Carl Klier

## Detailed description of the problem

A valid k-coloring of a graph is defined to be a coloring of the nodes, using at most k colors, so that any pair of nodes sharing an edge have a different color. Finding a valid k-coloring is NP complete for k >= 3. There are algorithms that try and approximate valid k-colorings while running under a reasonable time constraint. One such algorithm utilizes Markov Chain Monte Carlo and the Boltzmann distribution. This involves setting the target distribution with a specific temperature T proportional to exp(-H(x)/T) and building a markov chain that converges to this distribution. To construct the target stationary distribution, we use the Metropolis Hastings algorithm which eliminates the need to know the normalizing constant. Using Glauber dynamics, we propose moving from a state X to state X' and accept that move with a certain probability. The move from X to X' is accepted with probability min{1, exp^(-[H(X') - H(X)]/T). This exponential is designed such that if the proposed state X' is "better", we accept the proposed state with probability 1. If the proposed state results in a higher energy value, we still accept it was some probability. Accepting a worse state with some probability helps prevent the chain from getting stuck in a local minima when we are trying to find the global minimum.

In this project, we want to examine how different temperatures and the number of iterations/steps of the chain affect the result of obtaining a valid k-coloring. We also examine solving the k-coloring problem on a quantum computer, specifically D-Wave's quantum annealing system and compare the number of reads/samples needed to ensure a valid k-coloring is consistently found.

## Scope of work

The scope of the work involves coding in python to implement the markov chain monte carlo for the Boltzmann distribution using Glauber dynamics. We run this algorithm on a graph starting at some initial coloring and then after a specified number of iterations, we examine the graph's coloring and verify whether the coloring is valid. We perform several modifications including varying the temperature and number of iterations to see their effect on the algorithm's ability to find a valid coloring.

The scope also involves implementing code in python to solve the graph coloring problem using quantum annealing on D-Wave's DW_2000Q_6 system. Once we receive a solution back from the quantum computer, we verify the result to see if it is indeed a valid coloring. We vary the number of samples/reads and examine the effect that has on the returned solution from the quantum computer.

In order to limit the scope of this project, we examine complete graphs. This is because the chromatic number of a complete graph is simply the number of vertices. Using the chromatic number, we know exactly how many colors are required, at a minimum, in order to color the graph in a valid way. Thus, we can set the k-value to the chromatic number. This allows us to easily see the difference between the approximate k-coloring using markov chain monte carlo and the real k-coloring that we would get by solving the graph coloring problem with brute force. In addition, this simplification allows us to easy verify if a coloring is a valid k-coloring by just making sure every node has a unique color.

In addition, to simplify the problem of solving the graph coloring problem on the quantum annealing machine. We let D-Wave handle mapping our variables to specific qubits. While an explicit embedding could possibly provide better results (due to some qubits having higher error rates), we don't worry about this potential source of error.

## Motivation

The graph coloring problem has many real-life applications that motivate researchers to find increasingly more efficient algorithms. One such motivation is the application of scheduling. In this problem, we want to assign a set of tasks to specific time slots where some tasks might conflict if they need the same shared resource. This problem maps to the graph coloring problem in the following way. The vertices of the graph are the jobs and an edge between two vertices represents that the two jobs require the same shared resource. The available colors are the available time slots for the jobs to be completed in. Thus, a valid coloring shows what jobs to complete in what timeslots so that no conflicts arise. If we are able to find the minimum k value such that a valid k-coloring exists, called the chromatic number, this is equivalent to the minimum number of timeslots needed to complete all jobs with no conflicts.

