**Graph Coloring**

Author: Kyle Sokol

Languages Used: C++

Description: This program computes the minimum number of colors needed (chromatic number) to ensure that any two nodes on the same edge of a graph have different colors. The graph coloring problem has numerous applications, most famously with the map coloring problem where the minimum number of colors are used to color a map.

Mappings (Reductions): My program supports mapping from independent set and to 3-SAT

* Independent Set: My program takes in a graph used for an independent set, runs the graph coloring algorithms on it, then outputs the maximum independent set of the original graph. Each color set in the graph coloring problem is an independent set.
* 3-SAT: To map to 3-SAT, I had to reduce my problem from k-coloring to 3-coloring. My program prints to the console the 3-SAT representation of my input graph.

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)

How to Use:

**Compile:** g++ main.cpp parser.cpp algorithms.cpp -std=c++14

**Run:** ./a.out <graph\_file\_name>

Output: All output is displayed to the console. The 3-SAT format is printed first followed by the results for the heuristic and brute-force algorithms. For each algorithm, the maximum independent set is listed below it.

Limitation: If the heuristic and brute force don’t run there may be an incompatibility with line endings.

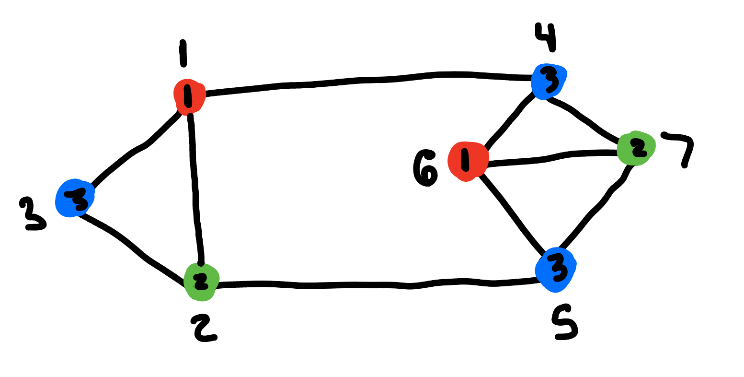
File Structure:

* Main.cpp (main program that calls the various algorithms)
* Parser.cpp/h (parses the graph file)
* Algorithms.cpp/h (contains the brute force and heuristic algorithms)
* Intractable.dat (graph that can’t be solved reasonably by the brute-force)
* Interesting.dat (graph heuristic gets wrong but brute force gets right)
* Graphs (directory that contains graphs I used for testing. The certificate is listed at the bottom of each file)

Note: I included a second brute force algorithm in my code but it is never called.

Results of Hard/Interesting problem:

Brute Force:



Nodes: 1 2 3 4 5 6 7

Colors: 1 2 3 3 3 1 2

Chromatic Number: 3

\*Heuristic on next page

Heuristic:

A picture containing indoor, attached

Description automatically generated

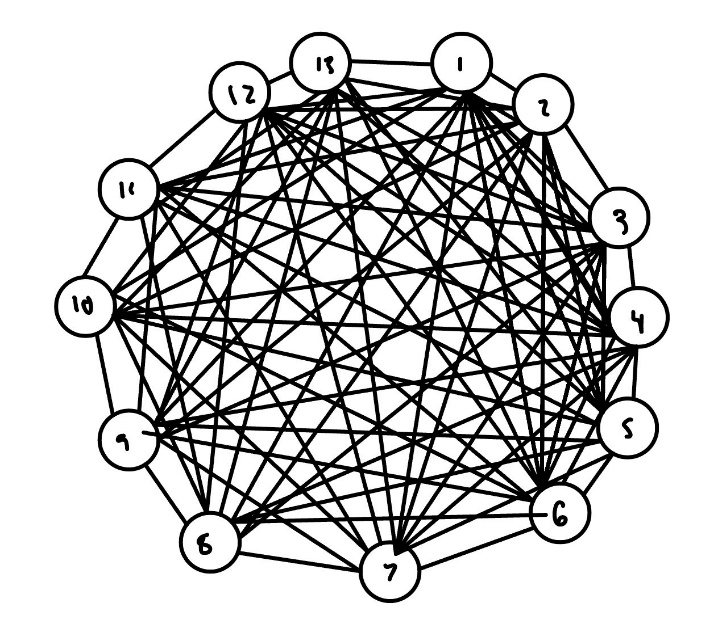
Nodes: 1 2 3 4 5 6 7

Colors: 1 2 3 2 1 3 4

Chromatic Number: 4

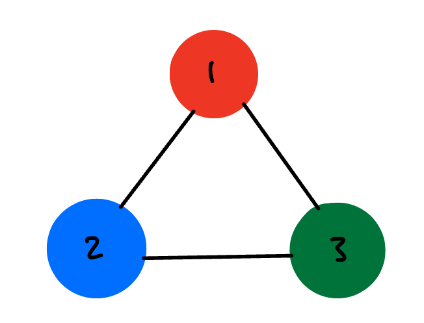
\*Note: As you can see the brute force algorithm finds the minimum coloring while the heuristic fails to do so.

Intractable Problem Results:



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.

3-SAT Output Example:



(-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)

Note: An in-depth explanation of how the 3-SAT mapping works is in my full paper.