Introduction to Probabilistic Computing

Kyle Lee

August 2024

*The MATLAB and Mathematica codes provided in the folder correspond directly to the contents described here. I recommend that you go play with those codes yourself to gain a better understanding of what you read here. If you have questions, ChatGPT is genuinely helpful for much of this content.

1 What is probabilistic computing?

Classical computers feature deterministic bits in a 0 or 1 state, employing conventional transistor technology. Quantum computers involve qubits in a delicate superposition of 0 and 1, typically requiring cryogenic temperatures in order to minimize thermal noise. Probabilistic bits (p-bits) lie between these two computing paradigms: they probabilistically flip between 0 and 1 states at a very high rate.

As the name suggests, probabilistic computers leverage randomness to do useful computations. One application of p-bits is quantum-inspired computing: quantum algorithms can be mapped to p-bit networks. While quantum computers promise incredible performance for solving certain classes of problems, they have not yet proven to be feasible with current technology. p-bits, on the other hand, promise to be feasible and scalable using present-day technology. p-bits are also widely applicable to machine learning applications (for example, in Boltzmann machines, a type of neural network for unsupervised ML tasks).

Probabilistic computers also promise to solve challenging combinatorial optimization problems, providing orders of magnitude improvement in prefactor. A famous example is the traveling salesman problem. Many practical, real-world problems have proven to be of similar difficulty, thus solving combinatorial optimization problems is of vital importance.

As the transistor scaling predicted by Moore's law comes to an end, there is increasing interest in domain-specific circuits and hardware accelerators. Probabilistic computers fall into this category: they don't replace existing computers for general computations. Rather, they are tailored to solve specific classes of problems with superior performance.

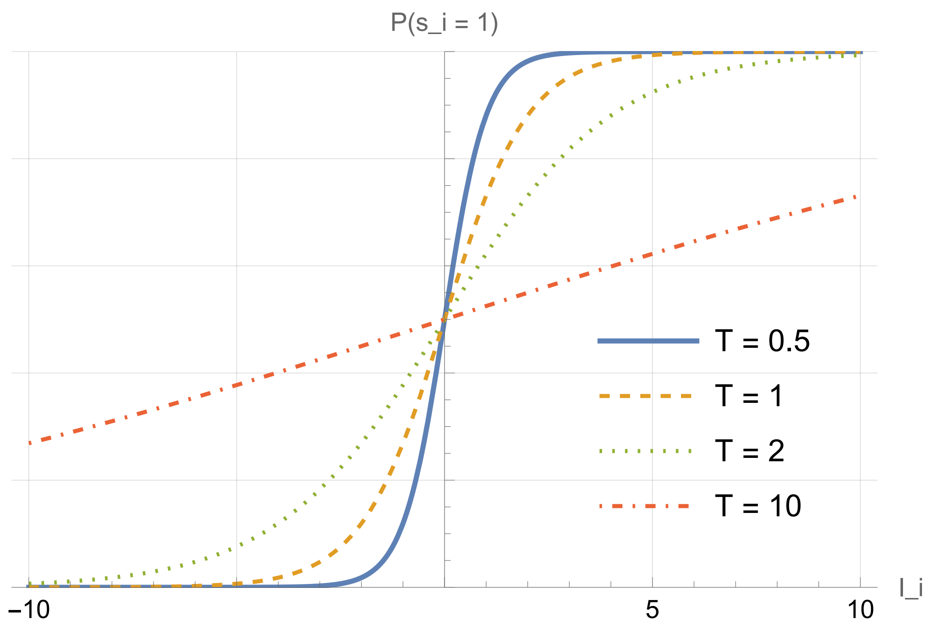


Figure 1: Each curve represents a single p-bit operating under constant temperature T, with an input I_i. As T increases, the curve "flattens". Intuitively, as T decreases, the p-bit becomes more sensitive to changes in the input I_i: inputting a I_i that is slightly above 0 will make the p-bit very likely to be a $+1$, and vice versa.

2 What is a probabilistic bit?

The probabilistic bit (p-bit) is the basic unit or "building block" of probabilistic computers. Unlike a classical bit, which deterministically represents 0 or 1, a p-bit has a certain *probability* of being a 0 or 1 state at any given time. The state of a p-bit is not fixed. A p-bit flips its state stochastically, conditioned on an input to the p-bit. A single p-bit is described by the following equation:

$$P(s_i = 1) = \frac{1}{1 + e^{-I_i/T}} \tag{1}$$

Where I_i is the input to the p-bit, T is absolute temperature, and s_i is the state of the p-bit, which is either 0 or 1. This relationship is shown graphically as a sigmoid function in Fig. 1. If $I_i = 0$, then $P(s_i = 1) = 1/2$, meaning that there is a 50% chance of being in the 1 state and a 50% chance of being in the 0 state. As the input to the p-bit I_i increases in the positive direction, the p-bit becomes more likely to be a 1. As I_i becomes more negative, the p-bit is more likely to be a 0.