## Steps/Tasks

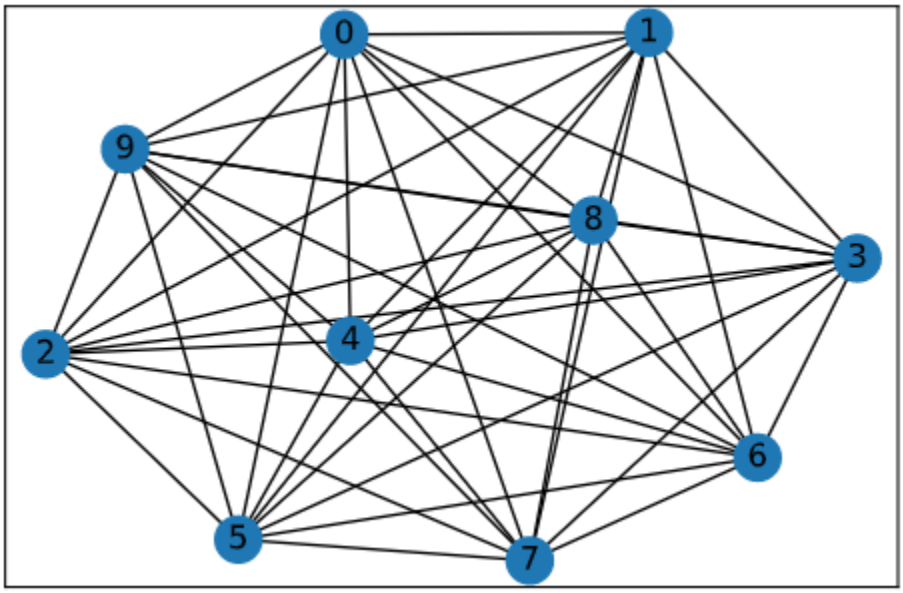
- For Markov Chain Monte Carlo with Boltzmann distribution
    - Create a complete graph on n vertices using python's network package
    - Create an array of n colors. The chromatic number for the complete graph on n vertices is n.
    - Implement a method to determine if every node has a unique color to determine if a valid k-coloring was found. This is essentially setting the k value to the chromatic number.
    - Create a Hamiltonian function to determine the energy value of the current state and the proposed next state. We want to ensure that a worse state has a higher energy value than a better state.
    - Implement the mokov chain using Glauber dynamics
    - Initialize the nodes to a an initial color state
        - initial color mapping: {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0}
        - all the nodes are initialized to the same color at the beginning
    - Run the markov chain for a specific number of iterations, at a specific temperature
    - Check and see if the coloring of the graph after the last iteration is a valid k-coloring
    - Compare and contrast the results of the colorings at the different temperatures and number of iterations.
- For quantum annealing using D-Wave's DW_2000Q_6 system
    - Create a complete graph on n vertices using python's network package
    - Set the k-value to n
    - Create the constraint satisfaction problem
        - There are two constraints to this problem. 1. A node should only be colored with a single color. 2. Two nodes that share an edge should not be colored the same color.
    - Convert the constraint satisfaction problem into the binary quadratic model
        - This is equivalent to the step of creating a Hamiltonian in the Markov Chain Monte Carlo program. Again, we ensure that states that get us

closer to a valid coloring have a lower energy value than states that take us further from a valid coloring

- We let D-Wave handle embedding our binary quadratic model. This is essentially a mapping of the variables of the binary quadratic model onto physical qubits of the D-Wave system.
- We then let the annealing process run. And we repeat the annealing process for a number of times. This number of times is called the number of samples or the number of reads. Since quantum computers are subject to errors due to the environment (including temperature), sometimes the system can jump from the ground state to a higher energy state. Thus, by taking multiple samples, we can get a distribution of the possible solutions and see that the solution with the highest count is most probably the solution to our constraint satisfaction problem.
- We vary the number of samples taken to see how consistently the lowest energy solution is a valid k-coloring and compare this to the number of iterations it took to get a valid solution when running markov chain monte carlo.

