THE FRONT PAGE FOR PROBABILISTIC SPIN LOGIC

by

Lakshmi Anirudh Ghantasala

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Electrical and Computer Engineering

School of Electrical & Computer Engineering

West Lafayette, Indiana

December 2019

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Supriyo Datta, Chair

Department of Electrical and Computer Engineering

Dr. Zhihong Chen

Department of Electrical and Computer Engineering

Dr. Vijay Raghunathan

Department of Electrical and Computer Engineering

Approved by:

Dr. Dimitrios Peroulis

Head of the Graduate Program

This thesis is dedicated to my parents Muralidhar and Sridevi Ghantasala

ACKNOWLEDGMENTS

I would like to first and foremost thank CAPSL for sponsoring my Master's education. I would like to specially thank Prof. Supriyo Datta for providing me invaluable guidance as I complete my Master's requirements. I would like to specially acknowledge Zeeshan Pervaiz and Brian Sutton for allowing me to contribute to their papers. I would also like to thank Kerem Camsari for valuable discussions and insights he's given me throughout my time in this group.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Author: Ghantasala, Lakshmi, A. MSECE
Institution: Purdue University
Degree Received: December 2019
Title: Purduep.com, The Front Page of Probabilistic Spin Logic Research
Committee Chair: Supriyo Datta

While probabilistic neural networks are a staple of the neural network field, their study in the context of real hardware has been limited. Probabilistic spin logic entails the study of probabilistic neurons that have real hardware counterparts. This comes under a new effort, termed Purdue-P, whose goal it is to develop efficient, probabilistic neural network hardware to solve some of today's most difficult problems. An important step in this effort has been the development of a website, purduep.com, to act as a "front page" for the effort. This website introduces the idea of probabilistic spin logic to newcomers, houses an online web simulator and blog, and provides instructions on how to access a powerful asynchronous p-computing co-processor through the cloud. The thoughts behind the flow of content, the web simulator, and cloud access of the co-processor constitute the crux of the thesis.

1. INTRODUCTION

A website has been developed that describes a novel computing paradigm termed Probabilistic Spin Logic (PSL). This site, "purdue.edu/p-bit", will be the topic of this thesis. While I have co-authored two publications over the course of my Masters in addition to the work presented here, those are part of other students' theses, and will not be discussed in any significant part here. However, those contributions have given me the necessary background to build this website. As this website will be the front page for Purdue's probabilistic computing effort for the foreseeable future, It warrants some discussion as to its design, implementation, and content. The homepage for "purdue.edu/p-bit" accounts for the brunt of the website. For readability, the homepage has been broken into multiple sections, each with their own purposes. Users will scroll through these sections in order as they proceed down the website, necessitating proper ideological transitions between sections. These transitions should carry over to the chapters of this thesis. Chapter 2 of this thesis will cover the first two sections which introduce the idea of PSL, while chapters 3 and 4 of the thesis will cover section 3 on the webpage. Section 3, "Tools", contains the Purdue-P web simulator and instructions to connect to the Purdue-P Co-processor, which each will have their own chapters.

The purpose of this website is to introduce the idea of PSL to newcomers, while providing a hub for PSL related information that returning users can also find useful. Design choices have been made to satisfy both ends of this spectrum. Firstly, a one-page design gives newcomers a controlled, curated experience with only one way to go forward: scrolling down the webpage. This contrasts an earlier multi-page structure in which sections had their own pages. Different users may have different experiences under this model as they explore this network of pages on their own. An animated "scroll down" indicator has been added to emphasize this idea. Information has

been organized to go from most generic, to most specific, from up to down. The **Overview** section, therefore, contains the most generic information, describing PSL in English with simplified diagrams to emphasize key points. The **Applications** section follows, which describes how PSL can be implemented to solve real problems with eye-catching images. These first two sections are written such that the first describes the "what" and the second section describes the "why" of PSL, providing a natural segue for readers.

At this junction, we have introduced the idea of PSL in a largely one-way fashion. The most casual sector of the userbase may find this information substantial enough. Now the experience becomes a two-way exchange with users being able to interact with tools that we have developed in the **Tools** section. These tools assume no prior knowledge of the user apart from the two sections presented previously on the homepage. For those that would like to then understand the science behind why these tools works, a **Resources** section follows with links to additional PSL information. Videos, links to slides, and even a blog are arranged in this section. The blog will become an extension of the website, containing useful snippets and up to date specifics related to PSL research that the homepage will not be able to cover. Finally, a **Research** section follows. It not only provides specific information to a more neural network familiar audience, but it lends credibility to the site and the tools being presented. The research section will be updated at intervals to highlight the most important papers being put out by the Purdue probabilistic spin logic research group.

Advanced users can quickly access the more nuanced aspects of the site with a quick links bar on the starting screen, which currently has links to the co-processor instructions, the web simulator, and the blog. This structure allows newcomers to "get off the train" at any stage of their one-page journey, while more advanced users can scroll all the way through resources and research to find

the most rigorous information regarding PSL. The remainder of this thesis will explore the various sections of the homepage in order, and hopefully take the reader on a similar, albeit more informative, experience as those who would come to “purduep.weebly.com”.

2. PROBABILISTIC SPIN LOGIC

2.1 Website Section 1 - Overview

While classical bits deterministically occupy a 1 or a 0, and quantum bits occupy some superposition of 1 and 0, a novel paradigm called probabilistic bits occupy a 1 or a 0 with some probability. These probabilistic bits, or p-bits, are at their essence random number generators that can be biased towards 1 or 0. Like transistors for classical bits, p-bits can be interconnected to form useful circuits that solve a variety of problems. These p-circuits are defined by two simple equations that will be referred to as eq(1) and eq(2) [1][2] . Eq(1) defines the update of a single p-bit given some biasing variable I_i, given by:

$$m_i = \text{sign}(\tanh(I_i) - \text{rand}(-1,1)) \tag{1}$$

Rand (-1,1) is a uniform random variable between 1 and -1, the range of the tanh activation function. Eq(2) defines the generation of the biasing variable. This general weighted summing operation is given by:

$$I_i = I_0\left(h_i + \sum_{j=1}^{N} J_{ij} m_j\right) \tag{2}$$

Probabilistic spin logic (PSL) is a paradigm built on these two equations, encompassing the study of this specific form of probabilistic neural network and how it can solve today's problems. While the idea of a probabilistic neural network is not new [3], studying these networks in the context of real hardware is; PSL shines in building a system which can be implemented with real transistor and/or magnetic devices that will compose an efficient hardware architecture [4]. These

two equations govern a very flexible system to which a wide variety of problems can be mapped. P-circuits have been shown to solve combinatorial optimization problems including TSP, Max-Cut, and Subset Sum, Bayesian inference problems, and Boolean logic problems [1][5].

2.2 Website Section 2 - Applications

Probabilistic spin logic encapsulates a very flexible system suitable for a range of real-world applications. Combinatorial optimization, Bayesian inference, and invertible Boolean logic are three such applications suitable for p-circuits that are emphasized on “purduep.weebly.com”. While there are additional applications like sampling and qubit emulation [6], the next two sub-sections will explore combinatorial optimization and invertible Boolean logic problems specifically. These sections will explain related examples that I have helped solve in published works over the past two years. These examples provide additional evidence that these families of problems map handily to PSL.

2.2.1 Combinatorial Optimization

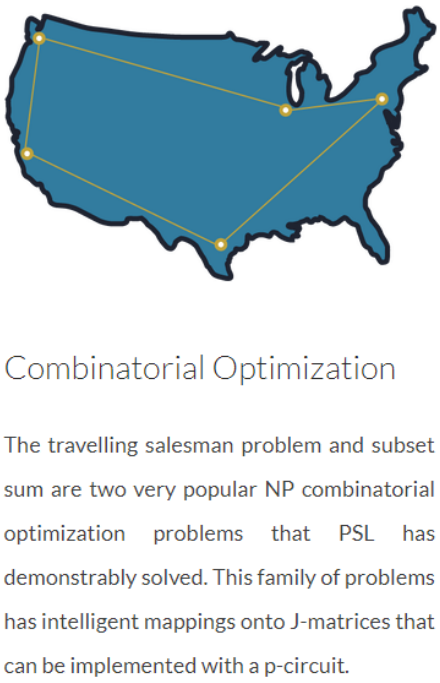

Figure 2-1 Combinatorial Optimization web-snippet

While the webpage puts forth a short snippet, shown in Figure 2-1, that describes PSL's role in solving combinatorial optimization problems, much work has gone on behind the scenes to establish optimization as a family of problems applicable to PSL. A recent FPGA demonstration by Brian Sutton and myself falls under combinatorial optimization. Figure 2-2 captures this demonstration of a 1000 p-bit, anti-ferromagnetically coupled network cooling to its ground state. This problem can be classified as a Max-Cut problem, a popular archetype within the NP combinatorial optimization class, for which the maximum cut is obtained by separating every alternate p-bit into a single group. This splits the network into two groups, each containing alternating p-bits, which results in the checkerboard pattern in Figure 2-2.

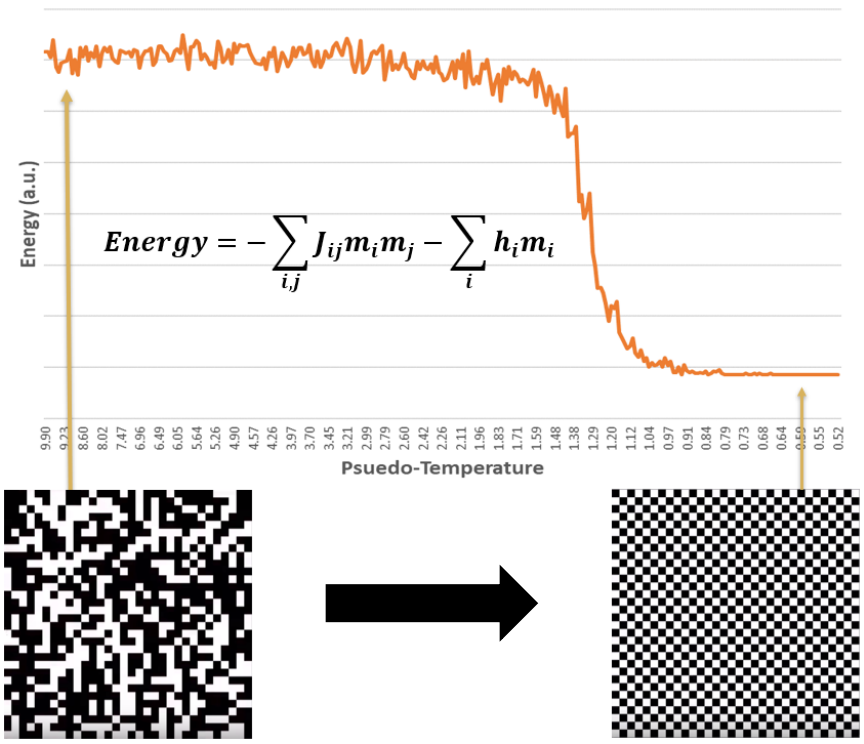

Figure 2-2 Convergence of 1000 p-bit anti-ferromagnetically coupled p-circuit

A 400 p-bit version of this problem is available as a pre-loaded example on the web-simulator and will be explored in more detail in chapter 3. Brian Sutton also demonstrated solving the travelling salesman problem using PSL in 2016 [7]. These examples evidence the claim that PSL is well suited to solve combinatorial optimization problems.