Note that we often use *bipolar* variables, as opposed to binary variables. Instead of having a binary state $s_i \in \{0, 1\}$, we use a bipolar state $m_i \in \{-1, 1\}$.

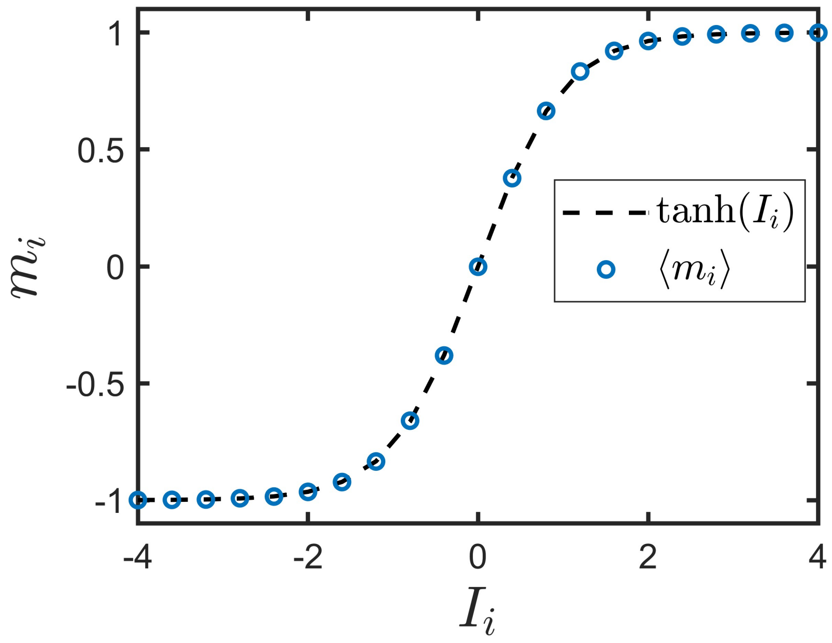


Figure 2: Temperature T is set to 1. For a given input I_i, a single p-bit is simulated by repeatedly applying the update rule (Eq. 2) and measuring the state m_i, which is either -1 or +1. Visually, as I_i increases, the average spin m_i is closer to +1, and vice versa.

We sometimes call m_i a 'spin', particularly in physics contexts. We'll use bipolar variables for all of our discussion henceforth. We introduce the p-bit update rule for software simulations of p-bits (sometimes referred to as an activation function):

$$m_i = \text{sgn}[\tanh(I_i/T) - \text{rand}_u(-1, 1)] \tag{2}$$

where sgn is the sign function, returning a +1 if its argument is positive and a -1 if its argument is negative. tanh is the hyperbolic tangent function (which looks like a sigmoid function, except transformed to be in the range $[-1, 1]$ instead of $[0, 1]$). $\text{rand}_u(-1, 1)$ is a random number between -1 and 1. m_i is the bipolar state of the p-bit, taking the value +1 or -1. I_i is the input to the p-bit. Note that often, instead of using temperature T, we prefer to use inverse temperature $\beta = 1/T$, in which case you would have a $\tanh(\beta I_i)$ in Eq. 2.

If you are doing a software simulation of a p-bit, then its current state is either +1 or -1. We then apply the update rule (Eq. 2) in order to determine the next state of the p-bit, conditioned on the p-bit's current input I_i. According to this update rule, the p-bit probabilistically decides to flip or remain in the same state.

In order to simulate a single p-bit in software, you would assign it some initial state, $m_i = +1$ or -1. Given some temperature T and some input I_i, you would then apply the p-bit update rule (Eq. 2). The p-bit would flip with a certain probability, or it may remain in the same state. You apply the same update rule repeatedly, sequentially. This would result in a stream of p-bit states: $m_i = +1, -1, +1, +1, \ldots$, and you will find that the frequency with which you measure the +1 state obeys the probability distribution in Eq. 1 and Fig. 1. For example, if I_i is 0, you would observe half +1 and half −1 states. If I_i is a large positive value, then you would observe almost all +1 states. This is illustrated in Fig. 2

3 Coupling 2 p-bits

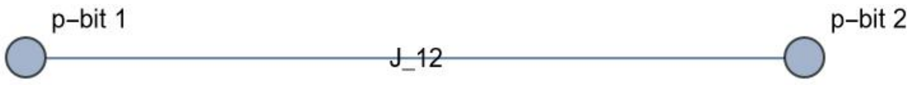


Figure 3: 2 p-bits coupled together by a weight J_{12}. We denote the state of the first p-bit as m_1 and the state of the second p-bit as m_2 (at any given time, each p-bit is in a +1 or -1 state). If J_{12} is positive, the p-bits tend to align with each other, whereas if J_{12} is negative, the p-bits tend to disagree with each other.

