Christopher Wang
Project #2 – CSP Color Graph
GW ID: G34579007
https://github.com/Wchris95/Project_2_CSP_GC_CW

Introduction

The code base above resolves finding a solution to several of the example test files given within the csp color repository here: https://github.com/amrinderarora/ai/tree/master/src/main/resources/csp/coloring. The code base attempts to implement different methods to obtain assign different vertices different colors while avoiding assigning a vertex a color that is the same as one vertex that is adjacent to it. This requires using a backtracking search algorithm, variable and value ordering heuristics (mrv & lcv) and inference methods that track the arcs within the problem to navigate the different changing restraints as the problem dives into assigning variables colors. The code based is mainly based off of the repository found based on the book “Artificial Intelligence: A Modern Approach” (Russell And Norvig)

Initial work

Within the start of the development creating a backtrack search without implementing any value/variable ordering heuristics or inference methods was able to resolve a majority of the test problems, however as the problems grew bigger in constraints and variable size the search became increasingly difficult to run eventually leading to hitting the recursion depth limit. After running this applying variable and value ordering heuristics vastly improved on our search time except the largest 88.7 mb. Currently with the current code base in the repository it is unable to resolve the largest file as the inference implementation has a bug for both inference methods of maintaining arc consistency with arc3 and the forward checking. Both methods are still able to solve problems if the solution is possible, but has issues stopping once our variables no longer have viable domains. The code gathers together within the main.py file which allows the user to designate what variable + value heuristic should be applied and same for the inference methods. The defaults set for arguments to solve are no inference method, lcv, and mrv applied. Once the arguments are set the code goes through several of the test files outputting the solution for each file.

Script info and Unit tests

The main.py script utilizes the argparse module to collect any changes the user may want to change regarding the value/variable heuristics or inference method. An example of the running the main.py script is below:

```
(base) chris@Chris:~/Documents/GitHub/Project_2_CSP_GC_CW$ python main.py -inf forward_check
 -value lcv -variable mrv
```

After running the main.py script the code will output the solutions it’s found with the time of how long the script took to search.

The script is composed of several modules to implement the different heuristics and inference methods also with a script to import the data from a text file and run through several unit tests to check if the modules have failed. The unit tests all mainly pass except the maintenance arc consistency, however I tested this case directly into the search module and appears with a solution meaning some where between the solution and unitTest module the solution is being overwritten.

FileParser.py

The file parser script is a simple script that takes in the filename to open the text file where it will go through every line in the file first checking if the line being read is empty or starts with a '#' if this is the case we continue through the file. If the line starts with colors that is where it'll extract our color information needed to feed into our color csp class. If any of these if statements are not satisfied we set a temporary variable to try to convert our 2 vertices to integers sort them and set them as a tuple to append to our edges object. If this fails we can assume the line we are at is not where our vertices are and can continue through the text file. Once we obtain the edges the script goes through each edge to find the neighbors for each vertex and the unique vertices. After going through the text and assigning our objects the info we obtain is placed into a data dictionary to be given to our coloring csp class.

ColoringCsp class

The coloringCsp class is our main class containing both the coloring csp class and our constraint class. This class holds and is our main object as the search is going containing information such as our vertices, the current domains we have set and been removed, constraints, and others. It also contains several necessary functions such as pruning, checking if constraints are consistent, applying the removals, and such. This class is mainly based off of the AIMA repository using the repository to get a better grasp of what the class needs to store as the search and methods are being implemented

Search Algorithm

The search algorithm has been separated into 2 parts one with the implementation of inference methods + heuristics and the other being a more simplistic basic search with heuristics being applied if necessary. The inference search algorithm is based on both the psuedocode and the AIMA repository's backtrack search algorithm.

Inference Methods – Forward Checking

The inference.py module begins with the forward_check function taking in the csp, variable, value, assignment, and removals. It will loop through the neighbors of the variable that inputted checking if that neighbor is already within our assignment, if not we need to check if the current domain value is consistent. These 2 variables with their values are evaluated if they are consistent with our constraints if it not we remove them from our possibilities. We want to also check if the neighbor domain is empty this is another check to see if the neighbor variable+value can be part of the assignment.

Inference Methods – Maintain Arc Consistency

The maintiain_arc_cons function incorporates both ac3 and revise function where it will structure the queue before handing it off the ac3 function. The queue consists of a pair of the variable and it's neighboring variables which symbolizes our arcs. This is then handed off the ac3 where it will go through the queue popping off arcs to hand off to the revise function and checks if the values and variables fit the constraints where if Xi=x is conflicting with every possible y in Xj=y then eliminate Xi=x. Once the revise evaluates the variable+values ac3 checks if the revised boolean is true we have domain values that need to be added to our removals to remove from the current domains. We then check if the first vertex does not have a empty domain within the csp.curr_domains if so we need to

return False for the search to unassign the variable from our assignment. We then loop through the neighbors of our first vertex and check that the second vertex from the queue is not a neighboring vertex from our first vertex.

Heuristics – Minimum Remaining Values (MRV)

Within our heuristics module we have several heuristics starting off with the original value order where we just grab the first vertex from the csp.variables as long as the variable is not within our assignment. Then we have the minimum remaining value(mrv) heuristic which has initially starts with an empty list to store variable ordering. We then for loop through the variables from csp.variables that are not in our assignments to calculate the domain count, the count of our constraints, and then append a list(variable, domain count, constraint count) of what we calculated to the temp list we initialized. After the for loop has gone through we sort the temp variable the domain count and then the constraint count. The second sort initially sorts the constraint count then reverses the assignment.

Heuristics – Least Constraining Values

The domain value ordering starts with just an unordered function just returning the domain values as they are originally. We then have the least constraining values that utilizes our counting of conflicts function within our coloring csp class. This will return our domain values sorted by the amount of conflicts each value has where the least constrained will be the start and most constrained at the end.