## Observations/findings/discussion

We examine a K10 graph, a compete graph on 10 vertices. Thus the chromatic number is 10 and we attempt to find a 10-coloring for this graph.




| | | | Iterations | | | | |
|---|---|---|---|---|---|---|---|
| | | 10 | 50 | 100 | 500 | 1000 | 10,000 |
| Temperature | .05 | 7 colors | Valid coloring | Valid coloring | Valid coloring | Valid coloring | Valid coloring |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | .1 | 7 colors | Valid coloring | Valid coloring | Valid coloring | Valid coloring | Valid coloring |
| | 1 | 7 colors | 5 colors | Valid coloring | 7 colors | 9 colors | 8 colors |
| | 5 | 7 colors | 5 colors | Valid coloring | 7 colors | 9 colors | 7 colors |

When it is not the case that a valid coloring is found, we report the number of unique colors that were used in the graph. This is in comparison with 10 colors that are used in a valid coloring.

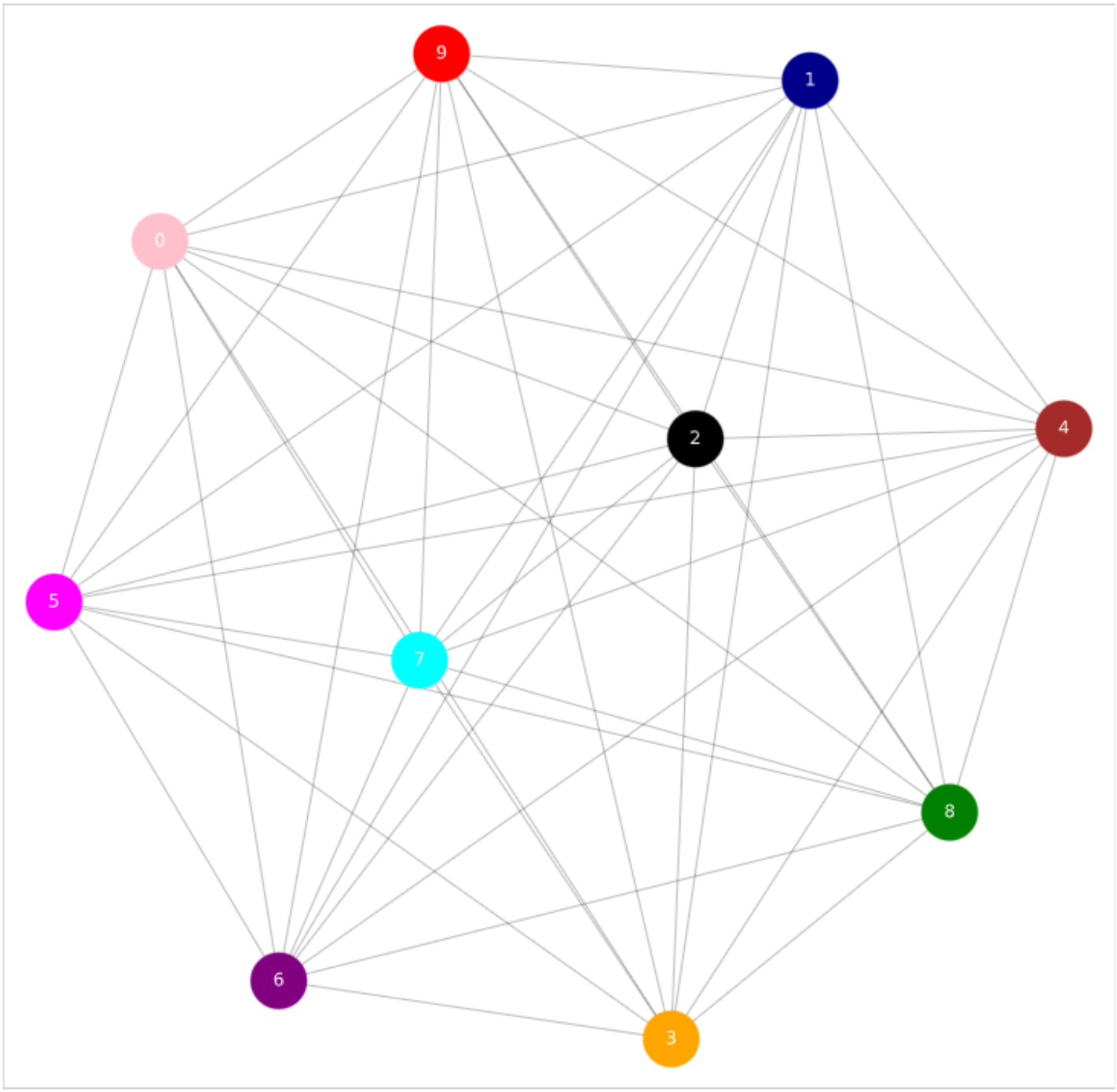



