A Programmable and Reconfigurable CMOS Analog Hopfield Network for NP-Hard Problems

Pranav O. Mathews, *Graduate Student Member, IEEE*, and Jennifer O. Hasler, *Senior Member, IEEE*

***Abstract*— Analog Hopfield networks perform continuous energy minimization, leading to efficient and near-optimal solutions to nonpolynomial (NP)-hard problems. However, practical implementations suffer from scaling and connectivity issues. A programmable and reconfigurable analog Hopfield network is presented that addresses these challenges through a reconfigurable Manhattan architecture with a high-precision 14-bit floating-gate (FG) compute-in-memory (CiM) fabric. The network is implemented on a field programmable analog array (FPAA) and experimentally tested on three different NP-hard problems with different scaling challenges: Weighted Max-Cut (high connectivity and weight precision), traveling salesman problem (TSP) (high connectivity and medium weight precision), and Boolean Satisfiability/3SAT (low connectivity and weight precision) where it solved each problem optimally in microseconds.**

***Index Terms*— 3SAT, field programmable analog array (FPAA), Hopfield, Max-Cut, traveling salesman problem (TSP).**

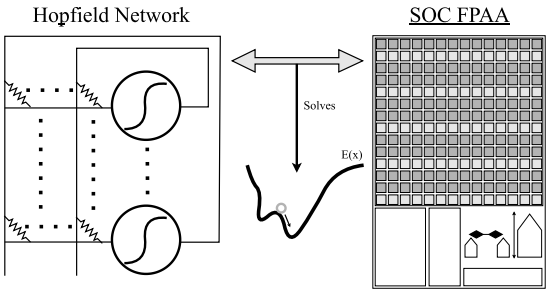


Fig. 1. Hopfield networks minimize quadratic energy surfaces. A Hopfield network made of analog circuits can be implemented on the SoC FPAA to create an analog system that can also minimize quadratic energy surfaces. If a problem is matched to an energy surface, the analog Hopfield network can give good or optimal solutions through its dynamics.

I. Efficient Optimization Networks

MANY difficult tasks reduce to one of Karp's 12 nonpolynomial (NP)-hard problems, such as route planning traveling salesman problem (TSP) and VLSI placement algorithms (Min-Cut) [1]. NP-hard problems can be verified quickly by digital computers, but finding the solution takes an NP amount of time. Digital solvers are therefore forced into inefficient heuristic or approximation algorithms to generate near-optimal solutions ([2], [3], [4]).

One common heuristic method used in optimizing neural network weights is energy minimization [5]. In this algorithm, the system's output is assigned an energy, with lower energy corresponding to better solutions. A numerical method such as gradient descent is then used to tweak system parameters and lower the energy, eventually reaching good solutions at an energy minimum. While digital energy minimization algorithms are effective, they can take many steps resulting in large time and energy costs.

Fortunately physical analog systems such as Hopfield and Ising networks have been shown to naturally perform energy minimization, solving the NP-hard quadratic unbounded minimization problem (QUBO) in the process ([6], [7], [8], [9], [10], [11], [12], [13]). These physical solvers can be built using analog circuits and are extremely energy efficient while still reaching near-optimal solutions. By definition, all NP-hard problems map to each other. Therefore, by solving QUBO, Hopfield/Ising networks can find solutions to all NP-hard problems [1].

However, scaling large combinatorial optimization problems to Hopfield/Ising networks comes with practical challenges in problem mapping and connectivity. NP-hard problems can scale up to $O(N^3)$ nodes when being mapped to Hopfield/Ising networks, and connections between nodes range from sparse to dense with varying levels of weight precision [1]. Fixed sparse architectures like the Chimera [14], Pegasus [15], and King's Graphs [16] allow for efficient sparse problems to be mapped but exacerbate inefficiencies when more dense connectivity is required [17]. Conversely, all-to-all topologies [18] come with many on-chip implementation issues and are difficult to practically realize. Compute-in-memory (CiM) techniques can be used for weighted connections, but previous implementations use limited precision 8-bit digital or memristive topologies ([7], [13]).

To solve these architectural and weight precision problems this work proposes using a Manhattan architecture and floating-gate (FG) devices to efficiently map both dense and sparse QUBO problems with high weight precision (Fig. 1). To prove the concept, this work presents the first experimental demonstration of QUBO on a Manhattan architecture by solving three different NP-hard problems with different scaling challenges: Weighted Max-Cut (dense, high weight precision), TSP (dense, medium weight precision), and Boolean Satisfiability or 3SAT (sparse, low weight precision) on a field programmable analog array (FPAA) [19]).

Received 21 May 2024; revised 17 August 2024 and 5 October 2024; accepted 10 October 2024. *(Corresponding author: Pranav O. Mathews.)*

The authors are with Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: prenziv@gmail.com; jennifer.hasler@ece.gatech.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2024.3480958.

Digital Object Identifier 10.1109/TVLSI.2024.3480958

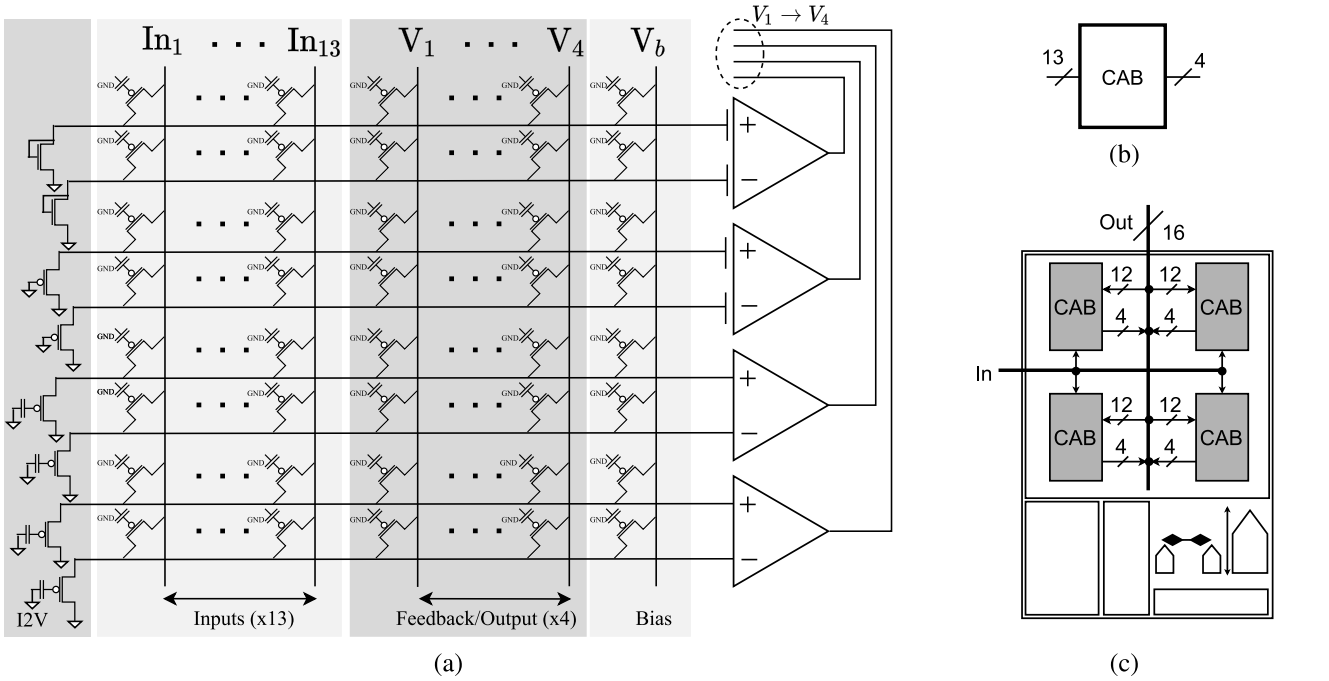


Fig. 2. (a) Circuit schematic of the analog Hopfield network. The inputs are weighted and summed using a VMM made of FG pFETs. A TA and I2V circuit takes the weighted current and maps it onto the TA sigmoidal nonlinearity. Both FG and normal TAs are used, and the I2V circuit uses FG pFETs, pFETs, and nFETs as shown. A bias element keeps an initial constant current in the row. (b) Block diagram of the Hopfield block is shown in (a). Seventeen lines are used in total. Thirteen take external inputs, and four hold the output states of the four neurons in one CAB. (c) Example 16 × 16 fully connected Hopfield network programmed onto the SoC FPAA by tiling together the base block shown in (b).