We now (hopefully) understand what a single p-bit does. However, a single p-bit flipping between a -1 and +1 state is not particularly useful or interesting. In order to construct a probabilistic computer, we must *couple* many p-bits together in a network and have them interact with each other. Let's consider the simple example of 2 p-bits coupled together by some weight J_{12}, which can be positive or negative (if $J_{12} = 0$, then the p-bits would be independent/uncoupled).

We are going to define an energy E for the system shown in Fig. 3. This may seem a bit arbitrary/abstract, but bear with me:

$$E = -J_{12}m_1m_2 - h_1m_1 - h_2m_2 \tag{3}$$

where J_{12} is the coupling strength between the 2 p-bits, m_1 and m_2 are the states of the 2 respective p-bits, and h_1 and h_2 are the *biases* of each respective p-bit. In certain contexts you will see the bias h referred to as a "field", as in a magnetic field which influences the direction of spins (if each p-bit represents a +1 or -1 spin, then they tend to align in the same direction as their bias. For example, if the bias on the first p-bit, $h_1 = 1$, then p-bit #1 would have an increased likelihood of being in the +1 state due to this bias).

The way that coupling p-bits works is that we make the input to p-bit 1, I_1, dependent on the state of p-bit 2. Similarly, we make the input to p-bit 2, I_2, dependent on the state of p-bit 1. Furthermore, we configure the inputs to each p-bit, I_i, such that the system naturally tends to minimize the energy that we defined in Eq. 3. Suppose we are applying the update rule (Eq. 2) to p-bit 1, while p-bit 2 is in a fixed state. The input to p-bit 1 is:

$$I_1 = -\frac{\partial E}{\partial m_1} = -\frac{E(m_1 = +1) - E(m_1 = -1)}{1 - (-1)} = J_{12} m_2 + h_1 \tag{4}$$

Recall that as I_i increases in the positive direction, a p-bit is more likely to flip to a +1 state than a -1 state after applying the update rule (and vice versa). We want to configure I_1 such that p-bit 1 is most likely to flip to a state that decreases the energy E when we apply the update rule to p-bit 1, given that the state of the other p-bit m_2 is held constant. If the system's energy after updating m_1 to the $+1$ state would be lower than the energy after updating m_1 to the -1 state ($E(m_1 = +1) - E(m_1 = -1) < 0$), then I_1 is positive. If I_1 is positive, then according to our update rule (Eq. 2), the p-bit is more likely to update to a $+1$ than a -1, and thus more likely to update to the state m_1 that minimizes system energy.

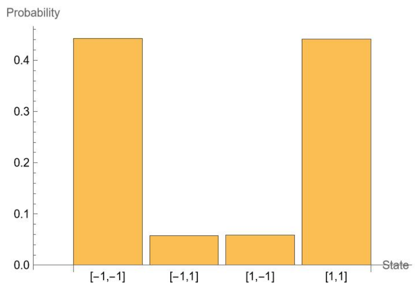
We configure the input to the second p-bit, I_2, in the exact same way:

$$I_2 = -\frac{\partial E}{\partial m_2} = -\frac{E(m_2 = +1) - E(m_2 = -1)}{1 - (-1)} = J_{12} m_1 + h_2 \tag{5}$$

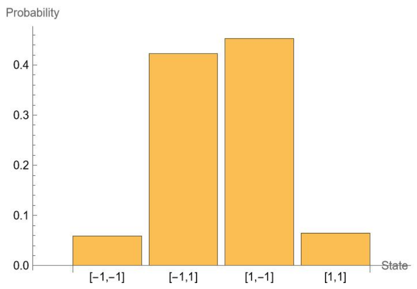
where, as we apply the update rule to p-bit 2, the state of p-bit 1 (m_1) is held constant during the update. The way that you would simulate this 2 p-bit system in software is as follows. Initialize m_1 and m_2 to some random initial state. Apply the update rule (Eq. 2) to the first p-bit, where the input $I_1 = J_{12} m_2 + h_1$. The first p-bit probabilistically decides whether to flip its state. After the first p-bit has updated, apply the update rule to the second p-bit, where $I_2 = J_{12} m_1 + h_2$. The second p-bit now probabilistically decides whether to flip its state. Then keep alternating: apply the update rule to p-bit 1, then p-bit 2, then p-bit 1, ...

