Graph Coloring Mappings:

Kyle Sokol

Brute-Force Algorithm: My brute-force algorithm is a back-tracking algorithm. This means it assigns each node a color one-by-one. Instead of checking the entire graph for validity, the program assumes the rest of the graph up to that point is correct and only checks nodes that could potentially be violated by giving the next node a color. If the next node can be assigned no valid color, then the program backtracks and gives the previous node a new color and tries again.

Runtime: O(m^V) (total color combinations) where m is the number of possible colors and V is the number of vertices. This is a more advanced brute-force algorithm, and its average runtime is faster than m^V.

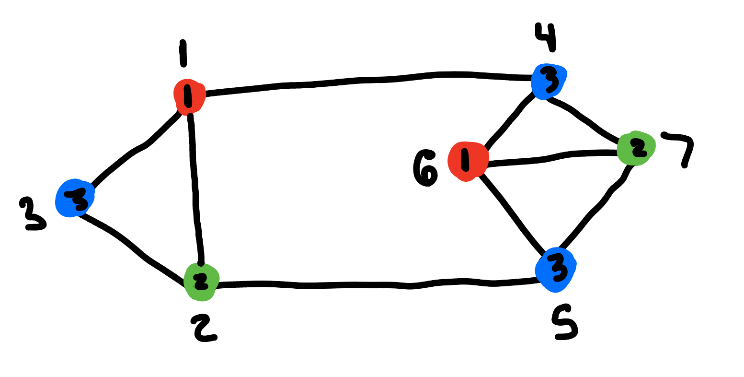
Heuristic: My heuristic gets its inspiration from the DSATUR graph coloring heuristic. The idea is similar to greedy coloring, but the nodes are colored in order of highest saturation. Saturation is determined by the number of colored neighbors a particular node has. In the case of a tie, the number of uncolored neighbors is used instead. Each node is then colored with the lowest possible color.

Runtime: O(n^2)

Below I have included two problems that I will reference for the rest of the paper.

**Hard Problem:** (heuristic fails to find the optimal solution)

Brute Force:

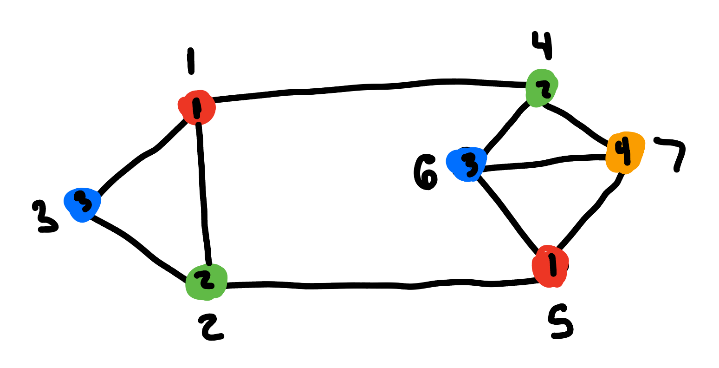


Nodes: 1 2 3 4 5 6 7

Colors: 1 2 3 3 3 1 2

Chromatic Number: 3

Heuristic:



Nodes: 1 2 3 4 5 6 7

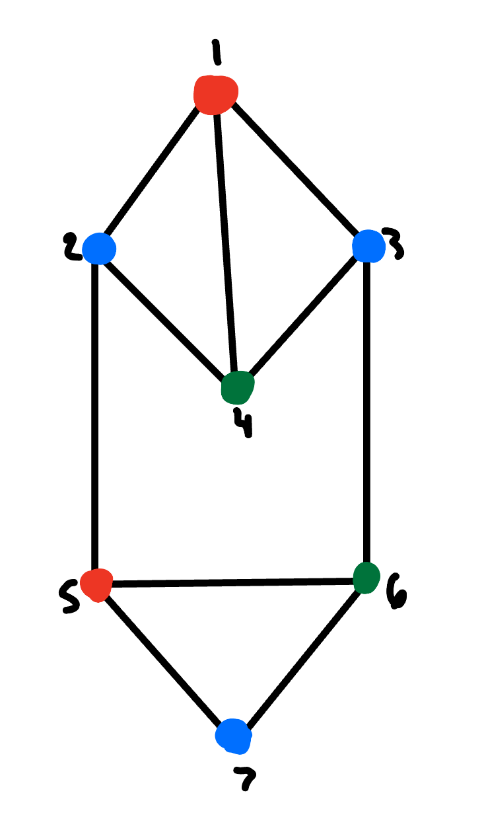
Colors: 1 2 3 2 1 3 4

Chromatic Number: 4

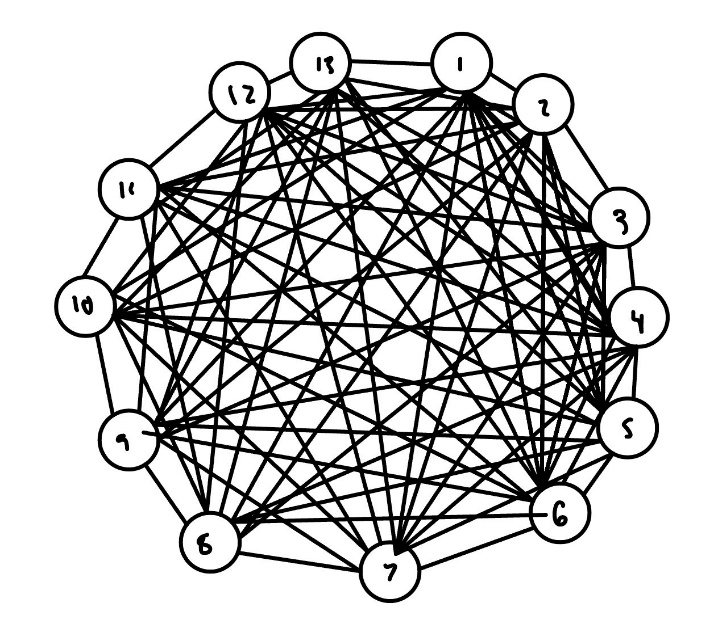
\*Note: As you can see the brute force algorithm finds the minimum coloring while the heuristic fails to do so.

Interesting findings: In my code, I use an unordered\_map. This is implemented differently in g++ and Microsoft C++. In g++, some problems can be solved that Microsoft C++ could not as the unordered\_map in g++ does not store items in the order they were inserted resulting in a degree of randomness which allows the g++ algorithm to get lucky.

Furthermore, rotating the graph and renumbering results in both the brute force and heuristic finding the optimal solution. This is because choosing a different starting point can affect the overall outcome.



**Intractable Problem (**brute force fails to find a solution in a reasonable time frame)



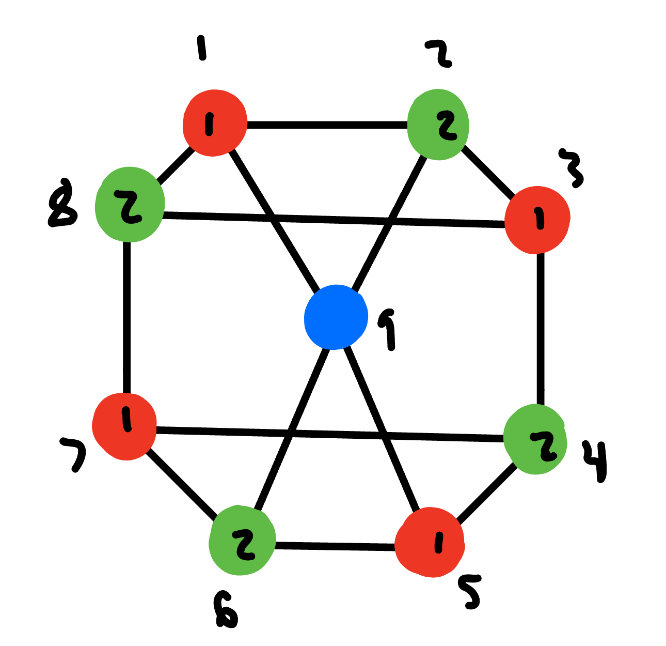
A 13-node fully connected graph is intractable for my brute-force algorithm. On a fully-connected 13-node graph, there are thousands of potential color combinations that brute-force can’t solve in a reasonable time frame. My heuristic solves the problem instantly. The solution is simple as all nodes are connected and must all have different colors.

**Mapping Input (Maximum Independent Set):**

Description: I chose to map an independent set problem to the graph coloring problem. Since no two adjacent nodes can have the same color, (nodes on the same edge must have different colors) this implies all color sets are independent sets.