The article is organized as follows. First, the core programmable Hopfield network is examined (Section II), explaining circuit details and a novel mismatch calibration algorithm. The network is then shown solving three different NP-hard problems (Max-Cut, 3SAT, and TSP), converging to optimal solutions in microseconds (Section III). Network scaling is explored in Section IV followed by a final discussion in Section V.

II. Analog Hopfield Network

A Hopfield network is made of N interconnected neurons. The network dynamics minimize a quadratic energy function

$$H = -\frac{1}{2}\sum_{i}^{N}\sum_{j}^{N} W_{ij}x_i x_j - \sum_{k}^{N} I_k x_i \tag{1}$$

where W_{ij} describes the weighted connections between different neurons, x_i represents the output state of neuron i, and I_k is an external bias [20].

A neurons output, x_i, is described by a sigmoid-like function (S_i) that converges to

$$x_i = \begin{cases} 1, & \sum_j x_{ji} \geq T \\ 0, & \sum_j x_{ji} < T \end{cases} \tag{2}$$

which can be broken into two main elements: a signed, weighted summation and a sigmoidal thresholding function. These two elements can be replicated using analog circuitry and then interconnected to form an analog Hopfield network.

The analog implementation of the thresholding function and weighted summation will be discussed in Sections II-A and II-B, respectively. A technique to deal with device mismatch will then be shown (Section II-C), followed by a discussion on the Manhattan architecture (Section II-D).

A. Analog Sigmoidal Threshold

Differential, high-gain thresholding functions are commonly used in many analog systems as comparators. A simple implementation is to use a differential transconductance amplifier (TA) which has an output of

$$V_{\text{out}} = \approx \begin{cases} 0, & V_+ < V_- \\ V_{\text{dd}}, & V_+ > V_- \end{cases} \tag{3}$$

where V_+/V_- are the positive/negative terminal voltage inputs, respectively, and V_{dd} is the power supply (Fig. 2).

B. Signed and Weighted Summation

A weighted summation done over multiple rows is a vector-matrix-multiply (VMM) operation, common in CiM applications. The use of a differential thresholding function means that each neuron requires two VMM rows to create the three-quadrant multiplication necessary for a Hopfield network ($+\times+$, $+\times-$, and $-\times+$).

FG devices have been successfully used as highly programmable CiM structures, outperforming other nonvolatile memory elements [19], [21]. In subthreshold the current through a saturated FG pFET can be shown as

$$I = I_{\text{th}} e^{\kappa\left(\left(V_{\text{dd}}-V_{\text{fg}}-V_{T0}\right)-\left(V_{\text{dd}}-V_s\right)+\sigma\left(V_{\text{dd}}-V_d\right)\right)/U_T} \tag{4}$$

where I_{th} is the threshold current, V_{dd} is the power supply, V_{T0} is the threshold voltage, V_s is the source voltage, V_d is the drain voltage, U_T is the thermal voltage, κ is the gate coupling

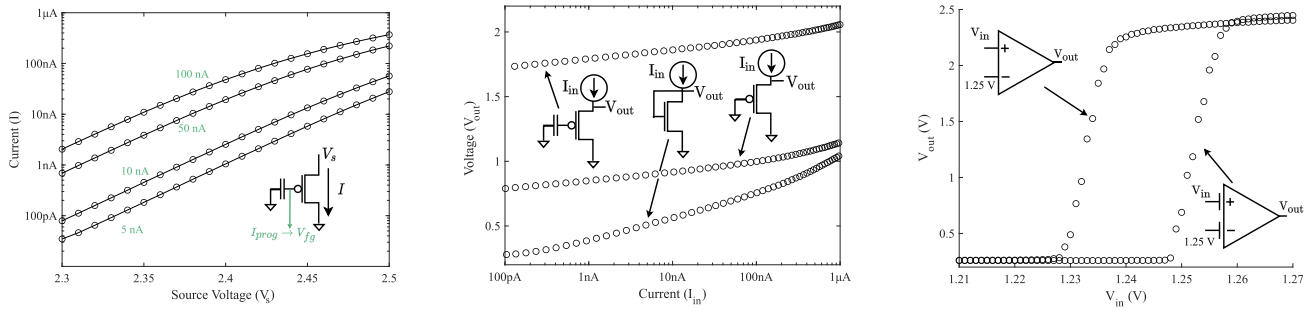


Fig. 3. (a) Source sweep of an FG pFET as used in the VMM. Different programmed charges weight the input voltage, pushing up the I–V curve. (b) Current in versus voltage out of the three different I to V converters used [Fig. 2(a)]. In the Hopfield network the current in is provided by a VMM representing the weighted sum of all the inputs to a neuron. (c) Sigmoidal response of a TA. Both normal and FG TAs are used in the Hopfield network; using the large gain option on the TA allows for a sharp response. Note that mismatch causes differences in where the transition occurs between TAs.