When we have finished applying the update rule to all p-bits in the system sequentially, we sample the state of the system. For example, I apply the update rule to p-bit 1, then apply the update rule to p-bit 2, and then I record the state of the system, which might look like $m_1 = +1$, $m_2 = -1$. I then do it again: I apply the update rule to p-bit 1, then apply the update rule to p-bit 2, and then record the state of the system. This is referred to as Gibbs sampling. Each sample that I take is referred to as a Monte Carlo sweep. This is our notion of "time" when simulating p-bits in software.

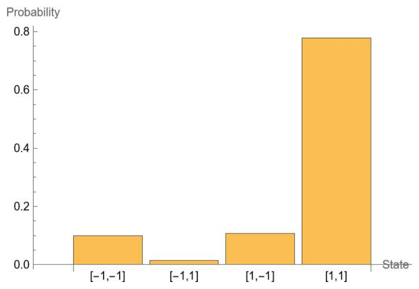
It is important to note that it does not work quite the same way when p-bits are implemented in hardware using spintronic devices such as magnetic tunnel junctions (MTJs). In hardware, p-bits do not explicitly have sequential updates where the update rule is applied to the first p-bit, then the second, then the first. They are just rapidly flipping in continuous time. However, we observe the same general system properties as relates to the frequency with which we sample certain system energies E or certain spin configurations $\{m\}$, which is what we really care about for practical application ($\{m\}$ is a list representing the state of the system; for example, in the 2 p-bit case, $\{m\} = \{m_1, m_2\}$).



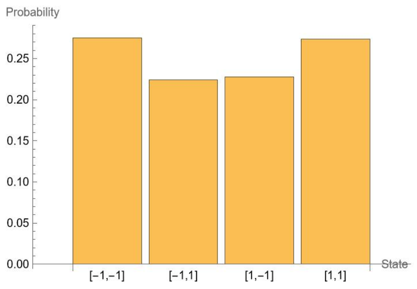
(a) $T = 1$. The coupling weight J_{12} is set to $+1$, and biases h_1 and h_2 are set to 0. The p-bits tend to agree or align, because that is the lowest energy configuration. We call this a ferromagnetic interaction.



(b) $T = 1$. The coupling weight J_{12} is set to -1, and biases h_1 and h_2 are set to 0. The p-bits tend to disagree, because that is the lowest energy configuration. We call this anti-ferromagnetic.



(c) $T = 1$. $J_{12} = +1$, $h_1 = +1$, and $h_2 = 0$. The bias on p-bit 1 encourages p-bit 1 to be in the $+1$ state.



(d) $J_{12} = +1$, and $T = 10$. Increasing the temperature tends to make the system explore more states and makes updates more random. As T approaches infinity in Eq. 2, I_i stops mattering, and the p-bit has a 50/50 chance of flipping to ± 1.

Figure 4: We simulate a 2 p-bit system for 10000 Monte Carlo sweeps (meaning we have sequentially updated p-bit 1 and p-bit 2 10000 times and we have taken 10000 samples of the system state). We express the frequency with which we sample each of 4 possible spin configurations as a probability.

This written explanation can be quite dry and challenging to understand. Please refer to Fig. 3 to gain some basic intuition, and play with parameters J, h, T in the provided Mathematica code.