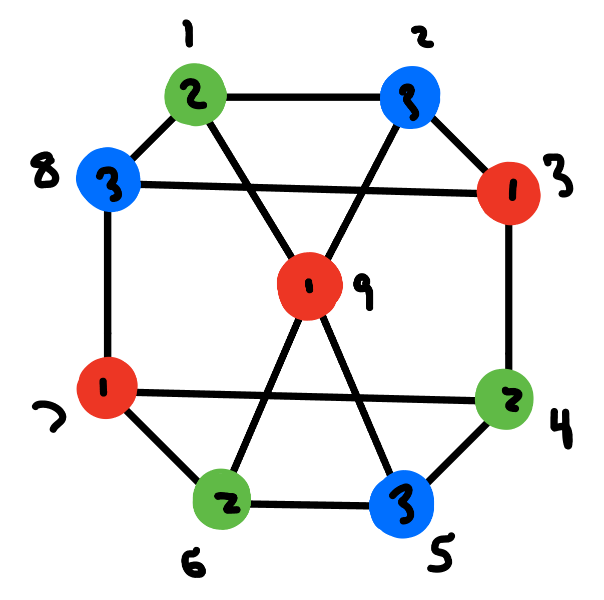
How the Mapping Works: Since both problems are graph problems, no modification of the input is needed. To find the largest independent set using the graph coloring algorithm, a helper function can be called to find the most common color and the corresponding nodes of that color after the colored graph has been generated. Each algorithm, brute-force and heuristic, will then display the results for what they found as the maximum independent set. In the case where there are multiple maximal independent sets, only one is displayed.

Findings: I was provided an independent set graph problem from Luke Lindsay that caused trouble for his heuristic. Running my brute-force algorithm produces an independent set of 4 nodes (green) with a chromatic number of 3. The coloring is shown below.



*Independent Set Brute Force*

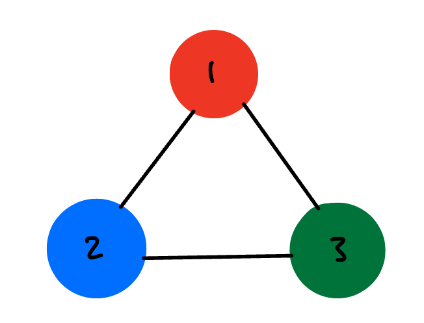
For my heuristic, it also produces the optimal chromatic number of 3 but only produces an independent set of size 3 (blue). This is interesting that both of our heuristics struggle to solve independent set, but my algorithm still solves the coloring problem with no issue. This is likely because the graph-coloring heuristic is designed to minimize colors and not maximize the independent set.



*Independent Set Heuristic*

**Mapping Output (3-SAT)**

Description**:** In order to map graph coloring to 3-SAT I needed to reduce my problem from a k-coloring to a 3-coloring problem. This mapping could have been generalized to K-SAT, but as there was no one in the class doing the more general problem, I was limited to 3-SAT. While k-coloring will always give a coloring, by limiting the problem to 3 colors, a coloring will only be produced if there is a valid coloring with only 3 colors. If more colors are needed, then there will be no result.



How the Mapping Works: We will choose three colors (red, green, and blue) to represent the possible colors that can be used. For each node, we will create three Boolean variables (iR, iB, iG) to indicate if the node is red, green, or blue. A value of 1 for the Boolean variable indicates that is the color of the node. To comply with the 3-SAT data file format, we cannot use variables like iR, iB, and iG. Instead, each node will have three numbers corresponding to it. For example, node 1 will have three Boolean variables (1, 2, 3) that correspond to (1iR, 2iB, 3iG), and node 2 would have Boolean variables (4,5,6). This process is continued for all nodes in the graph.

Now that we have the Boolean variables created, we need to create a satisfiability equation to satisfy the graph coloring problem requirements. To satisfy that no two adjacent nodes can have the same color, 3 equations are created for each edge. For example, if node 1 and node 2 are connected by an edge, the following equation would be produced: (-1 v -4 v -4) ^ (-2 v -5 v -5) ^ (-3 v -6 v -6). The first equation states that either node 1 or node 2 can’t be red. The second equation states that either node 1 or node 2 can’t be blue. To better explain, if 1 and 4 were true (they were both red) the equation would fail. The intersection of these three equations ensures that nodes 1 and 2 have different colors. The reason that the last value is repeated in each equation is to satisfy the 3-term requirement for 3-SAT. Each equation could be written without the last term, but its addition does not change the problem. Furthermore, this equation only needs to be produced for each edge once.