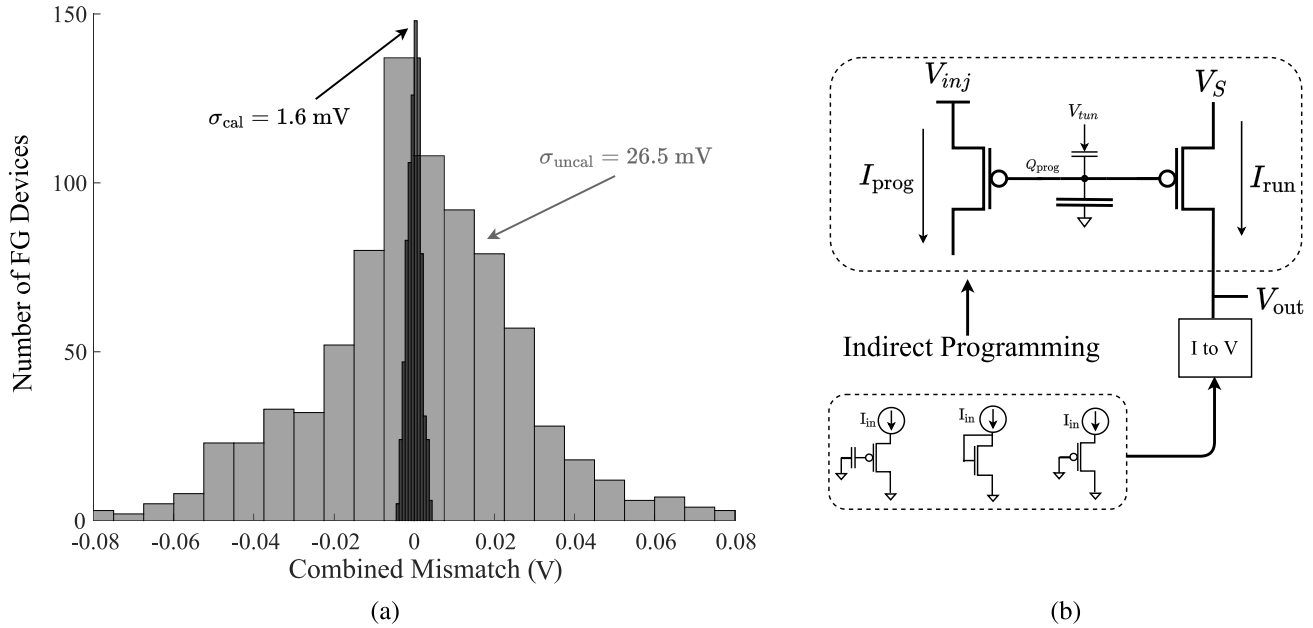


Fig. 4. (a) Plotted mismatch in the combined VMM + I2V circuit [Fig. 2(a)] before and after calibration. The FG devices used in the VMM are reconfigurable, allowing one to adjust the programming to account for device mismatch. Rather than calibrate each transistor individually, the whole circuit is considered. (b) Indirect programming scheme used in the SoC FPAA. Because the measured transistor during programming and the pFET used in the VMM are different devices, mismatch causes a difference in measured current I_{prog} and VMM current I_{run}.

factor, and σ is the drain coupling factor. V_{fg} accounts for the capacitive coupling of the control gate input and any charge on the FG node

$$V_{\text{fg}} = \frac{C_1}{C_T} V_g + \frac{C_{\text{ov}}}{C_T} V_d + V_Q. \tag{5}$$

In typical operation, the control gate, V_g, is held constant and applied through a large capacitance C_1. The overlap capacitance C_{ov} is much smaller than the total capacitance C_T, making the V_d term negligible. V_Q can be thought of as a weight on the output current of an FG pFET, and can be controlled through tunneling and injection mechanics, Assuming that $\sigma \approx 0$, (4) can be rewritten to clearly show this weighted current

$$I = W \times V_{\text{in}} \times I_0 = e^{-\kappa V_Q / U_T} \times e^{V_s / U_T} \times I_0 \tag{6}$$

where I_0 encapsulates the constant terms

$$I_0 = I_{\text{th}} e^{\kappa \left(\left(V_{\text{dd}} - \frac{C_1}{C_T} V_g - V_{T0} \right) - V_{\text{dd}} / U_T \right)}. \tag{7}$$

The relationship between the input (V_s) and output current is exponential, as shown in (4) and Fig. 3(a). Because I has an exponential dependence on V_s, the source voltages need to be near V_{dd} to cause a significant current output.

The VMM outputs a weighted, summed current. However, the thresholding TA takes voltage as an input. To fix this output/input mismatch, current from the VMM gets converted into a voltage through a grounded transistor. A diode-connected nFET, FG pFET, and normal pFET are all used, leading to different dc levels and I–V relationships [Fig. 3(b)]. The converted voltage for each row of the VMM goes to the corresponding terminal of a TA. Positive and negative rows are compared, and the voltage output of the TA is approximately like (3) for high gain or a large enough input voltage difference [Fig. 3(c)].

C. Mismatch in the VMM

If the same weight is programmed on a positive and negative row, they should effectively cancel after going through the current to voltage converter and TA threshold. Practically,

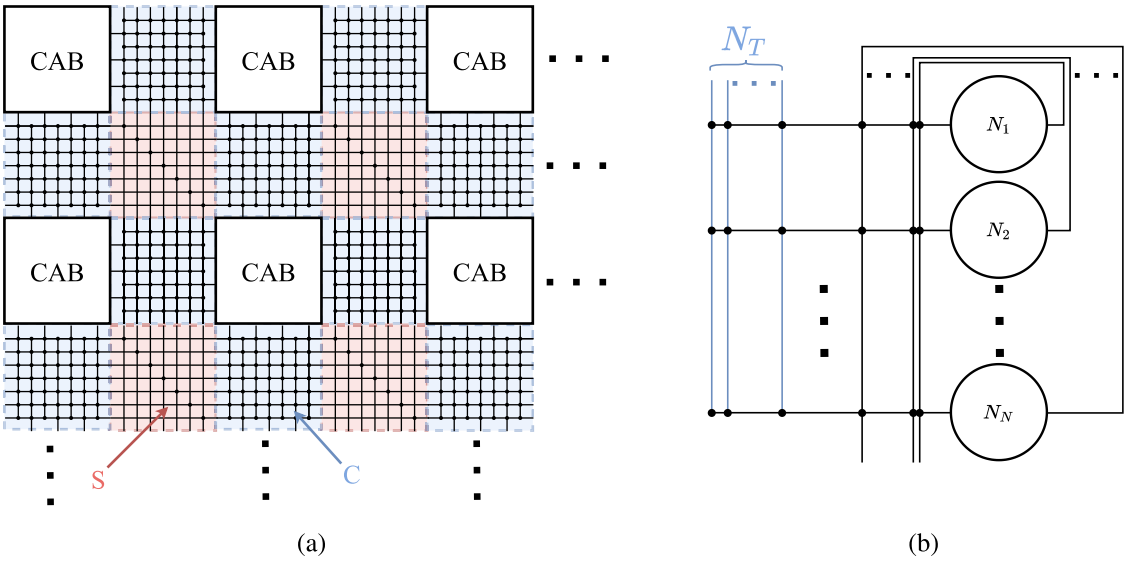


Fig. 5. (a) Manhattan architecture, composed of CABs, C, and S blocks. (b) CAB routing. Inputs come through the N_T from adjacent C blocks, and the CAB elements are connected in an all-to-all fashion. The outputs of each neuron go to C blocks to be routed between CABs.

a mismatch in the indirect programming scheme, current to voltage converter, and TA differential pair leads to offsets in the effective weight for each row. Fortunately, the programmability of FG devices allows us to correct mismatch if it is properly modeled.

For any given transistor, a mismatch can be modeled as a shift in V_{th}. By measuring this deviation and programming an offset ΔQ on the gate of a programmable device, a mismatch can be accounted for and greatly reduced. Previous work showed how a source follower-like circuit can be used to characterize and reduce mismatch [22]. Using each VMM pFET-specific IV transistor as the measuring device, the mismatch in both transistors can be corrected together.

V_{out} of a pFET + IV was measured for each input pFET and used to find its effective V_{th} shift and ΔQ necessary depending on the I–V used. For a pFET

$$\Delta Q \approx -\kappa \Delta V_{\text{out}} \tag{8}$$

and for a diode-connected nFET

$$\Delta Q \approx -\frac{\kappa_n}{\kappa_p} \Delta V_{\text{out}}. \tag{9}$$

Because of κ variation (8) and (9) don't yield the exact ΔQ needed to eliminate mismatch, but when repeated a very close value can be reached. An automatic calibration script was written, repeating this procedure for 816 FG pFETs and reducing the mismatch standard deviation from 26.5 to 1.6 mV (Fig. 4). The mismatch in this case is not threshold mismatch, but rather the deviation in targeted output voltage of the VMM + IV for a specific weighted current input and I–V transistor pair. Each neuron's positive and negative rows were matched because they share the same I–V converter, but different neurons were all calibrated to different targeted output voltages to match the specific devices used.

The TA input mismatch [Fig. 3(b)] can also affect circuit operation through incorrect comparisons, effectively leading to threshold mismatches across Hopfield neurons. This input mismatch did not massively affect circuit operation but may impact larger networks when scaled. However, FG TAs have programmable inputs so that the input mismatch can be programmed out manually in an iterative manner.