2.2.2 Invertible Boolean Logic

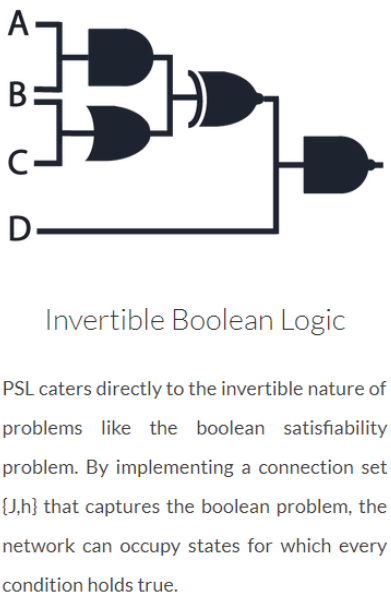

Figure 2-3 Invertible Boolean Logic web-snippet

PSL is invertible by nature, meaning p-circuits will naturally fluctuate inputs to match outputs that are pinned to certain values (Figure 2-3). Zeeshan et. Al. [5] showed in 2017 the subset-sum problem being solved on an FPGA, extracting the correct solution from a histogram the FPGA produced, shown in part c of Figure 2-4. Subset sum is an NP-Complete problem in which one must find a subset of elements within some set that add up to a value *k*. Though subset-sum is a combinatorial optimization problem, the problem was formatted as a series of Boolean logic gates, specifically Full/half–adders. This is one of the examples solidifying invertible Boolean logic as an application of PSL, and so will be explained in more detail.

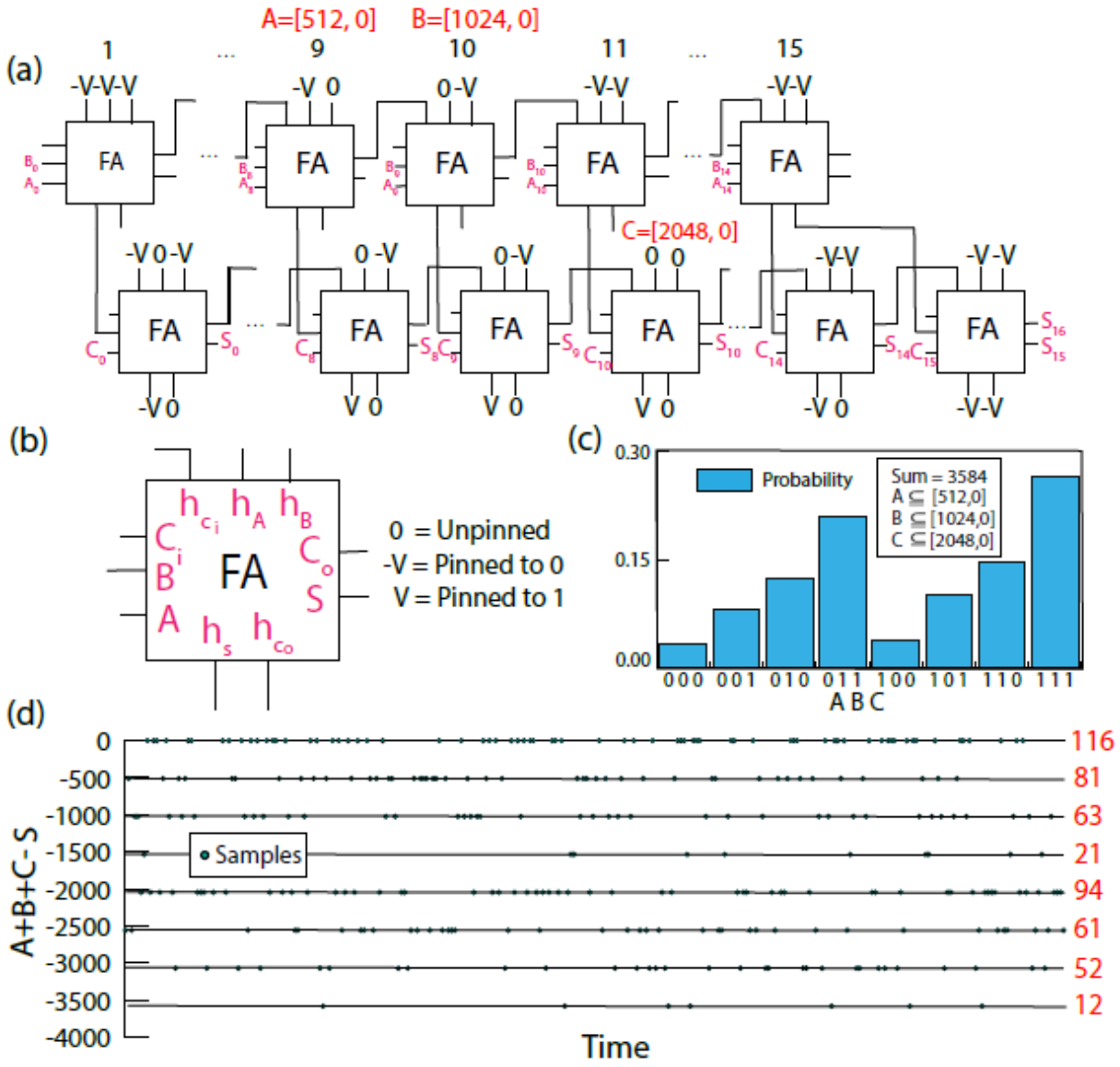

Figure 2-4 Subset-Sum figure

Part A of Figure 2-4 describes the architecture of the instance that was implemented. In order to sum two numbers, a row of full adders is required. For each subsequent number that is being summed, an additional row must be added where one input is the sum from the previous row and the other is the new number. For this instance, it was pre-decided that the subset would consist of a maximum of 3 numbers, and so two rows of full adders were implemented. Each full adder has 5 pins: two inputs (a and b), a carry-in, carry-out and sum. The carry out of each MSB is

connected to the carry in of the next significant MSB. The connection scheme is implemented in hardware via a J matrix that acts as an adjacency matrix for the entire subset sum graph, connecting different ports in each full adder together.

The output pins of the second row of Full Adders was clamped to 3854. Every bit of the inputs was clamped except for 3 bits, one bit from each input. The value taken by these three bits would decide which out of 4 possible numbers {0, 512, 1024, 2048} would sum to 3854. The solution to the problem is {111}, as 512 + 1024 + 2048 = 3854. The histogram of system state occupancy is shown in part c of Figure 2-4. It should be observed that the state of maximum occupancy is {111}, the correct answer. This shows that the network backtracked such that the inputs of the problem fluctuated to the correct values that summed to the pinned output. This is but one of the examples showcasing invertible Boolean logic with PSL.

3. PURDUE-P WEB SIMULATOR

3.1 Web Simulator Overview

The Web Simulator is an online platform built into the Purdue-P website that allows users to execute probabilistic spin logic on any moderately sized p-circuit. The simulator runs the same Parallel-PSL algorithms as those in the FPGA co-processor, albeit on a much slower timescale. The simulator has three options, each with their own flexible sub-options. One, the user can execute PSL on a randomly generated J-matrix and H-vector. Two, the user can execute PSL on their own specified parameter set. Three, the user can execute PSL on one of three pre-loaded example parameter sets. These parameter sets will be further detailed in Section 3-4.

The purpose of this web simulator is multi-fold. First, the simulator will connect the real world to the algorithms that we have designed. That is, the simulator should properly illuminate the real-world → PSL → real-world transition. The user should start in the real world, with some problem, and end in the real world with some solution.

Second, the simulator should give users a simple environment to execute PSL in, removing the need for extensive set up procedures that are otherwise required to run the algorithms in Python or MATLAB. The resultant streamlined experience removes the need to build read-out functions, install packages, and debug the resulting code.

Third, the web simulator should motivate users to take the additional steps needed to use the Purdue-P co-processor. The co-processor is the real innovation of the Purdue-P team, with significant thought put into its hardware parallel-execution of PSL. However, this co-processor requires multiple technical steps on the users' part to set up. With an easy to use web simulator, the user should be more inclined to take those steps and delve further into PSL research.

As a bonus, the web simulator also puts PSL execution on the smart phone for the first time. Networks as large as 400 p-bits have been run on a smartphone in a few seconds. Now, all you need is a J-matrix or H-vector ready, and you can run PSL anywhere!

3.2 Parallel PSL Algorithms

Eq.(1,(2) are sequential by nature; these equations are applied to p-circuits one p-bit at a time until every p-bit in the network has been updated, as is generally required for unrestricted boltzmann machines [8]. This scheme limits the rate at which networks of p-bits can be updated. A novel update scheme has been proposed that allows multiple p-bits to update per timestep, resulting in a faster convergence rate for many problems, while maintaining the accuracy of the predicted ground state. This new scheme describes a Parallel-Probabilistic Spin Logic (PPSL) framework. The original, sequential equations now describe a Classical-Probabilistic Spin Logic (CPSL) framework, for distinction. The new framework is better suited to model real hardware, where magnets lack a sequencer to tell them when to flip. This parallel scheme is what drives the Purdue-P Web Simulator.

3.3 Simulator Inputs

The simulator itself has two modes, one shows the text field setup for a specific J-matrix/H-vector problem (referred to as specify text field setup), and one for the random setup (referred to as random text field setup). These two setups are shown in Figure 3-1.

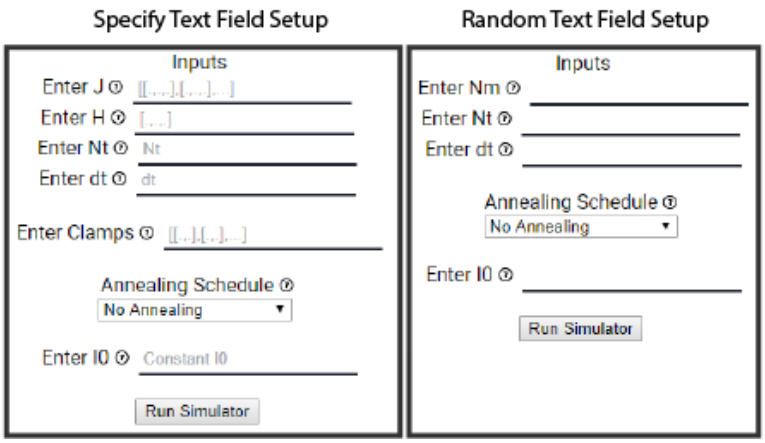

Figure 3-1 Specify and Random text field setups

The pre-loaded examples simply fill the specify text field setup with the parameters that reflect that problem. The specify text-field setup offers a variety of knobs that the user can tune.

- J-matrix - The connection matrix describing the weights between p-bits.
- H-vector - The bias vector describing initial state of p-bits.
- N_t - Number of time-steps to collect data.
- dt - Mean update frequency for p-bits. A small dt leads to more simultaneous updates, while a larger dt leads to fewer.
- Clamps – specifies which p-bits are clamped, if any, formatted as [[p-bit #, clamp value],[.,.],...]. Clamp values must be +-1.
- Annealing setup
    - No annealing - A constant scaling factor for the J-matrix. I_0 corresponds to inverse-temperature.
    - Linear annealing - I_0 is modified each timestep following $I_0 = I_0 + I_{end}/N_t$. This is equivalent to linearly reducing temperature to encourage the system to converge to the global minimum.

- Geometric annealing - I_0 is modified each timestep following $I_0(t+1)$ = growth_factor * $I_0(t)$. This exponentially reduces temperature to encourage the system to converge to the global minimum.

The random text field set up removes the option to input a J-matrix, H-vector, and clamps, while introducing a new parameter.

- N_m - Number of magnets in the p-circuit.

This parameter was previously inferred from the size of the H-vector, when the H-vector was an input, since the H-vector must provide a bias for each p-bit.