To ensure that all nodes are either red, green, or blue, for each node the following equation is built: Node 1: (1 v 2 v 3) Node 2: (4 v 5 v 6).

The 3-SAT equation for the above example would be as follows:

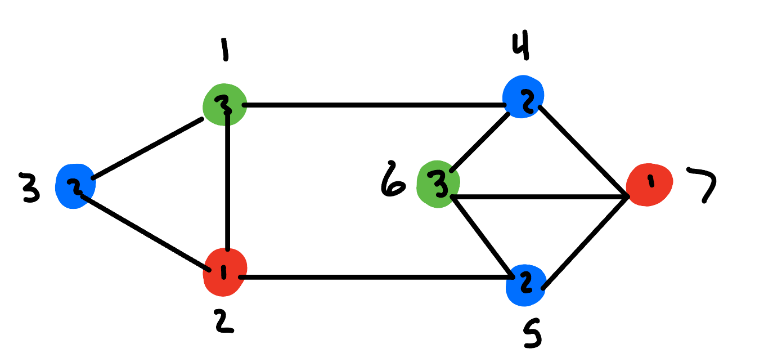
(-1 v -4 v -4) ^ (-2 v -5 v -5) ^ (-3 v -6 v -6) ^ (-1 v -7 v -7) ^ (-2 v -8 v -8) ^ (-3 v -9 v -9) ^ (1 v 2 v 3) ^ (-4 v -7 v -7) ^ (-5 v -8 v -8) ^ (-6 v -9 v -9) ^ (4 v 5 v 6) ^ (7 v 8 v 9)

Findings:

I was able to share my hard and intractable problems with Will Miller who was doing 3-SAT.

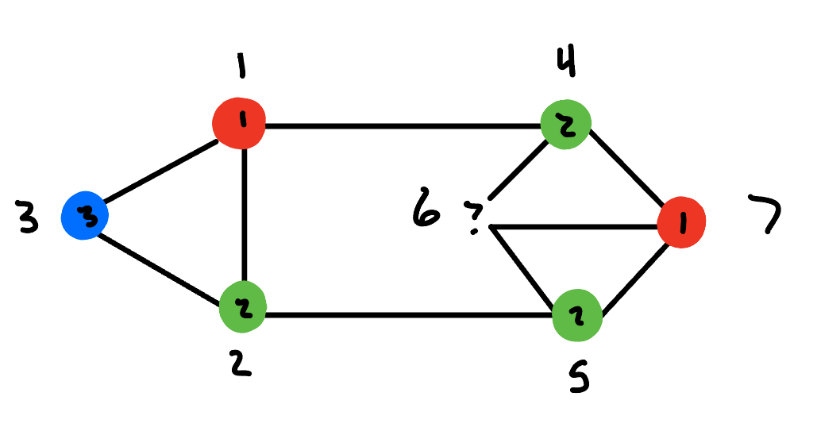
*Hard Problem:*

Will’s brute-force solution concluded that terms 2,4,9,12,15,17,19 in the 3-SAT formula were true which produces the following coloring.



*3-SAT Brute-Force*

For the heuristic, terms 1, 5, 9, 11, 14, 19 were true producing the following coloring



*3-SAT Heuristic*

As we can see, the brute-force algorithm was able to find a valid coloring although different from the brute-force written specially for the graph-coloring algorithm. On the other hand, the heuristic was incapable of finding any valid coloring. Both algorithms ran in less than a second (similar to the algorithms specifically for graph-coloring). As we can see, the graph coloring heuristic finds a solution (but not optimal) where the 3-SAT fails to find a solution. This is likely due to the fact that my graph coloring algorithm has access to k-colors whereas 3-sat only has access to 3.

*Intractable Problem:*

Will’s brute-force algorithm was unable to provide a coloring after running for 10 minutes which is the same outcome as my brute-force algorithm. Since the completely connected 13-node graph requires 13 colors, a 3-SAT algorithm (restricted to 3 colors) will never find a solution. Will’s heuristic finished in a timely manner but did not provide a valid result. Most nodes were left uncolored, and a few nodes were either red, green, or blue.