D. Hopfield Network in a Manhattan Architecture

Field-programmable gate arrays (FPGAs) and FPAAs use a Manhattan architecture to take advantage of routing requirements that are locally dense and globally sparse, such as in digital logic [23]. The Manhattan architecture consists of a fabric of computational analog blocks (CABs) or computational digital blocks (CLBs). Circuit elements can be routed together inside CABs/CLBs or between them with connection (C) and switch (S) blocks.

Hopfield networks also benefit from locally dense and globally sparse routing when mapping QUBO problems. Where for an FPAA/FPGA there would be various analog/digital circuit elements inside CABs/CLBs, one can replace them with the Hopfield sigmoidal thresholding function. The routing fabric can double as the VMM, a practice common in FPAA computational implementations [19].

The internal routing structure of a CAB leads to a simple embedding of a complete graph, the size of which is dependent on how many neurons are in the CAB. External routing lines from the S and C blocks increase the degree of each neuron, for a maximum all-to-all graph of V vertices found by

$$V = N_N + N_T \tag{10}$$

where N_N represents the number of neurons in a CAB and N_T is the number of input routing tracks (Fig. 5). In the system-on-chip (SoC) FPAA, $N_N = 4$ and $N_T = 12$. This result neglects mirror neurons, or Hopfield neurons that directly follow each other to effectively increase the degree of a neuron, which would increase this connectivity. If a larger all-to-all connectivity is desired, N_N or N_T can be increased.

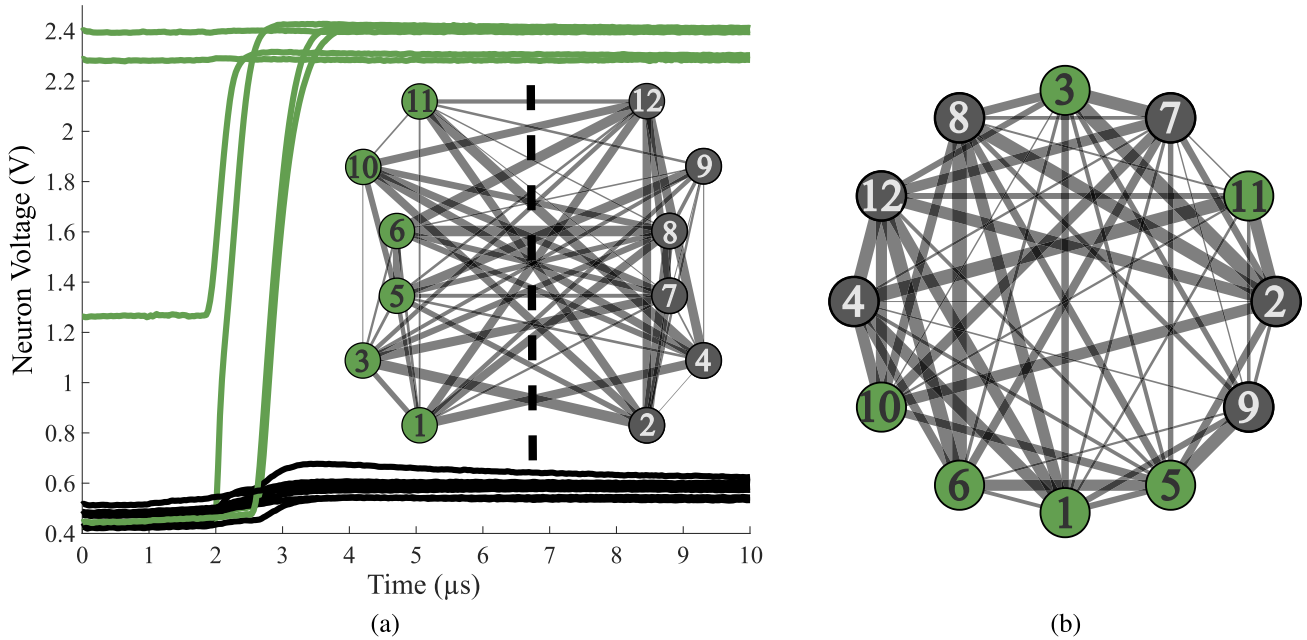


Fig. 6. (a) Analog Hopfield network converging on the optimal solution for a 12 vertex weighted max-cut problem. Weights were randomly generated from a uniform distribution between 5 and 100 with up to four decimal points. (b) Unpartitioned graph programmed onto the analog Hopfield network. The thickness of an edge represents the relative weight between the two vertices.

The reconfigurable S and C blocks increase routing area when compared to fixed sparse architectures such as Kings Graph, Pegasus, and Chimera. However, these configurations do not have the luxury of increasing their all-to-all connectivity like the Manhattan architecture. Instead, minor embeddings are found by taking advantage of their local complete graphs instead of through more heuristic methods. The NP-hard nature of mapping a problem to a sparse architecture leads to increased complexity in the practical usage of these architectures for denser problems [24].

III. Solving NP-Hard Problems Using the Analog Hopfield Network

Known equations map NP-hard problems to a Hopfield graph [1]. This graph is then embedded into the Manhattan architecture to be solved. An unstable initial condition is initially forced after which the network is allowed to converge at $t = 0$. The FPAA is programmed through custom tools ([25]) and the Hopfield Networks convergence is measured with an external oscilloscope.

Each subsection demonstrates a different NP-hard problem being experimentally solved on an analog Hopfield network programmed onto the SoC FPAA. The problems have different weight precision and connection density requirements. Section III-A will solve Max-Cut (high connectivity and weight precision), Section III-B will solve 3SAT (low connectivity and weight precision), and Section III-C will solve TSP (high connectivity and medium weight precision).

A. Graph Partitioning (Max-Cut)

The Max-Cut problem is a famous NP-hard graph partitioning problem [26]. Given a graph with N vertices and many edges between those vertices, what partition of the graph results in the maximum number of edges between the two separated sets? This problem can be additionally modified by adding weights to the edges and optimizing for the largest sum of weighted edges that can exist between the two partitions. Max-cut is also frequently size-constrained, where valid solutions have both partitions of equal size. An energy function for an Ising network with spins of 1 and -1 has been shown previously [1]

$$H = H_A + H_B = A\left(\sum_{i=1}^{N} s_i\right)^2 + B\sum_{(uv)\in E}\frac{1 - s_u s_v}{2} \tag{11}$$

where H_A constrains the partition split to be even, and H_B constrains the cut size. A positive B describes the min-cut problem by penalizing more cuts (raising the energy), while a negative B describes Max-Cut by encouraging more cuts (lowering the energy). Each vertex of the input graph maps to one neuron in the Hopfield network. Each vertex/neuron is partitioned depending on its output state (0/1).

Hopfield networks have outputs of [0, 1] rather than the $[-1, 1]$ of Ising networks. We can substitute using

$$s_i = 2x_i - 1 \tag{12}$$

where x_i is 0 or 1. Plugging into (11), simplifying, and manipulating into the Hopfield network form, (1), (ignoring partition size constraint, or $A = 0$) gives us

$$W_{ij} = \begin{cases} B * w_{ij}, & i, j \in E \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

$$I_k = -\frac{1}{2}\sum_j W_{kj} \tag{14}$$

where w_{ij} is the edge weight between vertices i and j.