4 The Ising Model

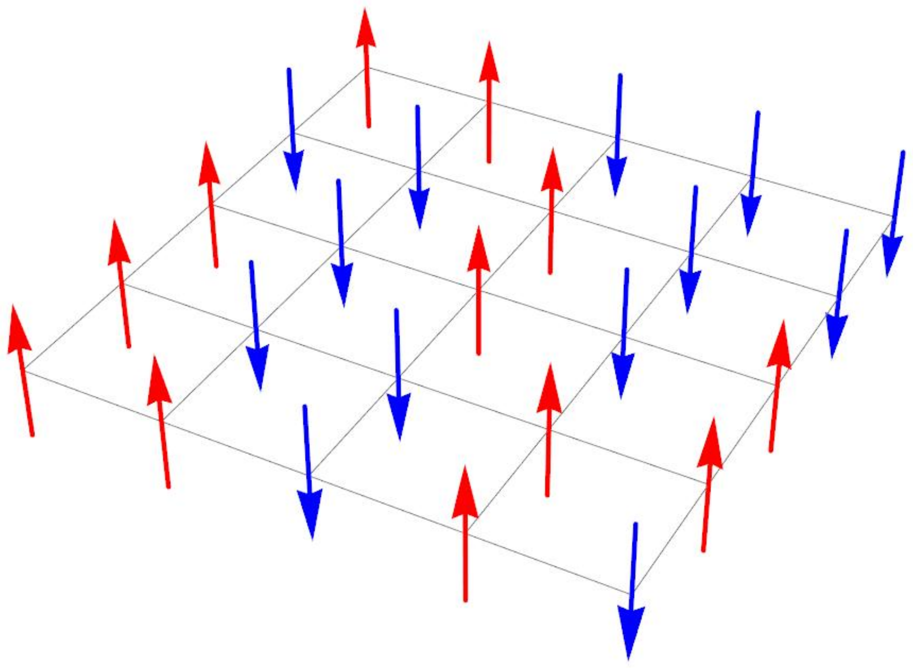


Figure 5: The 2D Ising model on a lattice.

In order to generalize what we did with 2 p-bits to a greater number of p-bits, we introduce the Ising model. The Ising model is a very popular model among physicists, used to describe physical phenomena such as magnetism and phase transitions (For example, heating up a magnetic material like iron causes a phase transition, similar to melting ice, and the material exhibits very different properties). As computer engineers, the Ising model is useful because challenging problems of practical interest can be mapped to the Ising model.

Now, instead of just 2 p-bits, we have N p-bits. A single p-bit can be coupled to multiple other p-bits now (we call the other p-bits "neighbors"). The input I_i to the ith p-bit in the system now depends on the state of all its neighbors:

$$I_i = \sum_j J_{ij} m_j + h_i \tag{6}$$

where J_{ij} is the coupling weight between the ith p-bit and its neighbor, the jth p-bit, and as before, h_i is the bias on the ith p-bit. Eq. 6 is often called the synapse equation, or a synaptic function. We also generalize the energy E that we had previously defined for a system of 2 p-bits. The Ising energy, sometimes called the Hamiltonian of the system, is:

$$E(\{m\}) = -\frac{1}{2}\sum_{i,j} J_{ij} m_i m_j - \sum_i h_i m_i = -\sum_{i<j} J_{ij} m_i m_j - \sum_i h_i m_i \qquad (7)$$

Note that both sides of the equals sign are equivalent summations. The 1/2 on the left is to avoid double-counting neighbors. Without the 1/2, you would double count the energy contribution of the same neighbors; for example, without the 1/2, you would end up with $J_{12}m_1m_2 + J_{21}m_2m_1$ in your final expression for energy, and since $J_{12} = J_{21}$, these terms are identical and it is double counted. If you apply the same logic as Eq. 4 to Eq. 7, you will obtain the correct synapse function for a system of N p-bits (Eq. 6).

Visually, when we apply the Ising model, it is a graph with nodes and edges. Each node is a p-bit or a spin, which can be in a +1 or -1 state. Each edge is a coupling weight between 2 p-bits. A classic example is the 2D Ising model on a lattice, shown in Fig. 5. If all the weights between each p-bit $J_{ij} = +1$, it looks like a ferromagnet; all spins tend to align in one direction or the other, because that is the lowest energy configuration. If all the weights $J_{ij} = -1$, then it looks anti-ferromagnetic, and you'll observe adjacent spins disagreeing with each other. At high temperature, thermal fluctuations dominate, and the configuration will look random (recall that as $T \to \infty$, p-bit updates become 50/50).

To conduct Gibbs sampling on a system of N p-bits, you apply the update rule (Eq. 2) sequentially to each p-bit in the system (update m_1 based on the state of all the other p-bits in the system. Then update m_2 based on the state of all the other p-bits in the system, etc. until you reach m_N). Once you have sequentially updated all N p-bits in the system, you have completed a single "Monte Carlo sweep", at which point you can take a sample of the state of the system, $\{m\}$, use that configuration to calculate a corresponding energy E.

This is referred to as a Markov Chain Monte Carlo (MCMC) algorithm. A Markov Chain is a system that changes its current state into a new state with a certain probability, and that probability is solely based on the current state (as opposed to a system with memory, that remembers previous states). "Monte Carlo" refers to the Monte Carlo casino in Monaco, alluding to the proabilistic nature of the algorithm.

We often refer to p-computing as "physics-inspired" computing. This is because we are taking physics models of natural phenomena like the Ising model and mapping practical problems to these "natural" computers. When p-bits are configured in the way I have described here, we refer to the network as an Ising machine.

