**REPORT**

**Project 3: Constraint satisfaction problems (CSP)**

**(Map Coloring)**

**Submitted By**

Jawad Chowdhury, (ID# 801135477)

**Map Coloring Problem Formulation**

Here, in this project, we have worked on the implementation of a Map Coloring Problem. To solve this problem, we have used Constraint Satisfaction approach with different strategy and heuristics.

In a general sense, a constraint satisfaction problem is a scenario where we have a set of variables (nodes) and each of them has a certain domain (possible values to assign) and a set of constraints. Our goal is to find out an allowable combination (satisfying the constraint) of variables and with the values.

**Map Coloring Problem**: The map coloring is a problem where there are different areas/regions in a map and taking a bunch of colors we need to color each of these regions in such a way so that no two neighboring regions has the same color.

**Constraint Graph**: we can formulate the map coloring problem in a constraint graph, and as each of the constraint relates only two of the regions (variables), so we have a bunch of binary constraints (edges) to satisfy.

In this project we have used different strategies to solve the problem, such as-

**DFS** – This is the basic strategy, it does not propagate the process for any further pruning. The main check for this strategy is to check whether the current assigned value to the current variable does not violate any past assignments.

**DFS + Forward Check** – This is a strategy which is based on the DFS but here there are additional checks. After assigning the value, according to this strategy, we eliminate any domain value to the unassigned variables that is not consistent with the current assignment.

**DFS + Forward Check + Singleton** – In case of DFS + Forward Check, if any neighbor has a domain with only one value, then that neighbor needs to be propagated further. And in case of the propagation, if we locate the similar case again, the singleton (domain with single value) propagation continues further.

**Program Structure (Variables, Functions-Procedures & Heuristics)**

In my implementation, I have used 6 separate scripts to isolate the implementations. **1.csp\_dfs.py** : for basic DFS strategy

**2.csp\_dfs\_fs.py** : for DFS + Forward Search (Forward Check)

**3.csp\_dfs\_singleton.py** : for DFS + Forward Search (Forward Check) + propagation through singleton domain.

**4.csp\_heuristic\_dfs.py** : for basic DFS strategy with heuristics

**5.csp\_heuristic\_dfs\_fs.py** : for DFS + Forward Search (Forward Check) with heuristics

**6.csp\_heuristic\_dfs\_singleton.py** : for DFS + Forward Search (Forward Check) + propagation through singleton domain with heuristics

**Functions-Procedures**: The mentionable functions-procedures that I have used in this implementation are as follows-

**1.check\_constraint(n,c)** - to check whether the assignment of color ‘c’ on node ‘n’ violates any assignment or not.

**2.get\_next\_with\_heuristic() –** this is an utility function to find out the best next possible variable to use according to the heuristics of MRV and Degree heuristic.

**3.get\_color\_with\_heuristic(n) –** this function returns the best possible color for the node ‘n’ according to the heuristic of Least Constraining Value.

**4.get\_removed\_domain(n,c) -**  for the assignment of color ‘c’ to the node ‘n’, this function eliminates the inconsistent values from the neighboring variables.

**5.color\_map(i,n)**- this is actually kind of the run function which executes recursively until it reaches to a base case.

**6.check\_map\_color()-** this function tries to satisfy the constraint problem using different number of colors starting from 1 to certain max limit. Whenever this function gets a success, we get the chromatic number χ(G), which is the smallest number of colors needed to color G.

**Heuristics:** Here I have used 3 different heuristic to formulate the preferred order of values and variables.

**MRV & Degree Heuristics:** These 2 heuristics are being used in the function **get\_next\_with\_heuristic().** So among the unassigned variables we order them first according to Minimum Remaining Value heuristic. Then among those ordered variables we use the Degree Heuristic as the second priority. MRV gives us the variables with Minimum possible domain value and Degree Heuristic gives us Variables which are mostly constrained.

**LCV (Least Constraining Value):** I have used this heuristic in the function **get\_color\_with\_heuristic(n).** So the strategy is to pick up the value from the domain which rules out the fewest possible value from the neighboring variables.

**3. Execution Results (According to the Specification)**

A. Implement Hill climbing search

B. Hill-climbing search with sideways move

C. Random-restart hill-climbing search

**4. Source Code**

I have 4 separated scripts for different variants (1 for basic, 1 for sideways, 1 for random restart with sideways, 1 for random restart without sideways)

===========================================================

*"""  
 Implementation of hill climbing search and its variants.  
 author: Jawad Chowdhury.  
"""***import** random, copy  
  
**class** Board:  
 *"""  
 This class maintains properties related to the different state of the N-Queen Problem.  
 """*