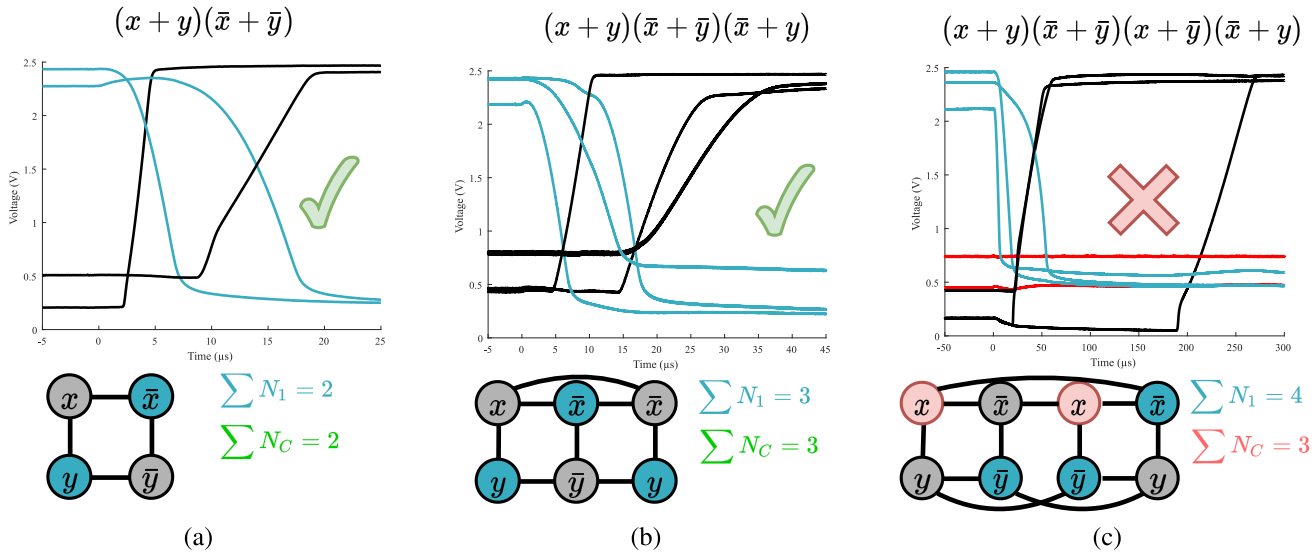


Fig. 7. 2SAT problems solved over multiple sizes on the analog Hopfield network. Blue/gray nodes in the MIS graph and Hopfield network convergence represent members of opposing sets, while the red nodes/lines show the two neurons that fail to evaluate in the unsatisfiable case. The equation shows the number of clauses (N_C) and the number of Hopfield neurons evaluating to "1" (N_1); if $N_C \leq N_1$ then the expression is satisfiable. (a) Two clause, four neuron problem showed to be satisfiable. An external input kicks the network between the two global minimum. (b) Three clause, six neuron problem shown to be satisfiable. An external input again kicks the network between two global minima; however, convergence takes a bit longer than the two-clause case. (c) Four clause, eight neuron problem shown to be unsatisfiable. The network is kicked between two minimums that each optimize for the number of clauses that can be solved (3/4).

A densely connected 12 vertex graph was generated and assigned random weighted edges (Fig. 6). Each weight was chosen randomly from a uniform distribution between 5 and 100 with up to four decimal places; $w_{\max}/w_{\min}$ was 14.977. These edge weights were then mapped to the Hopfield network using (13) and (14) with $B = -1$. The calculated weights and input biases were then mapped to the circuit by programming the given weight as a current on the VMM. The network was forced to an unstable state by perturbing the input vector and then allowed to converge, where it gave an optimal solution. This solution was verified by digitally calculating all possible cut sizes for an even partition.

Mismatch in the TAs resulted in output voltages below V_{dd} for large input differences. This affected circuit operation because of the exponential effect of the source voltage on current [Fig. 3(c)], leading to incorrectly weighted currents being reached at steady state. To combat this a second column of TAs was added to the circuit [Fig. 2(a)] to strengthen the sigmoid gain and give an output at V_{dd}. The second TAs had their negative terminal at $V_{\text{dd}}/2$ and its positive terminal took the core Hopfield TA output. Two TAs were used in one CAB, so the second row took up twice the number of CABs originally used by the analog Hopfield network ($N/2$).

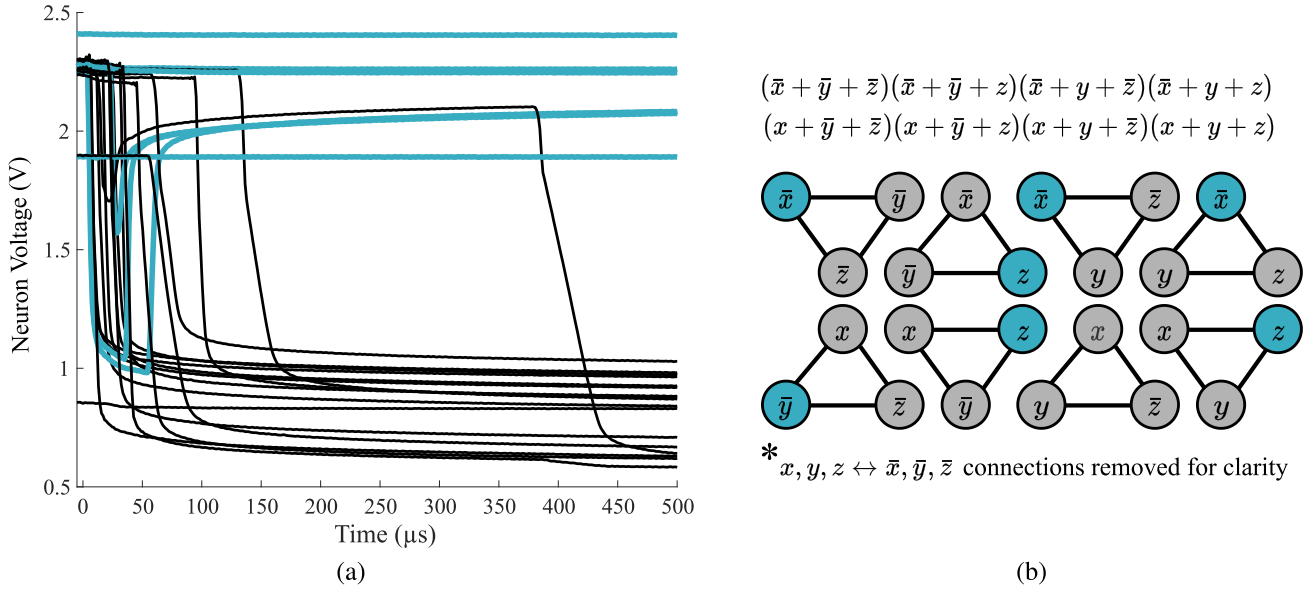
B. Boolean Satisfiability (3SAT)

The Boolean satisfiability problem, or SAT, is an NP-hard problem that asks whether a given Boolean expression can be True for some assignment of its input variables [26]. Any Boolean formula can be expressed in conjunctive normal form (CNF), which formats the equation as the logical and of many clauses, C, which are made up of Boolean variables, x, that are combined with logical or. Each x comes from some bank of Boolean variables which includes both a given variable and its complement. When the number of variables in each clause is three, this is called the 3SAT problem. Similarly, if the number of variables is two per clause, this is the 2SAT problem

$$F = C_1 C_2 \cdots = (x_1 + x_2 + x_3)(x_4 + x_5 + x_6) \cdots . \quad (15)$$

SAT can be further reduced to another NP-hard problem, maximal independent set (MIS) [27]. MIS is a special case of the set packing problem which asks for a set of the maximum number of vertices in a graph that do not share an edge. We can construct a graph with $N * m$ vertices for m clauses and N variables per clause. Each group of N represents a clause in the NSAT problem and edges are added between all members of the same clause. Edges are also added between every vertex that represents opposing Boolean variables (like x and $\bar{x}$) [1]. Only one variable in each clause can be a solution to the constructed MIS, and no variable and its inverse can be solutions. Solving MIS on this graph also gives the solution to SAT. The Hamiltonian of a MIS graph problem can be reformatted to fit a Hopfield network's energy function (1), where for the SAT problem W_{ij} is -1 for vertices that are connected by an edge and 0 otherwise [1]. This matrix ends up being sparse as connections between clauses are rare and the interclause connections are small in number.

