# CS 6511 Project 2: Graph Coloring Problem

## Introduction

For this project, we were asked to implement a constraint satisfaction problem solver for the Graph Coloring Problem. The code for this assignment is available on GitHub at the following link: <https://github.com/jesserobles/6511_project2>. The code was developed using the algorithms and pseudocode given in the course lecture slides as well as from the material found in (Russell, Norvig, & Chang, 2021). The solver implementation includes a backtracking search algorithm, variable and value ordering heuristics (mrv, lcv), and two inference methods (forward checking, maintaining arc consistency with ac3). These are described in the following, as well as being documented in the code repository. Several combinations of these were tested. Running backtracking search without inference or variable and value ordering heuristics can solve the basic test cases but did not fare so well with more complicated problems, taking too long or exceeding recursion depth limits and running out of memory. The largest improvement in performance resulted from adding the minimum remaining values heuristic. This allowed the backtracking to solve all but one of the test cases very quickly. The largest test file proved intractable unless inference is used, resulting in exceeding recursion and memory limits. Adding lcv and incorporating inference by maintaining arc consistency allowed this large problem to be executed, although it ran for almost 2 hours. The problem appears to have no solution based on the number of colors given. The defaults used in the main execution script are therefore backtracking using mrv, lcv, and maintaining arc consistency using ac3.

## Running the main.py Script and Unit Tests

To run the code on an input file, in the command line run python main.py filepath from the same directory as the main.py file, replacing filepath with the relative path to the input file (e.g., assets/input\_files/australia.txt). Running the command with \* as the filepath argument (python main.py \*) will run the graph coloring CSP backtracking on all but the largest input file and print out the solution and elapsed time. You can also pass additional arguments for variable and value ordering and inference method, as described in the help:



For example, you can solve an input problem with forward checking by running:

python main.py assets/input\_file/gc\_78317100510400.txt -inf fc

To run unit tests, run python unittests.py. This will simply print out how many tests it ran, and OK if all tests passed, and FAILURE and cause of the failure(s) otherwise. The unit tests test out the different variable and value ordering heuristics (mrv, and lcv), inference methods (forward checking, and maintaining arc consistency with ac3), and backtracking search with and without inference.

GraphColoringCSP Class

The graphcoloring.py module contains the GraphColoring CSP class, which represents the graph coloring constraint satisfaction problem. It also contains the GraphColoringConstraint class that implements the actual constraint between two neighbors. The GraphColoringCSP has several attributes stored for convenience including neighbors and constraints. It also has utility methods to read in the input files, count any conflicts for an assignment, apply and remove inferences to the domains, and several other helper methods.

## Search Algorithm to Solve the CSP – Backtracking Search

For this problem, my implementation of the backtracking search algorithm is very similar to the pseudocode given in the slides and text. These functions are available in the backtracking.py module. I also implemented a slightly simpler version of the backtracking search algorithm in the same module that does not use inference (forward checking or maintaining arc consistency) to benchmark the improvements provided by the inference. Both implementations take variable and value heuristic functions to be able to gage the impact of these on performance.

## Heuristics

### Variable Ordering Heuristic – Minimum Remaining Values

Heuristics are implemented in the heuristics.py module. The function in this module implements static ordering, which simply returns the next unassigned variable. This is used as a baseline to see what improvements the LCV function provides. The second function in the heuristics.py module is mrv, which implements the minimum remaining values heuristics with tie breaking. This function loops through unassigned variables collects tuples containing the variable, the length of the domain for that variable, and how many constraints that variable is subject to. The list is then sorted on the last two values, ascending domain length and descending constraint count.

### Value Ordering Heuristic – Least Constraining Value

The third function in heuristics.py is the unordered domain values. This simple returns the domains for a given variable in the default order. This function is also used for benchmarks. The final function in the module is the lcv function, which implements the least constraining value heuristic. This simply orders the domain values based on the number of conflicts the values create, calculated with the GraphColoringCSP.count\_conflicts method. This is returned in ascending order.

## Inference Methods

### Forward Checking

The inference methods are implemented in the inference.py module. The first inference method is the forward checking function. This function takes in a GraphColoringCSP object, a variable, and an assignment. It loops through the unassigned neighbors of the variable, then through that neighbor’s domain values and if the assignment is inconsistent with the value, it marks it for removal. If after reviewing the neighbor’s values there aren’t any values left, the function returns a failure since the solution would be invalid. If there is not failure, the function returns the domain values marked for removal.

### Constraint Propagation using AC3

The ac3 and revise methods are implemented next in the inference.py module. The ac3 and revise function are implemented almost identically to the pseudocode from the slides. It takes in a GraphColoringCSP object and queue of arcs. It pops an arc from the queue and applies the revise method, which returns any domain values that should be deleted. It checks if the removals so far would result in any empty domains and returns failure if so. Otherwise, it adds the neighbors of the current variable to the queue. The revise method takes a GraphColoringCSP object and the vertices comprising an edge. It loops through the first vertex’s domains and checks if any value in the second vertex’s domain can provide a consistent solution. If not, it marks the first vertex’s domain value for removal. The actual inference method that gets called from the backtracking search is the maintain arc consistency function, which in turn calls the ac3 on the arcs comprised of the neighbors of the current variable.

# Bibliography

Russell, S. J., Norvig, P., & Chang, M.-W. (2021). *Artificial Intelligence: A Modern Approach.* Hoboken, NJ: Pearson.