Figure 1: valid coloring found, T=.05, iterations=10,000

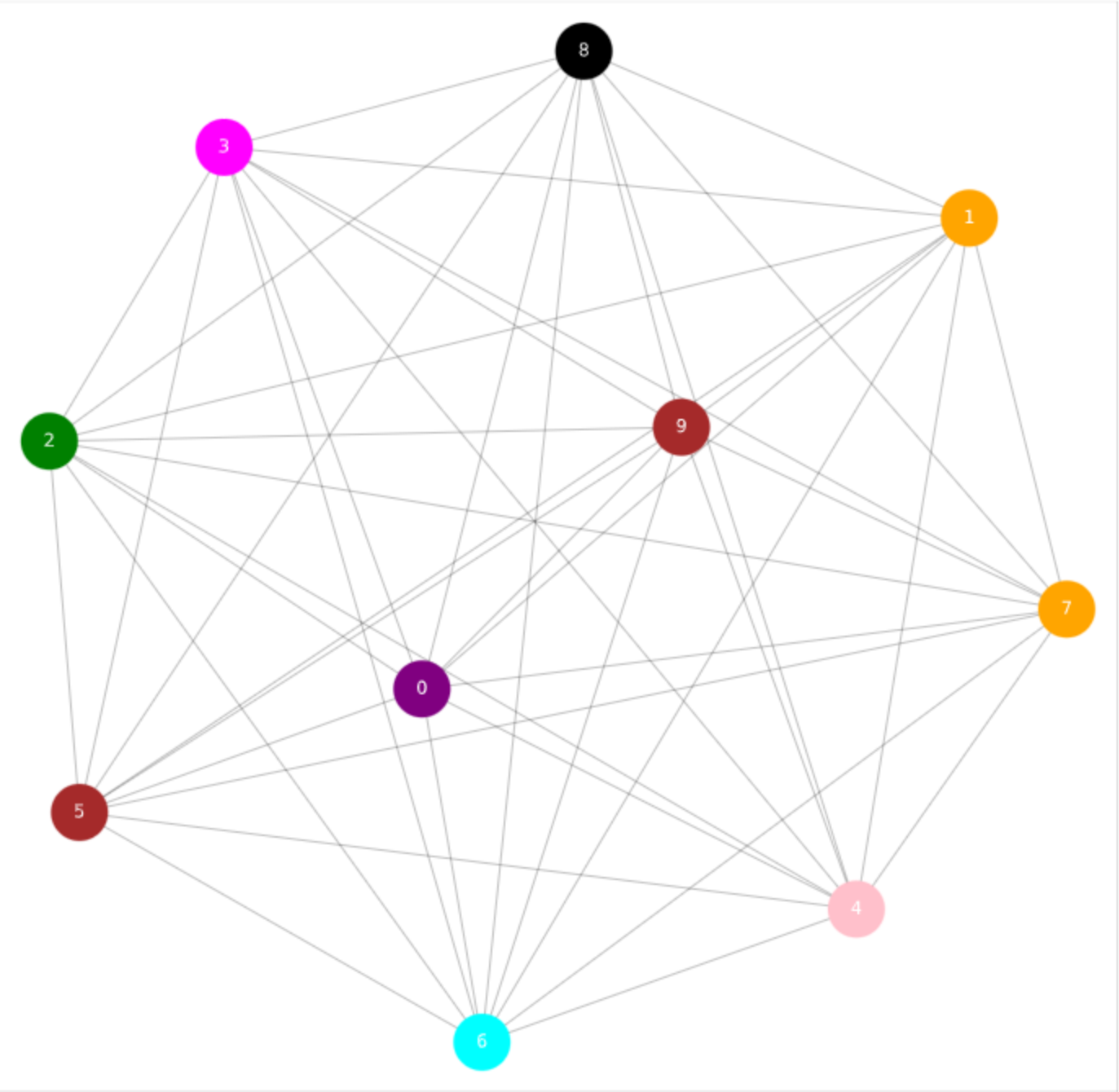



Figure 2: Invalid coloring with only 8 colors, T=1, iterations = 10,000

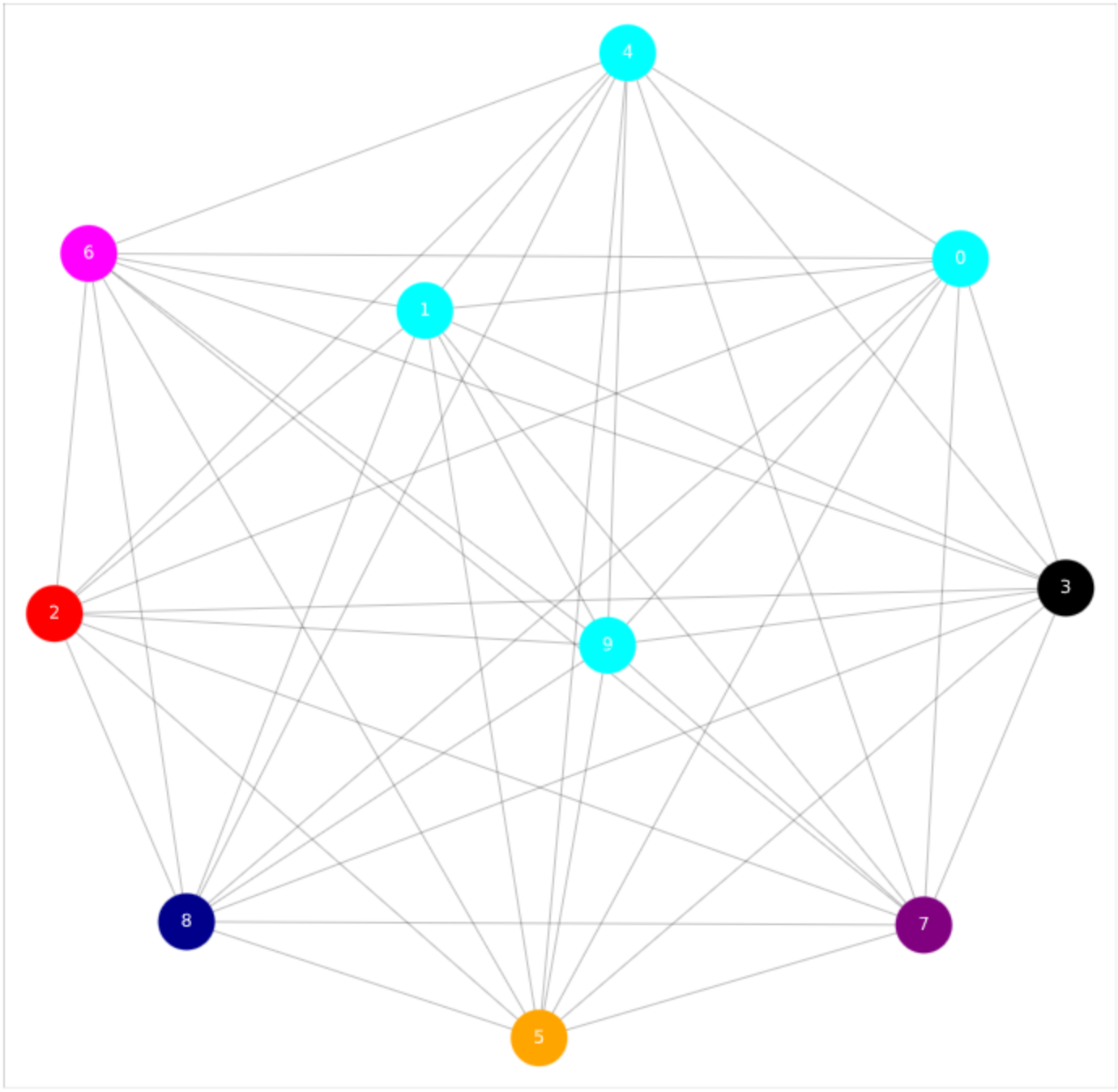



Figure 3: Invalid coloring with only 7 colors. T = .05, iterations =10

So as intuitive as it sounds, running the markov chain with a lower temperature for the Boltzmann distribution and running the chain for more iterations (helping it get closer to the steady state distribution, our target distribution for the Metropolis Hastings algorithm) least to a better approximation for finding a valid k-coloring. In the future, a better experiment would be to run this algorithm multiple times for each temperature and iteration combination across many different seeds in order to get a better understanding if the approximation finds a valid k-coloring