In the context of SAT, if a Boolean expression is solvable there will be at least m neurons with a high output. Otherwise, the number of high neurons will represent the maximum number of clauses that can be satisfied. While 2SAT is not an NP-hard problem, the mechanisms a Hopfield network uses to solve it are identical to 3SAT. The analog Hopfield Network solved three 2SAT problems of size 4, 6, and 8 (Fig. 7). Two satisfiable Boolean expressions are correctly identified, and a nonsatisfiable one is shown to satisfy at most three out of the four clauses.



(a) (b)

Fig. 8. (a) Analog Hopfield network converging from an unstable forced state to a minimum that solves a 24 neuron 3SAT problem. The problem is unsatisfiable, so the network optimizes for the number of possible clauses to solve (7/8). (b) Eight clause, three variable 3SAT Boolean and the corresponding MIS graph that is programmed on a Hopfield network. Each connection in the graph corresponds to a programmed connection in the Hopfield network; however, connections between a variable and its inverse are removed for clarity. Blue nodes on the graph correspond to the Hopfield neurons that end as "1" while gray nodes to neurons that end as "0." The group of three neurons that never evaluate to "1" show that the expression is unsatisfiable.

There can be more than one solution to a SAT problem, as multiple combinations of variables can evaluate the given Boolean expression as true. The Hopfield network converges to the closest solution relative to its starting initial state, but strong external inputs can kick the network between the global minimum (Fig. 7). An external input kicked the network between two solutions that both satisfied the expression [Fig. 7(a) and (b)], and for the nonsatisfiable case [Fig. 7(c)], the Hopfield network shows two different solutions that each evaluate three out of the four clauses to be true.

Sparse problems like 3SAT allow for larger problems to be programmed on a reconfigurable fabric. It was previously shown [Fig. 2(c)] that the largest fully connected network possible on the FPAA was 16 × 16; however, by exploiting the sparsity of the 3SAT problem a 24 neuron 3SAT problem was programmed and solved (Fig. 8), where it converged to a correct solution from an unstable starting condition. The expression could not be satisfied so the network optimized and found that seven out of the eight clauses could be True.

C. Traveling Salesman Problem

The TSP is a famous NP-hard puzzle. If there are N cities, all interconnected with various distances between, what is the minimum distance path that takes you through all cities exactly once and returns you to the start? A solution to this problem is an ordered list of cities or a pairwise list of each city with its place in the tour. To represent these two pieces of information N^2 Hopfield neurons are used to represent an N city TSP problem in a table

	1	2	3
A	0	0	1
B	1	0	0
C	0	1	0

(16)

where each column represents a stop on the path, and each row represents a city [28]. Only one item should be high in each row and column as in (16) which shows the tour $B \rightarrow C \rightarrow A$ for a three-city problem.

The weight matrix for the energy function that maps this representation of the problem is made of four parts

$$\begin{aligned} W_{Xi,Yj} = &-A\delta_{XY}(1-\delta_{ij}) \\ &- B\delta_{ij}(1-\delta_{XY}) \\ &- C \\ &- Dd_{XY}(\delta_{j,i+1}+\delta_{j,i-1}) \end{aligned} \tag{17}$$

with an external bias of

$$I_{Xi} = C * N \tag{18}$$

where X, Y are index terms for the cities in a TSP problem (rows) and i, j are index terms for the tour numbers (columns). The δ function is 1 if its indices are the same number, 0 otherwise. The A term inhibits along a row, the B term inhibits among columns, the C term is a global inhibition, and the D term is what encodes the distances from a TSP problem into the function. In summary, the A, B, and C terms impose the structure of the representation shown in (16), and the D term encodes the TSP problem we want to solve [28].

A three- and four-city TSP problem, corresponding to a 9- and 16-neuron Hopfield network, was solved on the analog Hopfield network (Fig. 9). Similar to Max-Cut a second row of TAs was used to increase the sigmoid gain. Every tour is an optimal solution for the three-city problem, but the Hopfield network correctly imposes the structure of the TSP table shown in (16), returning a valid solution. The four-city problem also returns a valid solution, choosing the optimal path with a length of 1.2 m.