3.4 Annealing

In the context of p-bits, annealing refers to modifications to temperature that influence the rate at which p-bits update their states. A high temperature will encourage a network of p-bits to explore a wide range of states, while a low temperature will freeze the p-bits such that they fall into some lower energy state, and remain there. This strategy has been widely used in simulated annealing methodologies for optimization problems in the past [9]. While smaller networks of p-bits will converge to ground states without any annealing, larger connection matrices introduce local minima that the network can get stuck in. For optimization problems, it becomes especially difficult to deduce when the network's apparent ground state is the true ground state or a local minimum which the network cannot escape. It is therefore crucial to provide slow annealing features. This temperature translates to an inverse pseudo-temperature variable I_0 in software. Any annealing scheme for p-bits increments I_0 until the network converges to some state. The web simulator currently presents three options for the user to choose from. First, a constant I_0 that does not change. A constant I_0 is generally sufficient for small network sizes ($N_m \sim 10$).

(3)

$$I_o(t+1) = I_o$$

The second option is a linear annealing schedule, for which I_0 changes by some constant value each timestep. The user may select an $I_{end} > 0.0000001$ until which I_0 will increment over N_t timesteps.

(4)

$$I_o(t+1) = I_o(t) + \frac{I_{end}}{N_t}$$

$$I_o(0) = 0.0000001$$

The third option is a geometric annealing pattern. The user selects both a starting I_0 and an ending I_0 under this schedule, along with some growth factor. The starting I_0 is multiplied by the growth factor each iteration, for N_t timesteps.

(5)

$$\text{If gf} * I_o(t) < I_{end},$$

$$I_o(t+1) = \text{gf} * I_o(t)$$

$$\text{else } I_o(t+1) = I_o(t)$$

$$I_o(0) = I_{start}$$

As it is generally known that different annealing schedules are better suited to different energy landscapes, these annealing options will be expanded over time to maximize flexibility for the user. An example of annealing using the linear annealing schedule is given below in Figure 3-2.

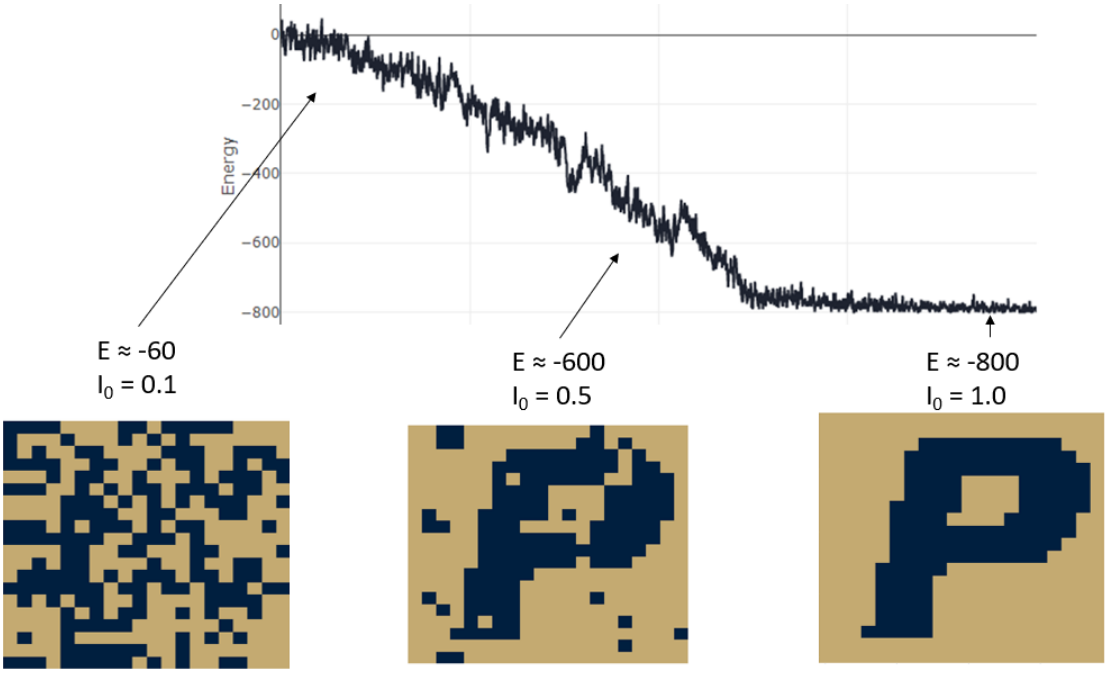

Figure 3-2 Linear annealing schedule applied to 20x20 p-bit grid

The connection matrix encodes a specific ground state in order to make clear that the annealing was successful in finding the correct ground state. The energy plot confirms the linear annealing schedule.

3.5 Simulator Outputs

Given the inputs as they are described, the simulator calculates and displays a series of data values, charts, and graphs. Each output is described in the order with which It appears on the webpage. Firstly, the final energy and final state of the network are displayed in text. This gives the user an exact final energy up to the 16^{th} significant place. For large networks with a larger absolute final energy, the value may be rounded to a lesser significant digit. The final state is shown with binary values rather than bipolar values. This final state is often important in optimization problems, where it generally (hopefully) is the ground state.

A heatmap is built from the final state to visually show the final state of the network. For large problems, this may help confirm whether the ground state was reached or provide some reference for the string of 1s and 0s provided in the final state text field. The pre-loaded 400 p-bit Max-Cut example utilizes this heat map to confirm the ground state of an optimization problem.

A histogram is built by compounding the state of the network at each timestep for sufficiently small networks. For certain machine learning problems, the probability of occupancy of each state may be important, which can be extracted from this histogram. The 6 p-bit Max-Cut problem uses this histogram to ensure that the most occurring state, and the second most occurring state solve the problem at hand.

Finally, an energy over time curve is presented to the user. This visualizes the effects of the annealing schedule and presents the energy of the state occupied at each time step over the Nt timesteps. A linear annealing schedule results in a linear energy decline, while a geometric annealing schedule will present an inverted exponential curve.

An extensive notification message system has been implemented as well. This system alerts the users to mis-inputs as shown in Figure 3-3. The messages are specific to each text field and tell the user exactly what the error is and how to fix it. This system supports displaying multiple errors at once and supports retention for as long as the user needs (the user can simply mouse over the error).

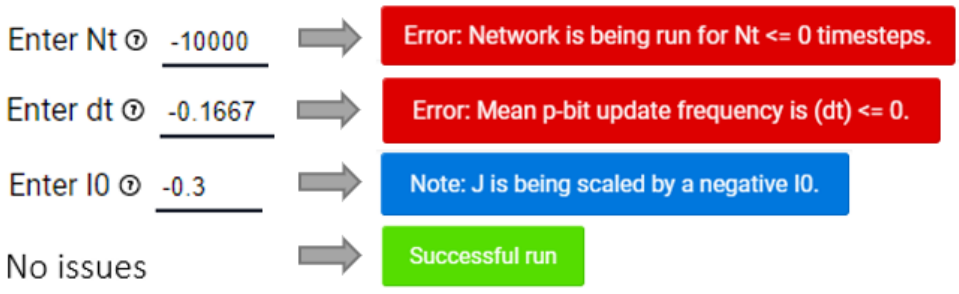

Figure 3-3 Notification message samples

The messages are color coded as follows.

RED – Error that needs to be fixed for the simulator to run.

BLUE – a note that isn't necessarily an error. The simulator will still run, although the output may not be valid.

Green – Successful run.

3.6 Pre-loaded Examples

A series of examples have been preset to ease the introduction of PSL for new users. These preset examples are simple parameter sets that are loaded into text fields on the web simulator. The text fields, and the algorithm behind the simulator, is the same for every example. Each example is explained in detail in its own tab above the simulator. The explanations follow a 2-step structure. First, they explain what the input is to the simulator in order to encode the given problem. This shows how a real-world problem can be translated into the PSL world. Second, the example shows how the output of the simulator can be interpreted within the context of the real world. This shows how PSL solved the encoded problem. There are currently 3 examples pre-loaded into the simulator. These examples grow in complexity, starting with a simple AND gate, then a 6 p-bit Max-Cut problem, then finally a 400 p-bit Max-Cut problem. Each of these examples is explored below.

3.6.1 And Gate

A p-circuits capability to model logic functions can be demonstrated with a simple AND gate. The three pins of an AND gate can be replaced with p-bits that execute roughly equivalent functions. Two p-bits serve as inputs and one as output. The difference though, is that while

conventional AND gates provide some output for set inputs, p-circuits can also provide inputs that satisfy some set output. The AND gate can be transformed into the simulator world as shown in Figure 3-4. Part a shows how a network of p-bits can emulate an AND gate's behavior, with certain weighted connections. These connections can be converted into a matrix, and input into the simulator as in part b of Figure 3-4.

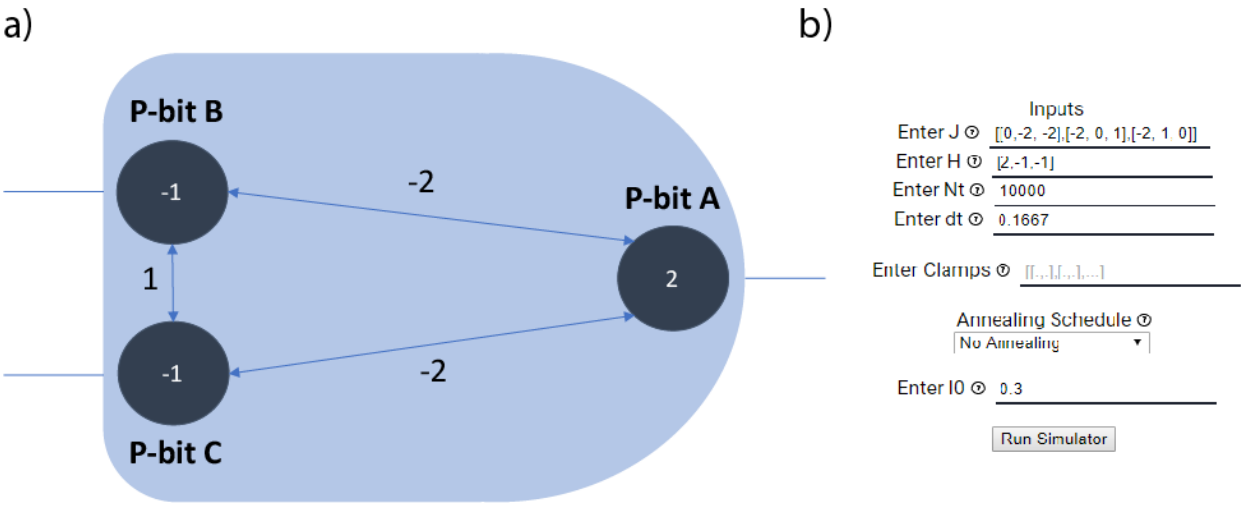

Figure 3-4 AND gate Input Parameters

Given these parameters, the simulator is run to reveal each of the outputs described in Section 3-3 (Simulator Outputs), shown in Figure 3-5. To derive the weights for any given function, such as an AND gate, the constraints set by the Hamiltonian for that function must be satisfied. The matrix satisfying the constraints becomes the J matrix whose ground state will minimize the function's Hamiltonian. Though I will not go into detail regarding the methods by which J-matrices are derived, others have invested time to this end [10].