5 A Concrete Example and the Boltzmann Distribution

Let's illustrate a > 2 p-bit system by example. Consider the 4 p-bit system shown in Fig. 6. This system can be represented by a symmetric adjacency

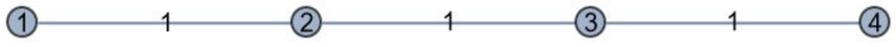


Figure 6: An arbitrary system of 4 p-bits.

matrix of weights J_{ij} and a column vector for the biases h_i:

$$J = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \mathbf{h} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

J_{ij} refers to the connection weight between the ith and the jth p-bit in the system. In matrix form, i is the row and j is the column. If, for example, the entry in the 2nd row and the 1st column is a 1, then it means the coupling between the 1st p-bit and the 2nd p-bit has a weight of 1. The matrix is symmetric, meaning $J_{ij} = J_{ji}$. As a simple example, $J_{12} = J_{21}$ - obviously, because the weight between the 1st and 2nd p-bit is the same as the weight between the 2nd and the 1st.

The h vector contains the biases on each individual p-bit. For example, the first entry of h is the bias h_1 on the first p-bit in the system.

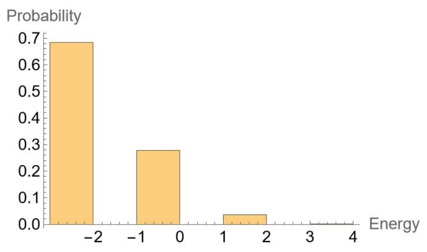
We just use matrices and vectors to make coding and calculations easier. Instead of using for loops and summations, we can instead use linear algebra. For example, $-\frac{1}{2}\sum_{i,j} J_{ij}m_i m_j - \sum_i h_i m_i = m^T Jm - h^T m$, where m is a column vector containing the spin states $\{m_1, m_2, \cdots, m_N\}$.

An interesting and useful feature of p-bit Ising machines is that their energy distribution converges to the Boltzmann distribution. The Boltzmann distribution is given by:

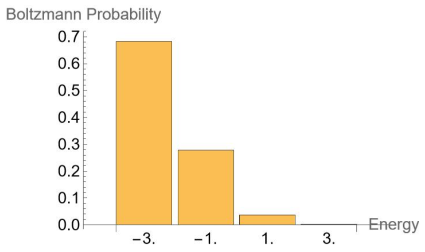
$$P(\{m\}) = \frac{1}{Z} e^{-E(\{m\})/T} \tag{8}$$

Where Z is the partition function (Z is a constant chosen such that probabilities of all spin configurations add to 1). If you simulate a p-bit network for a large enough number of Monte Carlo sweeps and repeatedly sample its energy, you will observe that the distribution of energies is a Boltzmann distribution. This is of particular practical value because problems such as integer factorization or circuit satisfiability can be mapped to Ising machines such that the lowest energy configuration corresponds to the correct solution - and the lowest energy configuration has the highest probability of being sampled.

Fig. 7 shows the results of the 4-bit system in Fig. 6 being run for 10000 Monte Carlo sweeps, sampling the Ising energy at each sweep, and plotting the frequency of each energy sampled as a probability. I recommend that you inspect the Mathematica code, understand it, and play with it (you may change the adjacency matrix, the biases, the temperature, etc).



(a) Experimental results from p-bit simulation of Monte Carlo sweeps at constant $T = 1$



(b) Analytical results from explicitly calculating the probability of each spin configuration. It's the same!