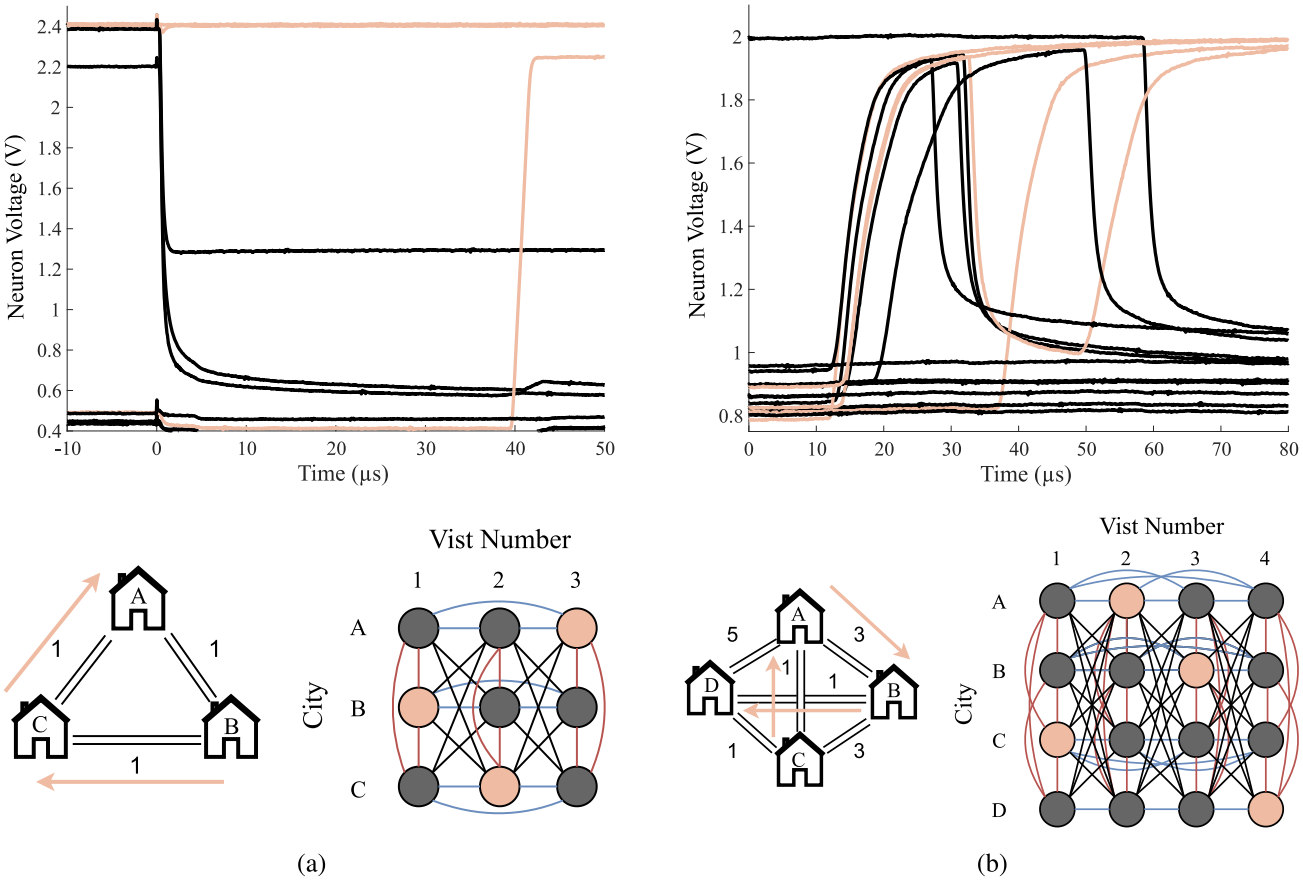


Fig. 9. Analog Hopfield network optimally solving (a) three-city and (b) four-city TSP problem. The top graphs show the network's convergence over time from a forced initial condition. Below the graph is a cartoon showing the TSP problem with the found optimal path and the Hopfield network that represents the problem next to it. The Hopfield network graph has red connections for column inhibitions, blue connections for row inhibitions, and black connections for distance inhibitory connections. Global inhibitory connections are removed for clarity.

IV. Manhattan Architecture Scaling

We have experimentally shown a Manhattan architecture (FPAA) that was used to implement an analog Hopfield network solving three NP-hard problems that have different scaling requirements. The architecture shows trade-offs in metrics such as power, speed, and area. In this section, these metrics will be discussed in the context of other hardware QUBO solvers and different NP-hard problem mappings.

Power usage in general scales with problem size. For the demonstrated Hopfield network, the sigmoidal thresholding function dominates, using excess power due to an inefficient topology and bias current. Temperature also affects power consumption; however, the highly reprogrammable nature of the bias currents allows for mitigation strategies. The network would benefit from colder temperatures, as the subthreshold swing and VMM efficiency both increase.

When compared to state-of-the-art solvers, our estimates show similar power per neuron numbers (Fig. 10) despite the high-power supply (2.5 V). Estimated experimental power is dominated by the neuron (two TAs with a 10 μA bias) because it drives the low impedance source-coupled VMM. A theoretical switch to a high impedance gate-coupled VMM allows a lower TA bias (one TA at 0.5 μA), and also enables direct tradeoffs of power and speed. Currents in the FG VMM are generally low, contributing small amounts to the overall system power usage. The choice of QUBO solver may also enable low-power operation; it has been shown that Hopfield networks may offer inherent energy savings over the oscillatory Ising topologies that most solvers use [29].

Comparisons also show the precision advantage of using FG devices (Fig. 10). Other digital or analog implementations top out at around 8 bits, while FGs have been shown to reach 14-bit accuracies [30]. FGs can also double as pass gates, allowing them to be used in routing infrastructure and not costing extra area for just the VMM. The extra precision offered by FGs allows for more complex, practical problems to be solved such as the weighted Max-Cut and TSP.

The Manhattan architecture has all-to-all connectivity that scales with CAB size (10). In the experimental measurements of dense problems such as TSP and Max-Cut, the FPAA CAB size was able to match the minimum neuron size. Fixed sparse architectures, like Chimera and Kings Graph, suffer massive overhead when mapping dense problems (Fig. 11).

Sparse problems also gain efficiency on the Manhattan routing structure, specifically locally dense and globally sparse mappings. One example is the 3SAT problem, where the highest connectivity is among clauses and inverses. Chimera and Kings graphs architectures would require more than the minimum nodes to solve the 24-node 3SAT problem (Fig. 11). The Manhattan routing structure has been shown to scale up for 3SAT; routing has been demonstrated for 1000 node problems from standard datasets [17].

	[18]	[13]	[7]	[11]	[16]	[14]	This Work
Power (mW)	629	.00114	10.9	2.56	52	25×10^6	1.05/.056*
Size	512	1024	60	55	1968	> 5000	12
Power/Neuron	1.23	0.001	0.182	0.047	0.021	50×10^6	0.087/0.0046*
Process	65nm	65nm	16nm	28nm	65nm	Superconductor	350nm
Precision	5 bits	8 bits	1 bit	1 bit	< 3 bits	4-5 bits	14-bit
Architecture	All-to-All	Lattice	All-to-All	All-to-All	Kings Graph	Chimera,Pegasus	Manhattan
Type	Digital	Digital	Hybrid	Analog	Analog	Quantum	Analog
Problem	Max-Cut	Max-Cut	Max-Cut	COP	3SAT	Analog	Max-Cut

*Experimental/Theoretical estimates

Fig. 10. Representative metrics for state-of-the-art QUBO solver compared against this work. Experimental power is estimated based on the bias currents and weights used in Section III, theoretical numbers assume minimum bias currents for a gate-coupled VMM structure operating at the same speed. The FGs provide superior weight programmability to other implementations, and power is comparable to other implementations.

	Number of Neurons			
Problem	All-to-All	Manhattan	King	Chimera
Max-Cut	12	12	169	72
TSP	16	16	256	128
3SAT	24	24	> 24	> 24

Fig. 11. Neuron size for the Hopfield network in this work versus King and Chimera architectures. The sparse topologies take massive hits for complete graphs, while still taking more neurons than the Manhattan architecture for a sparse problem ([31], [32]).

Routing requirements differ for sparse and dense problems, leading to performance differences. The flexibility of the Manhattan routing structure comes from pass gates in the S and C blocks; higher routing density means that signals go through more gates. This increases the effective resistance and capacitance on the line, leading to higher driving requirements which can lower speed or necessitate power increases. This can be shown in the 3SAT convergence time, which increases massively for larger connection densities (Fig. 12). In contrast, the denser TSP and Max-Cut problems converge faster because of the second TA used to increase the neuron's drive strength.

V. Conclusion

This work presents the first experimental demonstration of a Manhattan architecture for QUBO problems and the largest demonstration of FG devices for QUBO connections. It is shown that the Manhattan architecture can efficiently map dense problems while still retaining some efficiency for sparse mappings, a property current fixed sparse architectures do not have. When compared against other solvers this work has similar numbers despite being built in a much older process node. The precision of the FG devices allows for solutions to problems not feasible or possible on other systems, such as the weighted Max-Cut and TSP (Section III).

Mapping QUBO problems to sparse architectures such as Chimera and Pegasus is NP-hard [33], and there are many algorithms that work to find the most efficient mapping. Similarly, mappings of sparse problems to the Manhattan architecture should be investigated. Standard place and route algorithms have been shown to be effective [17]; however, more efficient embeddings may appear. In addition, the concept of "mirror neurons" should be explored in the analog circuitry to enable large sparse mappings.

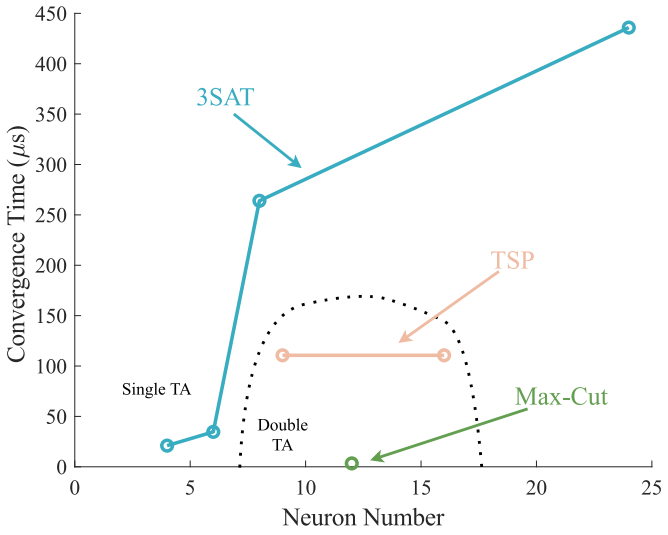


Fig. 12. Convergence time of different Hopfield network problems versus problem size. Problems with a second row of buffers converge faster, and in general larger problems converge slower.