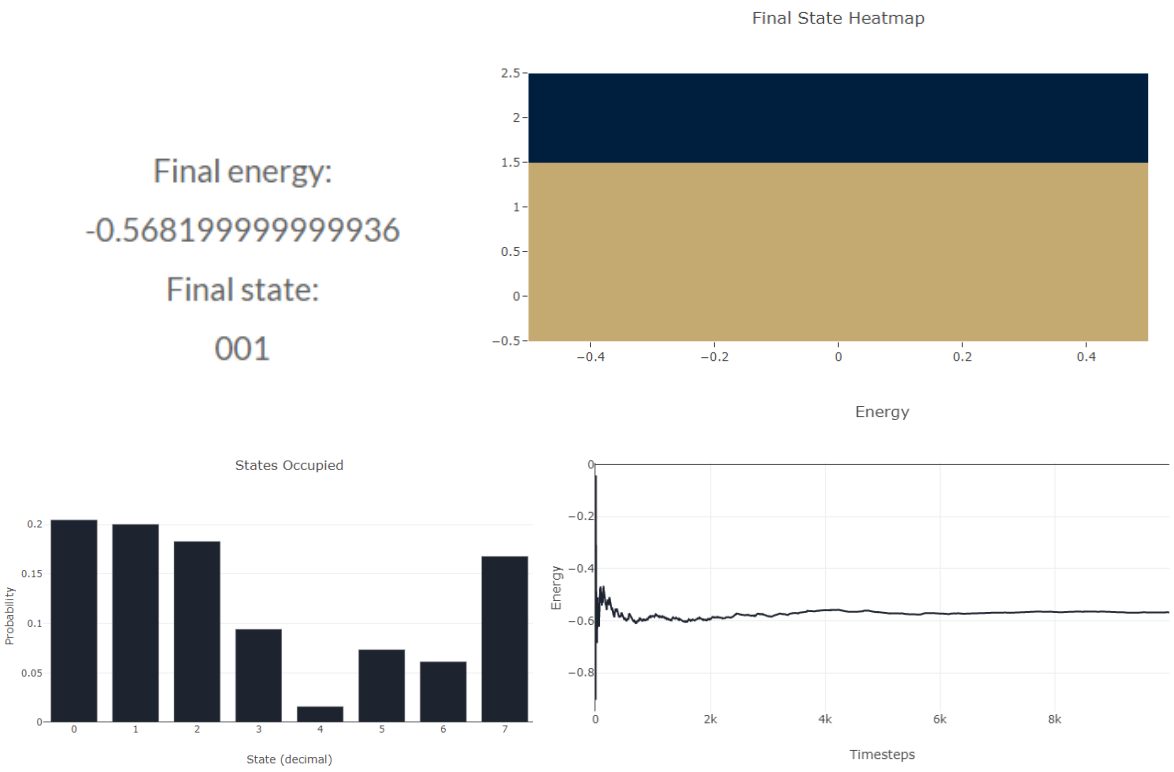

Figure 3-5 AND gate output

While the final state/heatmap and final energy are not as crucial for modelling an AND gate, the histogram shows the system to spend more time in the states that describe an AND gate. Figure 3-6 explains how this output confirms correct AND gate modelling by p-bits.

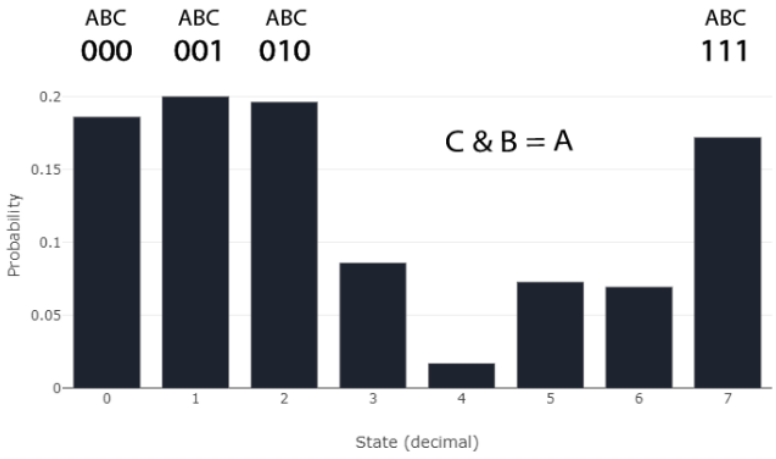

Figure 3-6 AND gate output description

When each state is read as a binary value, each bit in each state represents the state of a p-bit. The configuration of bits in each state show the state of the network. As described in Figure 3-6, only the binary configurations that satisfy A = B & C are occupied with high probability. This example further explores the capabilities of this simulator by allowing testing with the clamping system. By clamping p-bit A to 1, we see that the network conforms by keeping the inputs, p-bits B and C, in state 1. A state of 7 (111) is the only one that satisfies having p-bit A clamped to 1. This clamped input and output are shown in Figure 3-7. The remaining input parameters for this problem are the same as in Figure 3-4.

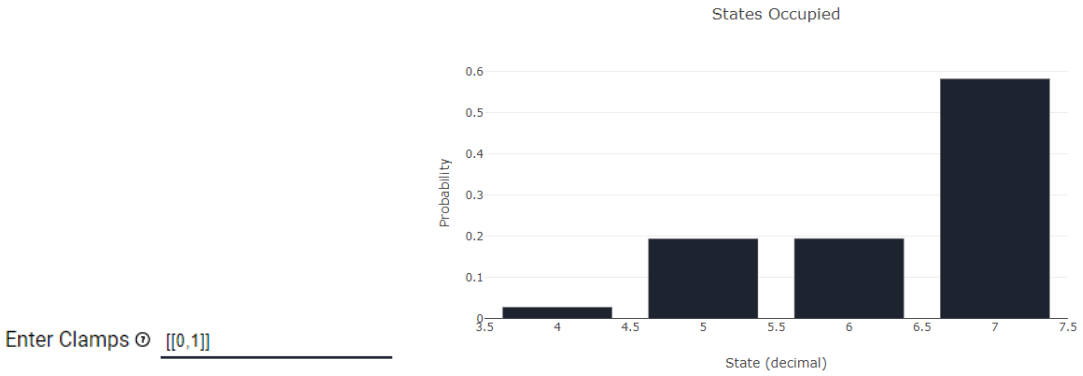

Figure 3-7 Clamping feature on AND gate

3.6.2 6 p-bit Max-Cut

While the AND gate demonstrates some simple logic gate functionality of PSL, this 6 p-bit Max-Cut example shows how specifically to map a graph onto the PSL framework. The Hamiltonian for Max-Cut is so similar to the biasing variable update Eq. 2 that to map a Max-Cut graph, one simply uses the adjacency matrix for the graph as the J-matrix. With this example, the user should understand how to map any graph they want onto the PSL framework. The input parameter set is shown in Figure 3-8, while the solution is explained in Figure 3-9.

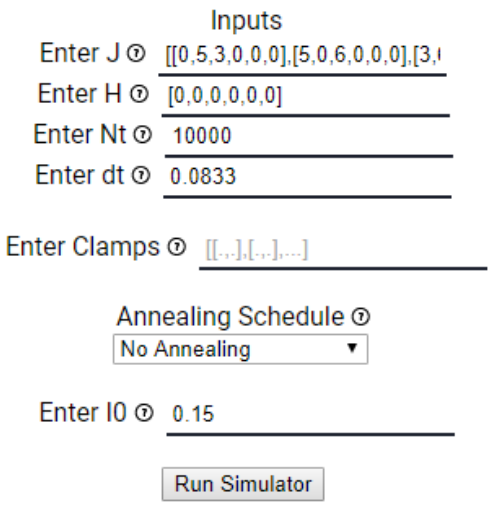

Figure 3-8 Six p-bit Max-Cut input parameters

A few modifications on the input parameter set of the AND gate should be pointed out. The J-matrix and H-vector are different to encode the desired graph. While the number of trials (N_t) remains the same, dt (mean p-bit update frequency) has been reduced. Experimentally, $dt < \frac{1}{2N_m}$ for arbitrarily connected networks prevents collisions to a degree that Parallel-PSL matches the statistics of Classical-PSL. In accordance with this ratio, dt was reduced to $\frac{1}{(2*6)}$ = 1/12. Since optimal convergence time is not a priority, a slower update rate can be afforded for increased precision. Since there are more inputs, and so the sums going into each p-bit are larger, I_0 has been slightly reduced to compensate. Scaling down the J matrix with a smaller I_0 ensures that the space is thoroughly explored while still showing large enough "correct" peaks.

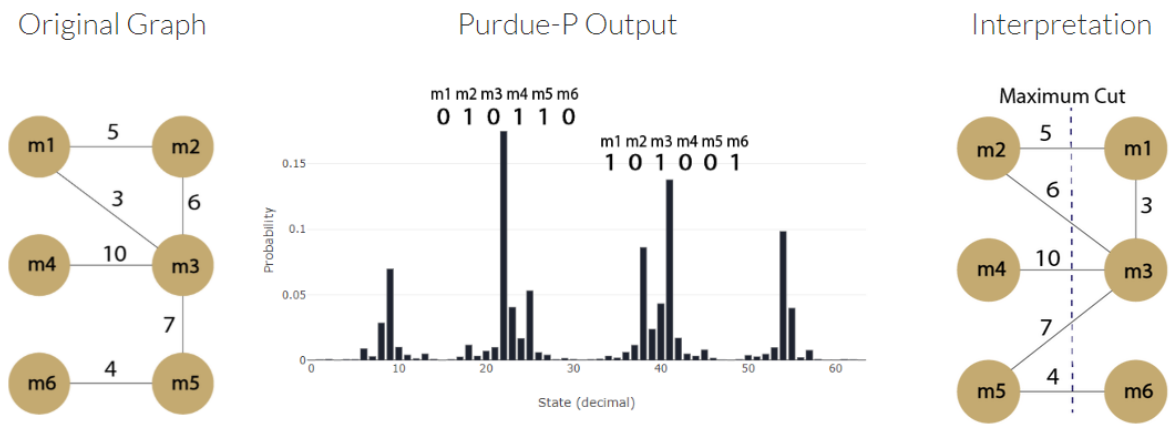

Figure 3-9 Six p-bit Max-Cut example output description

The 6 p-bit Max-Cut problem can be considered solved if the p-bits in the network occupy the states in which m_2, m_4, and m_5 are all one state, and m_1, m_3, and m_6 are all the other more often than they do every other possible state involving 6 bits. This can be considered a solution condition. There are 2^6 (64) possible states out of which 2 should stand out. The histogram shown in Figure 3-9 is the output of the web simulator, which clearly shows the two correct peaks standing out. These peaks satisfy our solution condition.

3.6.3 400 p-bit Max-Cut

Where the previous examples have taught how to map problems onto PSL, this example is a showcase of what PSL is capable of. While the 400 p-bit Max-Cut problem is technically a scaled-up form of the previous 6 p-bit Max-Cut problem, it encodes a less arbitrary J-matrix which converges to a regular final state pattern. The J-matrix is formed from a colored image overlaid onto a white background, as seen in Figure 3-10.