Figure 7: A system of 4 p-bits converging to the Boltzmann distribution

6 A practical optimization problem: Maximum Cut

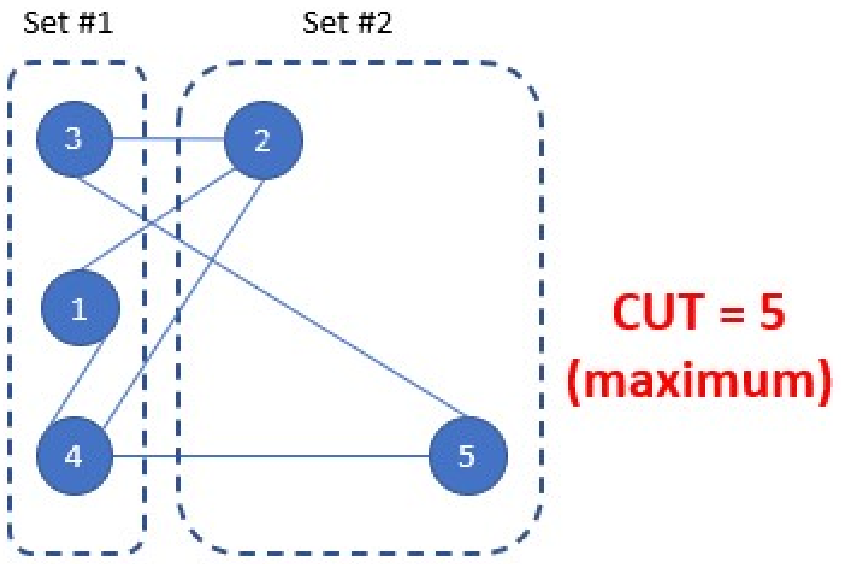


Figure 8: The maxcut problem.

We are finally ready to discuss a real application of a p-computer. Fig. 8 illustrates the Max-Cut problem. You have a graph with nodes and edges connecting the nodes. The goal is to find the way to divide the nodes into 2 groups such that there is the largest number of connections going in between the two groups (if you were to "cut" the edges like they're strings, we are interested in finding the way to group the nodes such that you cut the most strings). One real-world application of the Max-Cut problem is in portfolio management, where assets can be modeled as nodes and their correlations as edges; by maximizing the cut, one can help ensure that the portfolio avoids clustering highly correlated assets, thereby promoting diversification and potentially reducing risk. There are many more examples of real-world applications of the Max-Cut problem, and the Max-Cut problem is just one example of a challenging, NP-hard optimization problem that can be mapped to Ising machines.

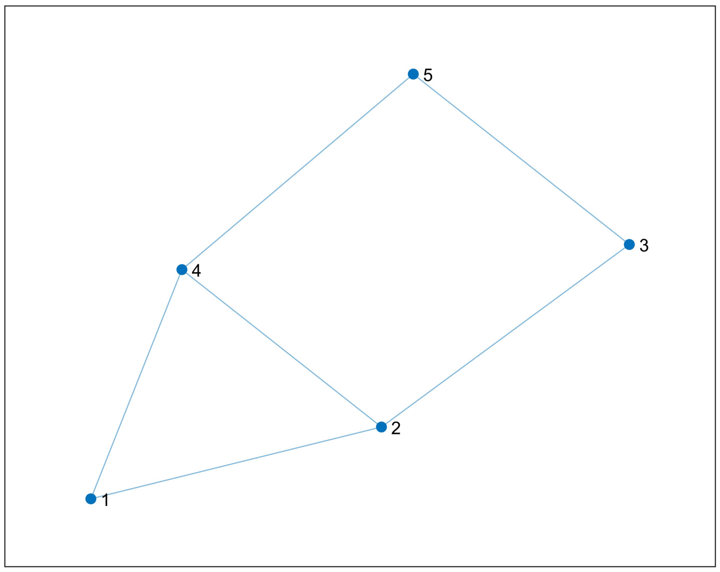


Figure 9: The maxcut problem mapped to p-bits. Each p-bit is a node, and connections have weight -1

The mapping of our Max-Cut problem to an Ising machine is straightforward. Fig 9 shows each node represented by a p-bit, and connections $J_{ij} = -1$ for connected nodes. It turns out that the lowest possible energy configuration of this p-bit system, which we call the ground state, solves our Max-Cut problem. In the ground state, p-bits will either occupy the +1 or -1 state: those in the +1 state are in one set, and those in the -1 state are in the other set. One way to find the ground state is to conduct Monte Carlo sweeps, sample the energies, and record the lowest energy you find. The Boltzmann distribution dictates that the ground state is the most likely configuration. For a small problem like this, it is not difficult to find the ground state in this fashion and solve the problem. However, this is difficult as the problem size is scaled up, particularly because of the existence of local minima and maxima.

Fig. 10 shows the Ising energies of 32 possible configurations of our 5-bit system. If we run Monte Carlo sweeps of p-bits, it is common to get "stuck" in a local minimum that is NOT the ground state energy or the global maximum which we desire. To get out of this local minimum, you might need to go to a higher energy configuration before going to a lower energy configuration. As an analogy, imagine a marble in a very shallow hole, and imagine that the small hole is adjacent to a very deep hole. Under the effect of gravity, the marble is going to stay in the small hole, when in reality, it could go deeper (lower potential energy) if it went UP in energy before going back down.

