD687              | 1786   | 1625   | 1783   | 1411        | 1680        |
| ENZYMES8           | 126    | 124    | 126    | 95          | 126         |

simple devices without stringent requirements on their physical implementations, because the computations are ultimately localized, feed-forward operations on stochastic bit streams.

Our simulations model random devices as perfectly fair coins generating random, independent bit streams. These assumptions are necessarily approximations to the true behavior of a random device, which may display the statistics of an unfair coin, show internal or external correlations, or display statistics that drift over time. These imperfections might have an impact on the performance of the circuits presented. The key circuit motif in each circuit implements the central limit theorem through the integration of large numbers of random devices. Thus, we expect robustness to deviations of individual devices from the idealized perfect coin as the number of devices grows. While there is a growing realization that stochastic devices can provide robust random bit streams [26], there currently are few standards for what makes a good true random number generator for randomized algorithms. For this reason, the circuits described here provide a much needed benchmark for device physicists to incorporate physically-detailed device models to assess the impact of device variability.

Neuromorphic computing is having a growing impact on graph algorithms [1]. Previous work has found neuromorphic solutions to graph problems such as max flow [18], cycle detection [18], shortest paths [2], [3], [17], and spanning trees [17]. This prior work has exploited connections between the graph structure of neural networks and the corresponding graph problems. In contrast, our work uses the statistical behavior of neural circuits and learning through synaptic plasticity to solve a new class of graph problems. Incorporating learning into neuromorphic circuits to solve specific computational problems is comparatively under-explored, and thus our work expands the neuromorphic acceleration of graph algorithms in a new direction.

Our approach presents a different strategy for incorporating neuroscientific insight into parallel computation. The most successful application of neuroscience principles to date occurs in the field of deep learning, which is based on connectionist principles inherited from early neuroanatomical studies. In contrast, our circuits use the integrative, statistical properties of neurons to achieve different kinds of computations. This was informed by early theoretical developments, like Oja’s rule for synaptic plasticity [24], [25]. These developments have been well-known for decades, but have not had as strong an influence on computation. Similar to how backpropagation was known for many years before physical implementations achieved the scale necessary to reveal its utility, we expect that recent advances in stochastic devices and neuromorphic hardware will reveal the utility of probabilistic neural computation.

ACKNOWLEDGMENT
The authors acknowledge financial support from the DOE Office of Science (ASCR / BES) for our Microelectronics Co-Design project COINFLIPS. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA0003525. This paper describes technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. SAND Number: SAND2022-13654 O

REFERENCES
[1] James B. Aimone, Prasanna Date, Gabriel A. Fonseca-Guerra, Kathleen E. Hamilton, Kyle Henke, Bill Kay, Garrett T. Kenyon, Shrun R. Kulkarni, Susan M. Mniszewski, Maryam Parsa, Sumedh R. Risbud, Catherine D.