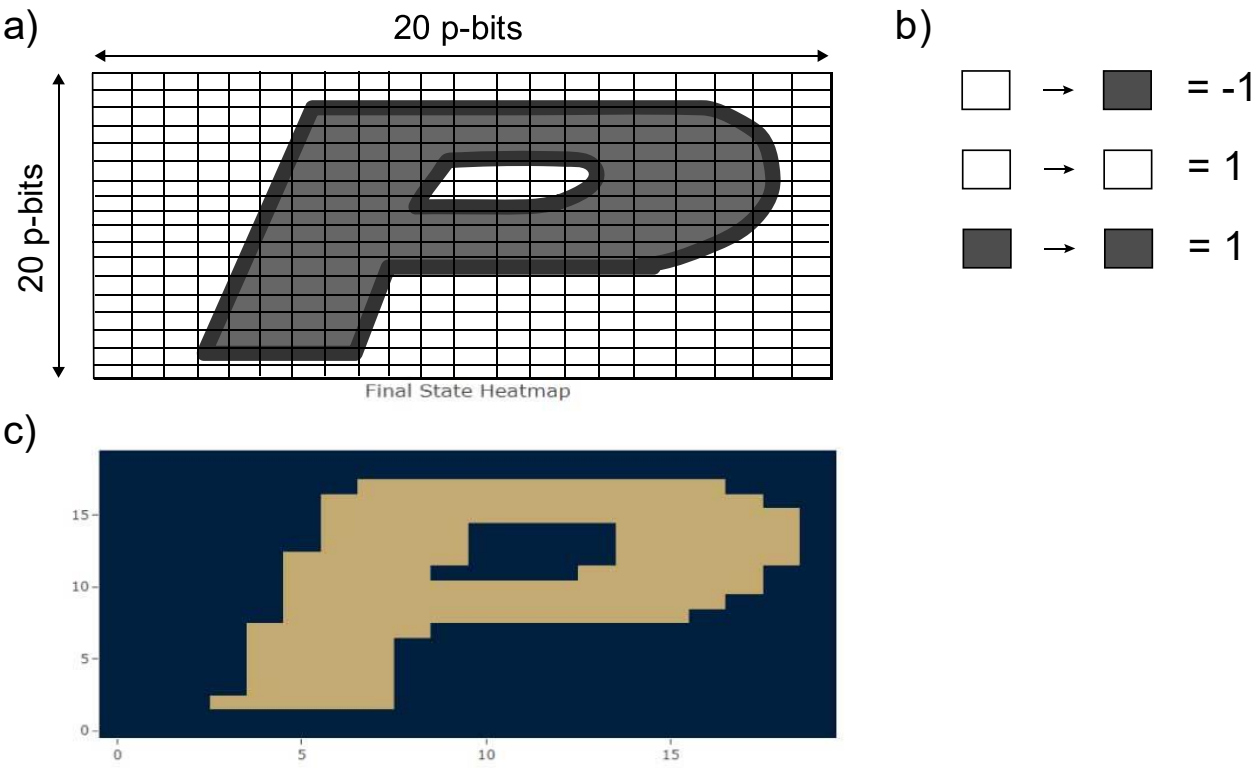

Figure 3-10 400 p-bit Max-Cut problem

A J-matrix that encodes this final ground state, with black pixels as 0,1 and white pixels as 1,0, can be formed by following two simple rules. P-bits of opposite colors must have a 1 connection, while p-bits of the same color must have a -1 connection. This antiferromagnetically couples opposite colors, forming edges in the final state. This example connects with scientific literature, where recognizable patterns are used to ensure that the algorithm converged to the correct final state for very large networks. The J-matrix is a 400x400 matrix, giving the output image 20 x 20 dimensions. The output by the web simulator on the J-matrix obtained as described is shown in part c of Figure 3-10.

The parameter dt was increased to 0.125 to obtain this output. For arbitrarily connected networks, $dt < \frac{1}{2N_m}$ is generally abided by. Since the maximum fan-in for any p-bit is 4 in this network, on average, 1/8th of the network could be updated at once with enough of a chance that two p-bits within the same group of 4 would not be updated together. Even if such an event occurs, enough "correct" events occur to overwrite their missteps on the energy landscape. Furthermore, since we are only interested in the final state, we are not as particular in following time-dynamic statistics of how we get there. A high collision rate resulting from a large dt would undoubtedly affect the time-dynamics of the network, providing incorrect peaks on a histogram, but this would nonetheless converge to an accurate solution.

3.7 Future Work

There is an assortment of developments that should improve the user experience with the simulator going forwards. The users could be allowed to save their parameter setups and give them a name, to allow simply loading their own presets at any time. Offloading the algorithmic scripts to a git hub repository, and linking that repository will improve page load time, and prevent the code from being discoverable through the web browser. Allowing file uploads for large J-matrix in CSV form rather than the bracket form could make running large networks much easier.

4. PURDUE-P FPGA CO-PROCESSOR

4.1 A Cloud Accessible Co-processor

The Purdue-P Co-processor is a cloud-hosted FPGA design for modelling and executing p-circuits asynchronously. In terms of execution, p-bits dispersed throughout the FPGA overcome the Von-Neuman bottleneck to a large degree compared to a general-purpose CPU running the same algorithms. This leads to noticeably faster convergence rates in the co-processor as opposed to MATLAB or Python running PSL equations. In addition to expedited execution of PSL, the co-processor runs an asynchronous PPSL framework which accurately models real hardware. The FPGA design has been compartmentalized such that different aspects of the PPSL algorithm may be swapped out, encouraging exploration for an optimal p-bit design in hardware. Access to the co-processor is provided via a MEX file for MATLAB, which will be downloadable from "purdue.edu/p-bit". Though this system has been built for MATLAB, the client C++ file (See Figure 4-1) through which a user's local computer communicates with the server can be attached to any user IDE. Support for Python, C++, and other languages can be made possible, further lowering the bar for access to the co-processor. Once connected, users can interact with the co-processor with commands that will affect the FPGA execution in real time, allowing users to modify temperature, pause, resume, and read the network state on the fly.

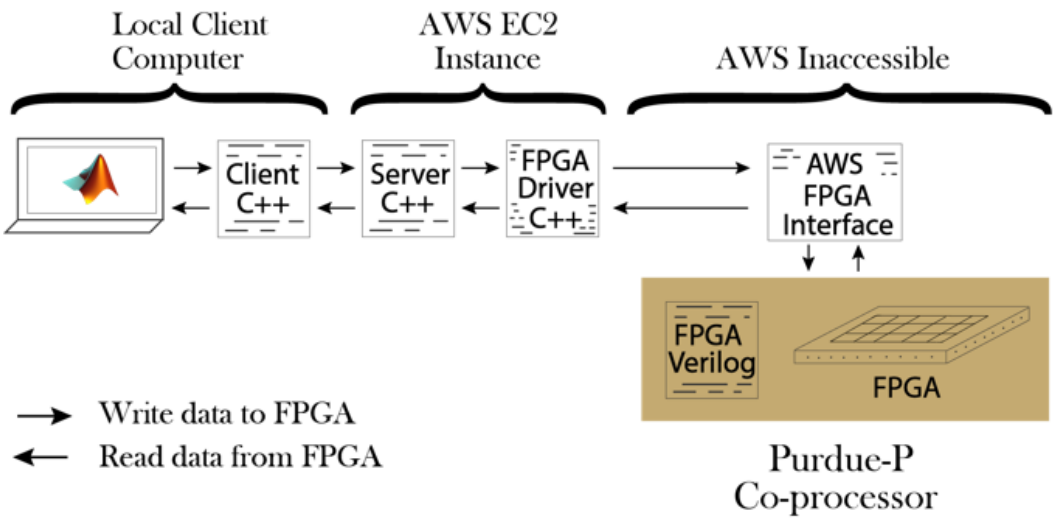

Figure 4-1 User to FPGA Connection Schematic

While the entire system labeled "Purdue-p Co-processor" in Figure 4-1 is provided, the user must download the libraries that are used by the client C++ file for the client C++ file to compile properly, and for data to be transmitted between the client and server. Boost.asio, cereal, and Eigen are three libraries that are crucial for this process to function. The role of these libraries will be briefly explored.

Boost.asio is an optimized, asynchronous IO framework that simplifies transmitting and receiving data between a client and a server. Boost.asio functions compiled on the client side require a destination socket and data, with corresponding security protocols satisfied. The destination socket is established by the "Server C++" file on the AWS server in Figure 4-1. For Boost.asio to send data, that data should be in a form that is transmittable, a series of bytes. Matrices especially, among other types of stacked, multi-dimensional data, are difficult to transmit and receive. As such, it is required to "serialize" the data before transmission. This arranges matrices, classes, and other multi-dimensional data containers into a single, easily transmittable string of bytes. This operation is handled within the Purdue-P Co-processor framework by the

cereal library. Since p-bit data will be read out repeatedly over the course of a single run, it is important that the data be read and written quickly and accurately. Since both the client and server files use the same cereal library, the method used to serialize the data is used in reverse to deserialize the data on the other side and re-create the data in its original dimensions. Finally, cereal accepts data in the form of vectors and matrices of the Eigen datatype. The Eigen library is used to format the data such that it can quickly be serialized and transmitted.

4.2 Connecting to the Cloud – Tutorial

An extensive sequence of steps can be found in Appendix A, under "Instructions on Accessing Purdue-P Co-processor" that guide users in connecting to the Purdue-P Co-processor. The following will be a more general overview of these steps. The entire setup sequence can be split into three phases. The first of these phases sets up the client side, referring to the blue box in Figure 4-1. These directions involve downloading the client C++ file form purduep.weebly.com and installing the appropriate libraries that will allow MATLAB to compile that C++ file. At the end of this phase, a compiled client C++ file should be present on the user's computer.

The second phase sets up a server on Amazon Web Services (AWS) with the FPGA connection mechanisms in place and instructs the user on how to access that server. This begins with the creation of AWS account and proceeds through the setup of an instance on the cloud. This instance is the server with which the user's MATLAB client will communicate. To communicate with this server, a free software called PuTTY is recommended. Instructions on how to SSH into the server with PuTTY follow. Other applications exist, like SecureCRT, through which the user can also access the server. There are specific key files and settings that must be set within PuTTY for the connection to bypass the server security protocols, which are outlined.

The final phase activates the server with some final setup procedures that run the server C++ file in Figure 4-1. Prior to this step, all the necessary files and commands have been pre-loaded onto the server, but the server has yet to be activated. The commands and setup procedures are provided. At the end of this phase of instructions, the user can run *./psl_server* in a specific place within the server directory and run the server. Again, Appendix A contains very thorough setup instructions with explanations at more important steps.

The server is ready to communicate with MATLAB at this stage. Specified commands within MATLAB will set up PSL problems to run on the FPGA, and these commands can be used to read the p-bit states at any point during execution to collect time-dynamic information.

APPENDIX A

Client Side

1. Download MATLAB
2. Install the libraries necessary to compile the client-side C++ file
    a. Eigen, found at http://eigen.tuxfamily.org/index.php?title=Main_Page
    b. Boost.asio, found at https://www.boost.org/
    c. Cereal, found at https://uscilab.github.io/cereal/

    Eigen and cereal are header only libraries, meaning they do not require an installation, simply download the .zip folders, extract the files into some location, and link their libraries using the command in step 4 with the paths to each library modified.
3. Download the c++ client file psl_accel.cpp from https://purduep.weebly.com/fpga-tool.html
4. Run the following command in the MATLAB terminal with pertaining paths to each library

```
mex psl_accel.cpp -I/usr/local/Cellar/cereal/1.2.2/include/ -
I/usr/local/Cellar/eigen/3.3.7/include/eigen3/ -
I/usr/local/Cellar/boost/1.67.0_1/include/ -
L/usr/local/Cellar/boost/1.67.0_1/lib/ -lboost_system;
```

    With that compiled, the client is ready to send requests up to the server, we must now establish a server with the fpga.

Server Side

1. Create an AWS account
2. Set up payment information so that AWS can allow you to create an FPGA server
3. Start an instance
    a. Use the Purdue-P FPGA User AMI (custom built AMI with all the settings pre-installed)
    b. f1.2xlarge instance type
    c. 65 gb volume
    d. Security groups default
    e. Default settings for rest of steps
4. Set up a keypair for authentication, store your keypair safely

5. Go to security groups, select the group associated with your instance, add a custom TCP rule with port 1234, accessible by anyone if you are ok with anyone using your instance, or accessible by "My IP" to keep it locked to your current IP.

The server is now set up with all the required FPGA files. You must now access the server and spin up the FPGA so that it can communicate with your MATLAB client.