Unfortunately, on the quantum computer, I was unable to successfully implement the constraint satisfaction problem for the complete graph on 10 nodes. I was however able to implement the constraint satisfaction problem on a complete graph with 4 nodes. So, I report on some of those results below.

To interpret the results returned by the quantum computer, lets understand that the letters correspond to nodes and the numbers next to the letter correspond to a possible color out of the k colors. The color that is chosen for the node is marked with the value 1. In the sample below, a is colored with color 2, b is colored with color 1, etc. The reason the first sample is an invalid coloring is because no color at all was chosen for node d.

With 10 reads, we see two invalid colorings with energy around 2 and energy around 8:

Sample: {'a0': 0, 'a1': 0, 'a2': 1, 'a3': 0, 'b0': 0, 'b1': 1, 'b2': 0, 'b3': 0, 'c0': 1, 'c1': 0, 'c2': 0, 'c3': 0, 'd0': 0, 'd1': 0, 'd2': 0, 'd3': 0}

Energy Value: 2.0000002956522582

Failed to color graph validly

Sample: {'a0': 0, 'a1': 0, 'a2': 0, 'a3': 0, 'b0': 0, 'b1': 0, 'b2': 1, 'b3': 1, 'c0': 0, 'c1': 0, 'c2': 1, 'c3': 0, 'd0': 0, 'd1': 1, 'd2': 0, 'd3': 0}

Energy Value: 8.000000236184658

Failed to color graph validly

With just 50 reads, we get substantially better results. The first two lowest energy results are printed below:

Sample: {'a0': 0, 'a1': 0, 'a2': 0, 'a3': 1, 'b0': 0, 'b1': 0, 'b2': 1, 'b3': 0, 'c0': 0, 'c1': 1, 'c2': 0, 'c3': 0, 'd0': 1, 'd1': 0, 'd2': 0, 'd3': 0}

Energy Value: 2.3652181013744666e-07

Sample: {'a0': 0, 'a1': 0, 'a2': 0, 'a3': 1, 'b0': 0, 'b1': 1, 'b2': 0, 'b3': 0, 'c0': 0, 'c1': 0, 'c2': 1, 'c3': 0, 'd0': 1, 'd1': 0, 'd2': 0, 'd3': 0}

Energy Value: 2.3652181013744666e-07

We can see that both of these samples correspond to valid colorings. Thus, not only is the quantum computer capable of very quickly and consistently finding a valid k-coloring, it is actually very quick to find all valid colorings. With 50 reads taken, the first 5 lowest energy values all corresponded to unique valid 4-colorings