A large custom ASIC that uses the Manhattan architecture, FG, and analog circuit concepts from this article would clearly demonstrate the scaling potential of this work. Efficiency optimizations in the threshold function, VMM, and routing can be co-designed to maximize the potential of a reconfigurable Hopfield network. Combining a large-scale Manhattan architecture QUBO solver with research into efficient mappings would contribute greatly to the research into enabling efficient solutions to NP-hard problems.

References

[1] A. Lucas, "Ising formulations of many NP problems," *Frontiers Phys.*, vol. 2, p. 2, Feb. 2014.
[2] K. M. Zick, "Improved sparse Ising optimization," 2023, *arXiv:2311.09275*.
[3] V. Paschos, "An overview on polynomial approximation of NP-hard problems," *Yugoslav J. Oper. Res.*, vol. 19, no. 1, pp. 3–40, 2009.
[4] R. Howell, "Simulated annealing on NP-complete problems," in *Proc. ACMS Conf.*, Jun. 1989, pp. 104–123.
[5] Y. Lecun, S. Chopra, R. Hadsell, M. A. Ranzato, and F. J. Huang, "A tutorial on energy-based learning," in *Predicting Structured Data*, G. Bakir, T. Hofman, B. Scholkopt, A. Smola, and B. Taskar, Eds., Cambridge, MA, USA: MIT Press, 2006.
[6] J. J. Hopfield and D. W. Tank, "Computing with neural circuits: A model," *Science*, vol. 233, no. 4764, pp. 625–633, Aug. 1986.

[7] F. Cai et al., "Power-efficient combinatorial optimization using intrinsic noise in memristor Hopfield neural networks," *Nature Electron.*, vol. 3, no. 7, pp. 409–418, Jul. 2020.

[8] T. Wang and J. Roychowdhury, "OIM: Oscillator-based Ising machines for solving combinatorial optimisation problems," in *Unconventional Computation and Natural Computation*. Cham, Switzerland: Springer, 2019.

[9] P. O. Mathews et al., "High-speed CMOS-free purely spintronic asynchronous recurrent neural network," *APL Mach. Learn.*, vol. 1, no. 1, Mar. 2023, Art. no. 06107.

[10] P. O. Mathews and J. O. Hasler, "Physical computing for Hopfield networks on a reconfigurable analog IC," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2023, pp. 1–5.

[11] S. Dutta et al., "An Ising Hamiltonian solver based on coupled stochastic phase-transition nano-oscillators," *Nature Electron.*, vol. 4, no. 7, pp. 502–512, Jul. 2021.

[12] A. Lu et al., "Scalable in-memory clustered annealer with temporal noise of charge trap transistor for large scale travelling salesman problems," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 13, no. 1, pp. 422–435, Mar. 2023.

[13] Y. Su, T. T.-H. Kim, and B. Kim, "FlexSpin: A CMOS ising machine with 256 flexible spin processing elements with 8-b coefficients for solving combinatorial optimization problems," *IEEE J. Solid-State Circuits*, vol. 59, no. 8, pp. 2659–2670, Aug. 2024.

[14] K. Boothby, P. Bunyk, J. Raymond, and A. Roy, "Next-generation topology of D-wave quantum processors," 2020, *arXiv:2003.00133*.

[15] I. Ahmed, P.-W. Chiu, W. Moy, and C. H. Kim, "A probabilistic compute fabric based on coupled ring oscillators for solving combinatorial optimization problems," *IEEE J. Solid-State Circuits*, vol. 56, no. 9, pp. 2870–2880, Sep. 2021.

[16] W. Moy, I. Ahmed, P.-W. Chiu, J. Moy, S. S. Sapatnekar, and C. H. Kim, "A 1,968-node coupled ring oscillator circuit for combinatorial optimization problem solving," *Nature Electron.*, vol. 5, no. 5, pp. 310–317, May 2022.

[17] T. Jagielski, R. Manohar, and J. Roychowdhury, "FPIM: Field-programmable Ising machines for solving SAT," 2023, *arXiv:2306.01569*.

[18] K. Yamamoto et al., "STATICA: A 512-spin 0.25 M-weight annealing processor with an All-Spin-Updates-at-Once architecture for combinatorial optimization with complete Spin–Spin interactions," *IEEE J. Solid-State Circuits*, vol. 56, no. 1, pp. 165–178, Jan. 2021.

[19] S. George et al., "A programmable and configurable mixed-mode FPAA SoC," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 6, pp. 2253–2261, Jun. 2016.

[20] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci. USA*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982.

[21] C. Brando, M. Park, S. N. Chowdhury, M. Chen, K. Lee, and S. Shah, "Modeling and analysis of analog non-volatile devices for compute-in-memory applications," in *Proc. IEEE 66th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, vol. 4, Aug. 2023, pp. 103–107.

[22] S. Kim, S. Shah, and J. Hasler, "Calibration of floating-gate SoC FPAA system," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 9, pp. 2649–2657, Sep. 2017.

[23] S. M. S. Trimberger, "Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology: This paper reflects on how Moore's law has driven the design of FPGAs through three epochs: The age of invention, the age of expansion, and the age of accumulation," *IEEE Solid StateCircuits Mag.*, vol. 10, no. 2, pp. 16–29, Spring. 2018.

[24] E. Lobe, L. Schürmann, and T. Stollenwerk, "Embedding of complete graphs in broken chimera graphs," *Quantum Inf. Process.*, vol. 20, no. 7, Jul. 2021, Art. no. 234.

[25] S. Kim, S. Shah, R. Wunderlich, and J. Hasler, "CAD synthesis tools for floating-gate SoC FPAAs," *Design Autom. Embedded Syst.*, vol. 25, no. 3, pp. 161–176, Sep. 2021.

[26] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations*, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, Eds., Boston, MA, USA: Springer, 1972, pp. 85–103.

[27] V. Choi, "Adiabatic quantum algorithms for the NP-complete maximum-weight independent set, exact cover and 3SAT problems," Apr. 2010, *arXiv:1004.2226*.

[28] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons.," *Proc. Nat. Acad. Sci. USA*, vol. 81, no. 10, pp. 3088–3092, May 1984.

[29] P. O. Mathews and J. O. Hasler, "Hopfield vs Ising: A comparison on the SoC FPAA," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 71, no. 9, pp. 3999–4008, Sep. 2024.

[30] S. Kim, J. Hasler, and S. George, "Integrated floating-gate programming environment for system-level ICs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 6, pp. 2244–2252, Jun. 2016.

[31] Y. Sugie et al., "Minor-embedding heuristics for large-scale annealing processors with sparse hardware graphs of up to 102,400 nodes," *Soft Comput.*, vol. 25, no. 3, pp. 1731–1749, Feb. 2021.

[32] *QPU Solvers: Minor-Embedding-D-Wave System Documentation Documentation*. Accessed: May 10, 2024. [Online]. Available: https://docs.dwavesys.com/docs/latest/handbook_embedding.html#reusing-embeddings

[33] E. Lobe and A. Lutz, "Minor embedding in broken chimera and derived graphs is NP-complete," *Theor. Comput. Sci.*, vol. 989, Mar. 2024, Art. no. 114369.