Download PuTTY to access the server you have started.

6. The keypair you have downloaded will be a .pem file, while PuTTY uses .ppk. use PuTTYgen to convert the .pem file into a .ppk file
7. Right click on your instance (found in the instance tab along the left side) in the AWS console and click **instance state → start**
8. Set up PuTTY to access the server
    a. In the IP address box, type "centos@" and paste the IP address given in the AWS console for your server under "Public DNS (IPv4)"
    b. Connection type should be ssh
    c. Under **connection → ssh → auth** on the left side of PuTTY go to browse under "private key file for authentication" and link the .ppk file you have created using your AWS provided keypair
    d. Type a name for your session under "saved sessions" and save it so that you don't have to re-connect your keypair on the next login

    Each successive login onto your server from PuTTY will require you to replace the IP address with the AWS provided public DNS address, since this will change each time you spin up the server.
9. Click Open to enter the server
10. Go to **src → project_data → aws_fpga**
11. Go to **./hdk → cl → developer_designs → … (will be modified soon)**

    Run the following command to end in the correct folder

```
cd /home/centos/src/project_data/aws-fpga/hdk/cl/developer_designs/…
```

12. Run

```
make clean all
```

    A file named "psl_server" should appear

    Some final server configurations are necessary

13. Type "aws configure"

   a. Set the access key and secret access key found in your aws account by clicking on your name in the top right corner, clicking on “my security credentials”. Go to access keys, and generate a new access key and secret access key.
   b. Set the default region name to be the region your instance is running on, found on your aws console. Ex. “us-east-1”
14. Load the bitstream onto the FPGA based on which topology setup you would like to use, there are currently two available topologies
   a. 500 all-to-all

```
sudo fpga-load-local-image -S 0 -I agfi-…
```

   b. 90x90 nearest neighbor

```
sudo fpga-load-local-image -S 0 -I agfi-…
```

If you are unsure of which topology, the 500 p-bit all-to-all will be a safe bet for any small to medium sized problem.

15. Ensure that port 1234 is set as custom TCP rule in **security group** (your selected security group) → **inbound rules** on your AWS console
16. To run the server, run “sudo ./psl_server”

```
sudo ./psl_server
```

Your server is officially running and accepting commands from your MATLAB client!

Soon only a compiled AMI will be released to the public, so that the user will not need to access source code for the FPGA designs. Rather the server should spin up immediately following the launch of a new instance.

APPENDIX B

Scroll down indicator on starting header (HTML, CSS, JavaScript)

```
<html>
  <head>
  <style>
      @import url(https://fonts.googleapis.com/css?family=Josefin+Sans:300,400);

.demo a {
  position: absolute;
  top: 50px;
  z-index: 2;
  display: inline-block;
  -webkit-transform: translate(0, -50%);
  transform: translate(0, -50%);
  color: #000;
  font : normal 400 20px/1 'Josefin Sans', sans-serif;
  letter-spacing: .1em;
  text-decoration: none;
  transition: opacity .3s;
}
.demo a:hover {
  opacity: .5;
}

#section07 a {
  padding-top: 80px;
}
#section07 a span {
  position: absolute;
  top: 0;
  left: 50%;
  width: 24px;
  height: 24px;
  margin-left: -12px;
  border-left: 1px solid #000;
  border-bottom: 1px solid #000;
  -webkit-transform: rotate(-45deg);
  transform: rotate(-45deg);
  -webkit-animation: sdb07 2s infinite;
  animation: sdb07 2s infinite;
  opacity: 0;
  box-sizing: border-box;
}
#section07 a span:nth-of-type(1) {
  -webkit-animation-delay: 0s;
  animation-delay: 0s;
}
#section07 a span:nth-of-type(2) {
  top: 16px;
  -webkit-animation-delay: .15s;
  animation-delay: .15s;
}
#section07 a span:nth-of-type(3) {
  top: 32px;
  -webkit-animation-delay: .3s;
```

```
  animation-delay: .3s;
}
@-webkit-keyframes sdb07 {
  0% {
    opacity: 0;
  }
  50% {
    opacity: 1;
  }
  100% {
    opacity: 0;
  }
}
@keyframes sdb07 {
  0% {
    opacity: 0;
  }
  50% {
    opacity: 1;
  }
  100% {
    opacity: 0;
  }
}



</style>
  </head>
  <body>

<section id="section07" class="demo">
 <a href="#ajsection-overview"><span></span><span></span><span></span></a>
</section>

</body>
</html>
```

Animated horizontal line on scroll down (HTML, CSS, JavaScript)

```
<html>
  <head>

  <style>
    body
    {
      background: #ffffff;

      margin: 0;
      align-items: center;
      justify-content: center;
      text-align: center;
      font-family: Helvetica neue, roboto;
    }

    h1
    {
```

```
      color: #bdbdbd;
      font-weight: 300;
    }
    .line
    {
      background: #c4aa71;
      display: block;
      margin: 0 auto;
      opacity: 1;
      height: 2px;
      transition: 1s;
      width: 0%;

    }
    .line.show
    {
      opacity: 1;
      width: 100%;
    }
  </style>
  </head>
  <body>
  <div class="line"></div>

        <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
        <script>window.jQuery || document.write('<script src="js/vendor/jquery-1.11.2.min.js"><\/script>')</script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/ScrollMagic/2.0.5/ScrollMagic.min.js"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/ScrollMagic/2.0.5/plugins/debug.addIndicators.min.js"></script>
   <script>
        $(document).ready(function(){

    // Init ScrollMagic
    var controller = new ScrollMagic.Controller();

    // build a scene
    var ourScene = new ScrollMagic.Scene({
        triggerElement: '.line'
    })
    .setClassToggle('.line', 'show') // add class to project01
    //.addIndicators({
    //  name: 'fade scene',
    //  colorTrigger: 'black',
    //  indent: 200,
    //  colorStart: '#75C695'
    //}) // this requires a plugin
    .addTo(controller);

});
        </script>

  </body>

</html>
```