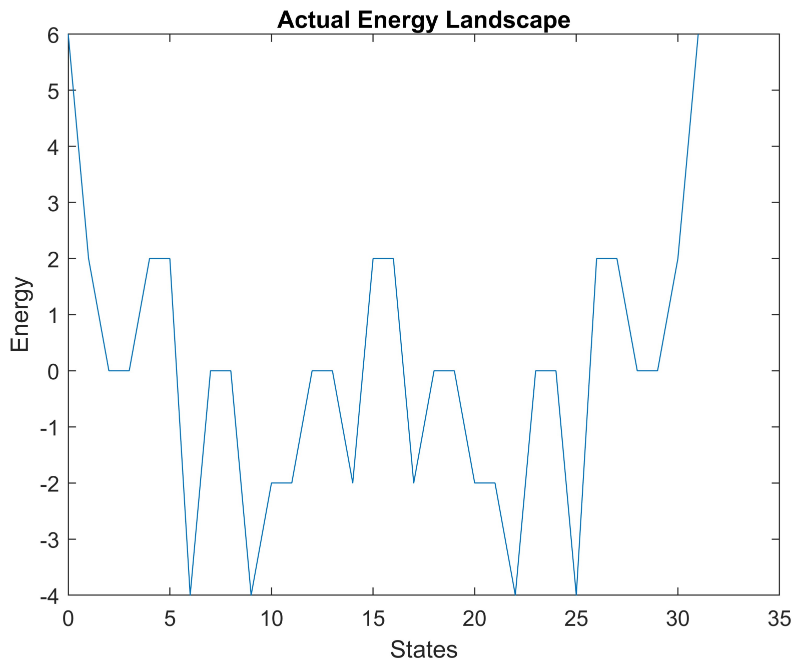
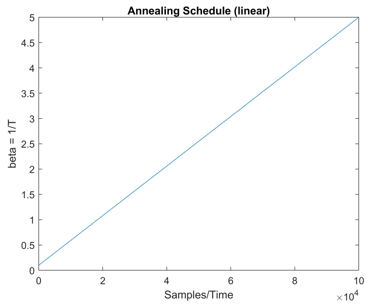
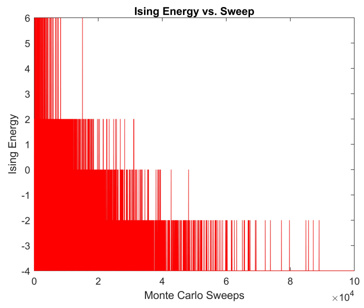


Figure 10: Energy landscape of our maxcut problem

Because of this issue, we typically don't solve optimization problems, where the goal is finding the ground state, by running Monte Carlo sweeps at constant temperature and sampling the system state (though in other contexts, sampling from the Boltzmann distribution is useful). We instead use more sophisticated algorithms. The algorithm I shall introduce here is simulated annealing. We start the system at a high temperature T (low inverse temperature β), and we slowly cool the system down. At high temperatures, the system is very likely to accept updates via the p-bit update rule (Eq. 2) that increase the overall system energy. Recall that at high temperatures, p-bit updates appear to be more noisy and closer to the 50/50 case of complete random noise. This noise allows the system to *explore* different system configurations without getting stuck in local minima. As the system is cooled, p-bits tend to flip less often/easily, and they have a much lower probability of accepting a flip that would increase the overall system energy. In optimization terms, this is *exploitation*, where we take the best point that we have already found in the state space/energy landscape and we search near that point for better solutions. Simulated annealing helps us avoid the issue of local minima, and it has been shown that an infinitely slow anneal would theoretically obtain the ground state. Of course, in real life, we cannot have an infinitely slow anneal, so we just conduct simulated annealing over as many Monte Carlo sweeps as is practical.



(a) Annealing schedule. Inverse temperature β is increased from 0 to 5 over the course of 10^5 sweeps.



(b) The sampled energy of the system over time. The lowest energy configuration, or the ground state, is -4. This configuration solves our max-cut problem.

Figure 11: Simulated Annealing

Fig. 11 shows simulated annealing in action. At the beginning, the system is at high temperature, so it explores a lot of energy configurations. As the system is cooled, it tends toward lower energy configurations, eventually finding our ground state and solving the problem.

I recommend that you understand and explore the MATLAB code provided to you. The annealing schedule does not need to be a linearly increasing β over time as shown here - for example, you could geometrically increase β, or you could linearly increase T.

7 Prof Camsari's Video Lectures and Slides

https://ucsb.app.box.com/s/n2p4qcfiog6zi3yoq9y4u82h3szdz1je

8 Recommended intro paper

“p-Bits for Probabilistic Spin Logic”, KY Camsari, BM Sutton, S Datta, Applied Physics Reviews 6, 2019