Web Simulator Code (HTML, CSS, JavaScript)

```
<!DOCTYPE html>
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/mathjs/5.9.0/math.min.js"></script>
  <script src="https://code.jquery.com/jquery-3.4.0.min.js"></script>
  <script src="https://cdn.jsdelivr.net/gh/StephanWagner/jBox@v0.6.4/dist/jBox.all.min.js"></script>
  <link rel="stylesheet" type="text/css" href="styles.css">
  <link href="https://cdn.jsdelivr.net/gh/StephanWagner/jBox@v0.6.4/dist/jBox.all.min.css" rel="stylesheet">

  <style>
  .all_inputs .input{
    background: transparent;
        border: 0;
        border-bottom: 2px solid #1E2330;
        padding: 4px;
        outline: none;

  }
  </style>

  </head>
<body>
  <div class='all_inputs'>
    <div id="specify rand_Flag">
        <label> User specified or random network</label><span class="tooltip" title="Run a random network or specify the weights for a custom network"><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-fill.png" alt="10pt-qc-no-fill" border="0"></span>
        <br>
        <select id="rand_choice">
          <optgroup label="General Use">
          <option value = "1"> Specify </option>
          <option value = "0"> Random </option>
          </optgroup>
          <optgroup label="Pre-loaded Networks">
          <option value = "2"> AND gate </option>
          <option value = "3"> 6 p-bit Max-Cut </option>
          <option value = "4"> 400 p-bit Max-Cut </option>
          </optgroup>
        </select>
    <button id="hide" onclick='hide(); pre_load();'> Select </button>
    </div>
    <div id='allinput' style='display: none;'>
<fieldset>

    <div id='inputs_random' class = "class_inputs_random" style="display: none;">
```

```
46.           <legend> Inputs </legend>
47.           <label> Enter Nm </label><span class="tooltip" title="Number of mag
   nets in the p-circuit."><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-
   fill.png" alt="10pt-qc-no-fill" border="0"></span>
48.           <input type='text' class="input" id='Nm_rand_text' value='50'/>
49.           <br>
50.           <label>Enter Nt </label><span class="tooltip" title="Number of time
   -steps to collect data."><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-
   fill.png" alt="10pt-qc-no-fill" border="0"></span>
51.           <input type='text' class="input" id='Nt_rand_text' value='10000'/>

52.           <br>
53.           <label> Enter dt </label><span class="tooltip" title="Mean update f
   requency for p-
   bits. A small dt leads to more simultaneous updates, while a larger dt leads t
   o fewer."><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-fill.png" alt="10pt-
   qc-no-fill" border="0"></span>
54.           <input type='text' class="input" id='dt_rand_text' value='0.01'/>
55.           <br>
56.           <br>
57.          <div id="annealing_schedule">
58.          <label> Annealing Schedule </label> <span class="tooltip" title="Sim
   ulated annealing slowly cools the system (increases I0) following some schedul
   e, encouraging the network to find the global minimum."><img src="https://i.ib
   b.co/wgYvNP5/10pt-qc-no-fill.png" alt="10pt-qc-no-fill" border="0"></span>
59.          <br>
60.          <select id="anneal_choice_rand" onChange="check_anneal(1)">
61.          <option value = "0"> No Annealing </option>
62.          <option value = "1"> Linear Annealing </option>
63.          <option value = "2"> Exponential Annealing </option>
64.          </select>
65.          <br><br>
66.          </div>
67.
68.          <div id='show_no_anneal_r'>
69.           <label> Enter I0 </label><span class="tooltip" title="A constant sc
   aling factor for the J-matrix. I0 corresponds to inverse-
   temperature. "><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-
   fill.png" alt="10pt-qc-no-fill" border="0"></span>
70.           <input type='text' class="input" id='I0_rand_text' value='1'/>
71.          </div>
72.          <div id='show_anneal_lin_r' style="display: none;">
73.           <label> Enter ending I0 </label><span class="tooltip" title="I0 is
   modified each timestep following I0(t+1) = I0(t) + Iend/Nt. This is equivalent
    to linearly reducing temperature to encourage the system to converge to the g
   lobal minimum."><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-
   fill.png" alt="10pt-qc-no-fill" border="0"></span>
74.           <input type='text' class="input" id='I0_start_lin_rand_text' value=
   '1'/>
75.          </div>
76.          <div id='show_anneal_geo_r' style="display: none;">
77.           <label> Enter starting I0 </label><span class="tooltip" title="I0 i
   s modified each timestep following I0(t+1) = growth_factor * I0(t). This expon
   entially reduces temperature to encourage the system to converge to the global
```

```
    minimum."><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-fill.png" alt="10pt-qc-no-fill" border="0"></span>
78.             <input type='text' class="input" id='I0_start_geo_rand_text' value='0.001'/>
79.             <br>
80.             <label> Enter growth factor</label><span class="tooltip" title="The growth factor by which I0 is scaled each timestep, i.e growth_factor in I0(t+1) = growth_factor * I0(t)."><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-fill.png" alt="10pt-qc-no-fill" border="0"></span>
81.             <input type='text' class="input" id='I0_growth_rand_text' value='1.001'/>
82.           </div>
83.             <br>
84.             <br>
85.             <button onclick='output(); check_anneal(1);'> Run Simulator </button>
86.             <br>
87.             <br>
88.     </div>
89.     <div id='inputs_non_random' class='class_inputs_non_random' style="display: none;">
90.             <legend> Inputs </legend>
91.             <label>Enter J </label><span class="tooltip" title="The connection matrix describing the weights between p-bits"><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-fill.png" alt="10pt-qc-no-fill" border="0"></span>
92.             <input type='text' class="input" id='J_text' placeholder="[[.,.,.],[.,.,.],...]"/>
93.             <br>
94.             <label>Enter H </label><span class="tooltip" title="The bias vector describing initial state of p-bits"><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-fill.png" alt="10pt-qc-no-fill" border="0"></span>
95.             <input type='text' class="input" id='H_text' placeholder="[.,.,.]"/>
96.             <br>
97.             <label>Enter Nt </label><span class="tooltip" title="Number of time-steps to collect data"><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-fill.png" alt="10pt-qc-no-fill" border="0"></span>
98.             <input type='text' class="input" id='Nt_text' placeholder="Nt"/>
99.             <br>
100.                <label>Enter dt </label><span class="tooltip" title="Mean update frequency for p-bits. A small dt leads to more simultaneous updates, while a larger dt leads to fewer."><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-fill.png" alt="10pt-qc-no-fill" border="0"></span>
101.                <input type='text' class="input" id='dt_text' placeholder="dt"/>
102.                <br>
103.                <br>
104.                <label> Enter Clamps </label><span class="tooltip" title="Enter which p-bits should be clamped, if any, formatted as [[p-bit #, clamp value],[.,.],...]. Clamp values must be +-1."><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-fill.png" alt="10pt-qc-no-fill" border="0"></span>
```

```
105.                 <input type='text' class="input" id='clamps' placeholder="[[.
    ,.],[.,.],...]"/>
106.                 <br>
107.                 <br>
108.
109.                 <div id="annealing_schedule">
110.                 <label> Annealing Schedule </label> <span class="tooltip" titl
    e="Simulated annealing slowly cools the system (increases I0) following some s
    chedule, encouraging the network to find the global minimum."><img src="https:
    //i.ibb.co/wgYvNP5/10pt-qc-no-fill.png" alt="10pt-qc-no-
    fill" border="0"></span>
111.                 <br>
112.                 <select id="anneal_choice" onChange="check_anneal(0)">
113.                 <option value = "0"> No Annealing </option>
114.                 <option value = "1"> Linear Annealing </option>
115.                 <option value = "2"> Exponential Annealing </option>
116.                 </select>
117.                 <br>
118.                 <br>
119.                 </div>
120.                 <div id='show_no_anneal'>
121.                 <label> Enter I0 </label><span class="tooltip" title="A const
    ant scaling factor for the J-matrix. I0 corresponds to inverse-
    temperature. "><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-
    fill.png" alt="10pt-qc-no-fill" border="0"></span>
122.                 <input type='text' class="input" id='I0_text' placeholder="Co
    nstant I0"/>
123.                 </div>
124.                 <div id='show_anneal_lin' style="display: none;">
125.                 <label> Enter ending I0 </label><span class="tooltip" title="
    I0 is modified each timestep following I0 = I0 + Iend/Nt. This is equivalent t
    o linearly reducing temperature to encourage the system to converge to the glo
    bal minimum."><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-
    fill.png" alt="10pt-qc-no-fill" border="0"></span>
126.                 <input type='text' class="input" id='I0_start_lin_text' place
    holder="I0 end"/>
127.                 </div>
128.                 <div id='show_anneal_geo' style="display: none;">
129.                 <label> Enter starting I0 </label><span class="tooltip" title
    ="I0 is modified each timestep following I0(t+1) = growth_factor * I0(t). This
     exponentially reduces temperature to encourage the system to converge to the
    global minimum."><img src="https://i.ibb.co/wgYvNP5/10pt-qc-no-
    fill.png" alt="10pt-qc-no-fill" border="0"></span>
130.                 <input type='text' class="input" id='I0_start_geo_text' place
    holder="I0 start"/>
131.                 <br>
132.                 <label> Enter growth factor</label><span class="tooltip" titl
    e="The growth factor by which I0 is scaled each timestep, i.e growth_factor in
     I0(t+1) = growth_factor * I0(t)."><img src="https://i.ibb.co/wgYvNP5/10pt-qc-
    no-fill.png" alt="10pt-qc-no-fill" border="0"></span>
133.                 <input type='text' class="input" id='I0_growth_text' placehol
    der="I0 growth factor"/>
134.                 </div>
135.                 <br>
136.                 <br>
```

```
137.                 <button onclick='output(); check_anneal(0);'> Run Simulator <
   /button>
138.                 <br>
139.                 <br>
140.               </div>
141.
142.
143.         </fieldset>
144.         </div>
145.
146.       <br>
147.       <br>
148.       <div id='out'><p id='out_data'></p></div>
149.       <div id='chart'></div>
150.       <div id='state histogram'></div>
151.       <div id='heatmap'></div>
152.
153.       </div>
154.       <script>
155.       console.log('made it here');
156.       var error_flag;
157.       var I0;
158.
159.       new jBox('Tooltip', {
160.         attach: '.tooltip'
161.       });
162.
163.       function hide(){
164.           var rand_flag = JSON.parse(document.getElementById("rand_choice").va
   lue);
165.           var x1 = document.getElementById('inputs_non_random');
166.           var x2 = document.getElementById('inputs_random');
167.           var x3 = document.getElementById('allinput');
168.           check_anneal(1-rand_flag);
169.           x3.style.display = 'inline';
170.           if(rand_flag >= 1){      // specify
171.             x1.style.display = 'inline';
172.             x2.style.display = 'none';
173.           }
174.           else{                                // random
175.             x2.style.display = 'inline';
176.             x1.style.display = 'none';
177.           }
178.       }
179.       function pre_load(){
180.         var rand_flag = JSON.parse(document.getElementById("rand_choice").valu
   e);
181.         if(rand_flag == 1) // empty
182.         {
183.           document.getElementById("Nt_text").value = '';
184.           document.getElementById("dt_text").value = '';
185.           document.getElementById("I0_text").value = '';
186.           document.getElementById("I0_start_lin_text").value = '';
187.           document.getElementById("I0_start_geo_text").value = '';
188.           document.getElementById("I0_growth_text").value = '';
```

```
189.          document.getElementById("J_text").value = '';
190.          document.getElementById("H_text").value = '';
191.
192.        }
193.        if(rand_flag == 2) // AND gate
194.        {
195.          document.getElementById("Nt_text").value = "10000";
196.          document.getElementById("dt_text").value = "0.1667";
197.          document.getElementById("I0_text").value = "0.3";
198.          document.getElementById("I0_start_lin_text").value = "1";
199.          document.getElementById("I0_start_geo_text").value = "1";
200.          document.getElementById("I0_growth_text").value = "1.001";
201.          document.getElementById("J_text").value = "[[0,-2, -2],[-2, 0, 1],[-
   2, 1, 0]]";
202.          document.getElementById("H_text").value = "[2,-1,-1]";
203.        }
204.        else if(rand_flag == 3) // 6 p-bit maxcut
205.        {
206.          document.getElementById("Nt_text").value = "10000";
207.          document.getElementById("dt_text").value = "0.0833";
208.          document.getElementById("I0_text").value = "0.15";
209.          document.getElementById("I0_start_lin_text").value = "0.3";
210.          document.getElementById("I0_start_geo_text").value = "0.001";
211.          document.getElementById("I0_growth_text").value = "1.001";
212.          document.getElementById("J_text").value = "[[0,5,3,0,0,0],[5,0,6,0,0
   ,0],[3,6,0,10,7,0],[0,0,10,0,0,0],[0,0,7,0,0,4],[0,0,0,0,4,0]]";
213.          document.getElementById("H_text").value = "[0,0,0,0,0,0]";
214.
215.        }
216.         else if(rand_flag == 4) // 400 p-bit maxcut
217.        {
218.          document.getElementById("Nt_text").value = "1000";
219.          document.getElementById("dt_text").value = "0.125";
220.          document.getElementById("I0_text").value = "1";
221.          document.getElementById("I0_start_lin_text").value = "1";
222.          document.getElementById("I0_start_geo_text").value = "0.001";
223.          document.getElementById("I0_growth_text").value = "1.001";
224.          document.getElementById("J_text").value =[400x400 j-
   matrix for purdue-p symbol
225.      ]";
226.
227.        }
228.      }
229.
230.
231.      function check_anneal(rand){
232.          // if rand == 0, non random anneal_choice div
233.          // if rand == 1, rand anneal_choice div
234.
235.          if(JSON.parse(rand))
236.            anneal_str = document.getElementById('anneal_choice_rand').value;

237.          else
238.            anneal_str = document.getElementById('anneal_choice').value;
239.
```

```
240.          //var anneal = JSON.parse(document.getElementById('anneal_choice').v
   alue);
241.          var anneal = JSON.parse(anneal_str);
242.          console.log(anneal);
243.          var x1 = document.getElementById('show_no_anneal');
244.          var x2 = document.getElementById('show_anneal_lin');
245.          var x3 = document.getElementById('show_anneal_geo');
246.          var x4 = document.getElementById('show_no_anneal_r');
247.          var x5 = document.getElementById('show_anneal_lin_r');
248.          var x6 = document.getElementById('show_anneal_geo_r');
249.
250.
251.          if(anneal == 0){
252.            x1.style.display = 'inline';
253.            x2.style.display = 'none';
254.            x3.style.display = 'none';
255.            x4.style.display = 'inline';
256.            x5.style.display = 'none';
257.            x6.style.display = 'none';
258.          }
259.          else if(anneal == 1){
260.            x1.style.display = 'none';
261.            x2.style.display = 'inline';
262.            x3.style.display = 'none';
263.            x4.style.display = 'none';
264.            x5.style.display = 'inline';
265.            x6.style.display = 'none';
266.          }
267.          else{
268.            x1.style.display = 'none';
269.            x2.style.display = 'none';
270.            x3.style.display = 'inline';
271.            x4.style.display = 'none';
272.            x5.style.display = 'none';
273.            x6.style.display = 'inline';
274.          }
275.
276.      }
277.
278.      function ppsl_e(*) {
279.
280.        PROPRIETARY, WILL BE PUBLISHED SOON!
281.      }
282.      function error_handling(J, H, dt, Nt, Nm, I0){
283.
284.
285.      }
286.      function toast(colors, contents){
287.
288.            new jBox('Notice', {
289.              content: contents,
290.              color: colors,
291.              delayOnHover: true
292.              });
293.      }
```

```
294.
295.       function execute_anneal(anneal, I0_cur, I0_const, end_lin, start_geo, gr
   owth, Nt){
296.         var I0_next;
297.         if(anneal == 0)  // constant
298.           I0_next = I0_const;
299.         if(anneal == 1)  // linear
300.         {
301.           I0_next = I0_cur+ (end_lin/Nt);
302.         }
303.         else if(anneal == 2) // geometric
304.         {
305.           I0_next = growth * I0_cur;
306.         }
307.
308.         return I0_next;
309.       }
310.
311.       function output(){
312.
313.           //check if you need to create random J,h
314.           var rand_flag = JSON.parse(document.getElementById("rand_choice").va
   lue);
315.           if(rand_flag >= 1){    // specify
316.               var Nt = document.getElementById("Nt_text").value;
317.               var dt = document.getElementById("dt_text").value;
318.               var I0_const = document.getElementById("I0_text").value;
319.               var I0slt = document.getElementById("I0_start_lin_text").value;

320.               var I0sgt = document.getElementById("I0_start_geo_text").value;

321.               var I0gt = document.getElementById("I0_growth_text").value;
322.               var anneal = JSON.parse(document.getElementById("anneal_choice")
   .value);
323.               var clamps = document.getElementById("clamps").value;
324.
325.               if(isNaN(Nt))
326.                 toast('red','Error: Nt is non-numeric.');
327.               if(isNaN(dt))
328.                 toast('red','Error: dt is non-numeric.');
329.               if(isNaN(I0_const))
330.                 toast('red','Error: I0 is non-numeric.');
331.               if(isNaN(I0slt))
332.                 toast('red','Error: I0 start is non-numeric.');
333.               if(isNaN(I0sgt))
334.                 toast('red','Error: I0 start is non-numeric.');
335.               if(isNaN(I0gt))
336.                 toast('red','Error: I0 growth factor is non-numeric.');
337.
338.
339.               if(anneal == 0)
340.                 I0_const = JSON.parse(I0_const);
341.               else if(anneal == 1)
342.                 I0slt = JSON.parse(I0slt);
343.               else if(anneal == 2)
```

```
344.              {
345.                I0sgt = JSON.parse(I0sgt);
346.                I0gt = JSON.parse(I0gt);
347.              }
348.              if(clamps.length != 0)
349.              {  clamps = JSON.parse(clamps);
350.                console.log(clamps);
351.
352.              }
353.              else
354.                clamps = JSON.parse("[[-1]]")
355.              dt = JSON.parse(dt);
356.              Nt = JSON.parse(Nt);
357.
358.
359.              var J = JSON.parse(document.getElementById("J_text").value);
360.              var H = JSON.parse(document.getElementById("H_text").value);
361.              var Nm = H.length;
362.          }
363.          else{
364.            var Nm = document.getElementById("Nm_rand_text").value;
365.            var Nt = document.getElementById("Nt_rand_text").value;
366.            var dt = document.getElementById("dt_rand_text").value;
367.            var I0_const = document.getElementById("I0_rand_text").value;
368.            var I0slt = document.getElementById("I0_start_lin_rand_text").valu
    e;
369.            var I0sgt = document.getElementById("I0_start_geo_rand_text").valu
    e;
370.            var I0gt = document.getElementById("I0_growth_rand_text").value;
371.            var clamps = JSON.parse("[[-1]]")
372.
373.
374.            if(isNaN(Nm))
375.              toast('red','Error: Nm is non-numeric.');
376.            if(isNaN(Nt))
377.              toast('red','Error: Nt is non-numeric.');
378.            if(isNaN(dt))
379.              toast('red', 'Error: dt is non-numeric.');
380.            if(isNaN(I0_const))
381.              toast('red', 'Error: I0 is non-numeric.');
382.            if(isNaN(I0slt))
383.              toast('red', 'Error: I0 start is non-numeric.');
384.            if(isNaN(I0sgt))
385.              toast('red', 'Error: I0 start is non-numeric.');
386.            if(isNaN(I0gt))
387.              toast('red', 'Error: I0 growth factor is non-numeric.');
388.
389.            I0_const = JSON.parse(I0_const);
390.            I0slt = JSON.parse(I0slt);
391.            I0sgt = JSON.parse(I0sgt);
392.            I0gt = JSON.parse(I0gt);
393.            dt = JSON.parse(dt);
394.            Nt = JSON.parse(Nt);
395.            Nm = JSON.parse(Nm);
```

```
396.             var anneal = JSON.parse(document.getElementById("anneal_choice_ran
   d").value);
397.
398.             var J = [];
399.             var H = [];
400.             var hold=0;
401.             for(i=0;i<Nm;i++){
402.               J[i] = [];
403.               for(k=0;k<i;k++){
404.                 hold = math.random()*2-1;
405.                 J[i][k] = hold;
406.                 J[k][i] = hold;
407.               }
408.               J[i][i] = 0;
409.               H[i] = math.random()*2-1;
410.             }
411.           }
412.
413.
414.           // Error handling
415.           // Errors get red toast, notices get blue toast, successful run gets
    green toast at end of plotting
416.           error_flag = 0;
417.           if(Nt <= 0)
418.           {
419.             toast('red', 'Error: Network is being run for Nt <= 0 timesteps.')
   ;
420.             error_flag = 1;
421.           }
422.            if(Nm <= 0)
423.           {
424.              toast('red', 'Error: Network has Nm <= 0 p-bits.');
425.              error_flag = 1;
426.           }
427.           if(dt <= 0)
428.           {
429.             toast('red', 'Error: Mean p-
   bit update frequency is (dt) <= 0.');
430.             error_flag = 1;
431.           }
432.           if(I0_const < 0)
433.           {
434.             toast('blue', 'Note: J is being scaled by a negative I0.');
435.           }
436.           if(J.length != H.length || J[0].length != H.length)
437.           {
438.             toast('red', 'Error: Mismatch of # of rows and columns in J and le
   ngth of H.');
439.             error_flag = 1;
440.           }
441.           if(anneal == 2 && I0sgt == 0)
442.           {
443.             toast('red', 'Error: geometric annealing cannot start at 0');
444.             error_flag = 1;
445.           }
```

```
446.          if(anneal == 2 && I0sgt < 0)
447.          {
448.            toast('blue', 'Note: Geometric annealing is starting at a negative
    number.');
449.          }
450.          if(anneal == 2 && I0gt < 1)
451.          {
452.            toast('blue', 'Note: Geometric annealing with growth rate < 1 will
    increase temperature, preventing convergence.');
453.          }
454.         if(anneal == 1 && I0slt < 0)
455.          {
456.            toast('blue', 'Note: Linear annealing with ending < 0 will increas
   e temperature, preventing convergence.');
457.          }
458.
459.
460.          if(error_flag == 0)
461.          {
462.              var energy_state_laststate = ppsl_e(J, H, Nt, dt, anneal, I0_con
   st, I0slt, I0sgt, I0gt, clamps);
463.              energy = energy_state_laststate[0];
464.              state = energy_state_laststate[1];
465.              last_state = energy_state_laststate[2];
466.
467.              var xrange = [];
468.              for(i=0;i<energy.length;i++)
469.                xrange[i] = i;
470.              var trace1_line = {
471.                                  x: xrange,
472.                                  y: energy,
473.                                  type: 'line',
474.                                  marker: {color: "rgb(30,35,48)"}
475.                                };
476.
477.              var hist_end = math.pow(2,Nm)+1;
478.              var trace1_hist = {
479.                                    x: state,
480.                                    type: 'histogram',
481.                                    histnorm: 'probability',
482.                                    xbins: {
483.                                            end: hist_end,
484.                                            size: 1,
485.                                            start: -0.5
486.                                          },
487.                                    marker: {color: "rgb(30,35,48)",
488.                                              line: {
489.                                              color: "grey",
490.                                              width: 0.5
491.                                          }}
492.              }
493.              var data_line = [trace1_line];
494.              var data_hist = [trace1_hist];
495.              var layout_hist = {
496.                                  title: "States Occupied",
```

```
497.                                xaxis: {title: "State (decimal)"},
498.                                yaxis: {title: "Probability"},
499.                                bargap: 0.25
500.
501.              };
502.              var layout_line = {
503.                                title: "Energy",
504.                                xaxis: {title: "Timesteps"},
505.                                yaxis: {title: "Energy"}
506.              };
507.
508.              // print converged energy and state
509.              document.getElementById("out_data").innerHTML = 'Final energy:<b
   r>';
510.              document.getElementById("out_data").innerHTML += energy[Nt-1];
511.              document.getElementById("out_data").innerHTML += "<br>Final stat
   e:<br>";
512.              for(ii=0; ii<last_state.length; ii++)
513.              {
514.                if(last_state[ii] == -1)
515.                  last_state[ii] = 0;
516.                document.getElementById("out_data").innerHTML += last_state[ii
   ];
517.              }
518.
519.
520.
521.              Plotly.purge('chart');
522.              Plotly.purge('state histogram');
523.              Plotly.purge('heatmap');
524.              Plotly.plot('chart', data_line, layout_line);
525.              if(Nm < 15){
526.                Plotly.plot('state histogram', data_hist, layout_hist);
527.              }
528.
529.
530.              // find the optimal length and width for Nm magnets to show heat
   map
531.              // find all the factors, use the middle 2
532.              factors = [];
533.              for (i=1; i < last_state.length+1; i++)
534.              {
535.                if (math.mod(Nm, i) == 0)
536.                  factors.push(i);
537.                if(i**2 == Nm)
538.                  factors.push(i);
539.              }
540.              if(factors.length >= 2){
541.                var len = factors.length/2;
542.                var reshaped = math.reshape(last_state, [factors[len],factors[
   len-1]]);
543.                var colorscaleValue = [
544.                  [0, '#C4aa71'],
545.                  [1, '#001f3f']
546.                ];
```

```
547.                 var data = [
548.                 {
549.                   z: reshaped,
550.                   type: 'heatmap',
551.                   colorscale: colorscaleValue,
552.                   showscale: false
553.                 }
554.                 ];
555.                 var layout = {
556.                 title: 'Final State Heatmap',
557.               };
558.
559.                 Plotly.newPlot('heatmap', data, layout);
560.               } // if there are greater than 2 factors
561.               else{
562.                 toast("blue","Note: Cannot print heatmap for this network")
563.               }
564.
565.             toast('green', 'Successful run');
566.           }// error_flag == 0
567.           else{
568.             Plotly.purge('chart');
569.             Plotly.purge('state histogram');
570.             Plotly.purge('heatmap')
571.           }
572.
573.       }
574.       </script>
575.
576.       </body>
577.       </html>
```

REFERENCES

[1] A. Z. Pervaiz, L. A. Ghantasala, K. Y. Camsari, and S. Datta, “Hardware emulation of stochastic p-bits for invertible logic,” *Sci. Rep.*, vol. 7, no. 1, pp. 1–13, 2017.

[2] K. Yunus Camsari, R. Faria, B. M. Sutton, and S. Datta, “Stochastic p-Bits for Invertible Logic.”

[3] Y. Bengio, N. Léonard, and A. Courville, “Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation.”

[4] R. Faria, K. Y. Camsari, and S. Datta, “Low-Barrier Nanomagnets as p-Bits for Spin Logic,” *IEEE Magn. Lett.*, vol. 8, 2017.

[5] A. Z. Pervaiz, B. M. Sutton, L. A. Ghantasala, and K. Y. Camsari, “Weighted p-bits for FPGA implementation of probabilistic circuits,” pp. 1–6, 2017.

[6] K. Y. Camsari, S. Chowdhury, and S. Datta, “Scalable Emulation of Stoquastic Hamiltonians with Room Temperature p-bits.”

[7] B. Sutton, K. Yunus Camsari, B. Behin-Aein, and S. Datta, “Intrinsic optimization using stochastic nanomagnets OPEN,” *Nat. Publ. Gr.*, 2017.

[8] H. Suzuki, J. I. Imura, Y. Horio, and K. Aihara, “Chaotic Boltzmann machines,” *Sci. Rep.*, vol. 3, pp. 1–5, 2013.

[9] Y. Nourani and B. Andresen, “A comparison of simulated annealing cooling strategies,” 1998.

[10] A. Lucas, “Ising formulations of many NP problems,” vol. 2, no. February, pp. 1–15, 2013.

VITA

Lakshmi Anirudh Ghantasala

EDUCATION

BSEE In Electrical and Computer Engineering, Purdue University, West Lafayette, IN. Minors: English and Biology. Research Interests: Graphene transistor fabrication, copper transistor interconnects. Graduated May, 2017.

Pursuing MSEE In Electrical and Computer Engineering, Purdue University, West Lafayette, IN. Research focus: Probabilistic neural networks. Intended graduation May 2019.

PUBLICATIONS

For publications, please see publications section.

PUBLICATIONS

[1] A. Z. Pervaiz, L.A.Ghantasala, K.Y.Camsari, S.Datta, "Hardware emulation of stochastic p-bits" *Scientific Reports - Nature*, vol. 7 , published 8 sep, 2017.

[2] A. Z. Pervaiz, B. Sutton, L.A.Ghantasala, et. al, "Weighted p-bits for FPGA Implementation of Probabilistic Circuits" *IEEE TNNLS*. Published 30 oct, 2018