Hunter Hawkins
CS-470 Artificial Intelligence
Dr. Soule Summer 2020
Project 3 CSP

Summary and Algorithm Descriptions

So in my constraint satisfaction problem of map coloring I chose for my first algorithm to use a depth first search (DFS) with the degree heuristic. This heuristic is finding the maximum constrained variables and assigning them values first. This technique seemed to be efficient in terms of space and time complexity and it showed in the results. The results seemed to be accurate and fast everytime I ran the program. As for the second algorithm I chose was a hill climbing algorithm. This algorithm creates a random initial solution, and a temporary solution. It then compares the conflicts of these solutions and makes the current solution the one with the least amount of conflicts. It starts off by just using two colors in an attempt to find a solution and then adds another color into the mix at the user's desired time in seconds. Generally the longer it searches for the less colors it needs to use, and if it searches for a shorter time it generally results in more colors being chosen. This is what you would expect when increasing the branching factor for a hill climbing algorithm. Both algorithms are consistent at finding a solution, but generally the hill climbing algorithm is much slower than the DFS. I was struggling to read in the excel file with python so when attending office hours I was told I was able to slightly modify the file to help ease the process of reading it, so I modified it slightly and will attach it with the submission.

DFS Basic Logic:

#Step 1: Look for region with most connections, and assign it a color

#Step 2: Keep performing step 1 until there is an issue than assign it a new color

#Step 3: Once all regions are assigned a color return True, otherwise return False

Hill Climbing Basic Logic:

#Step 1: Create an initial state and value it. If goal state break

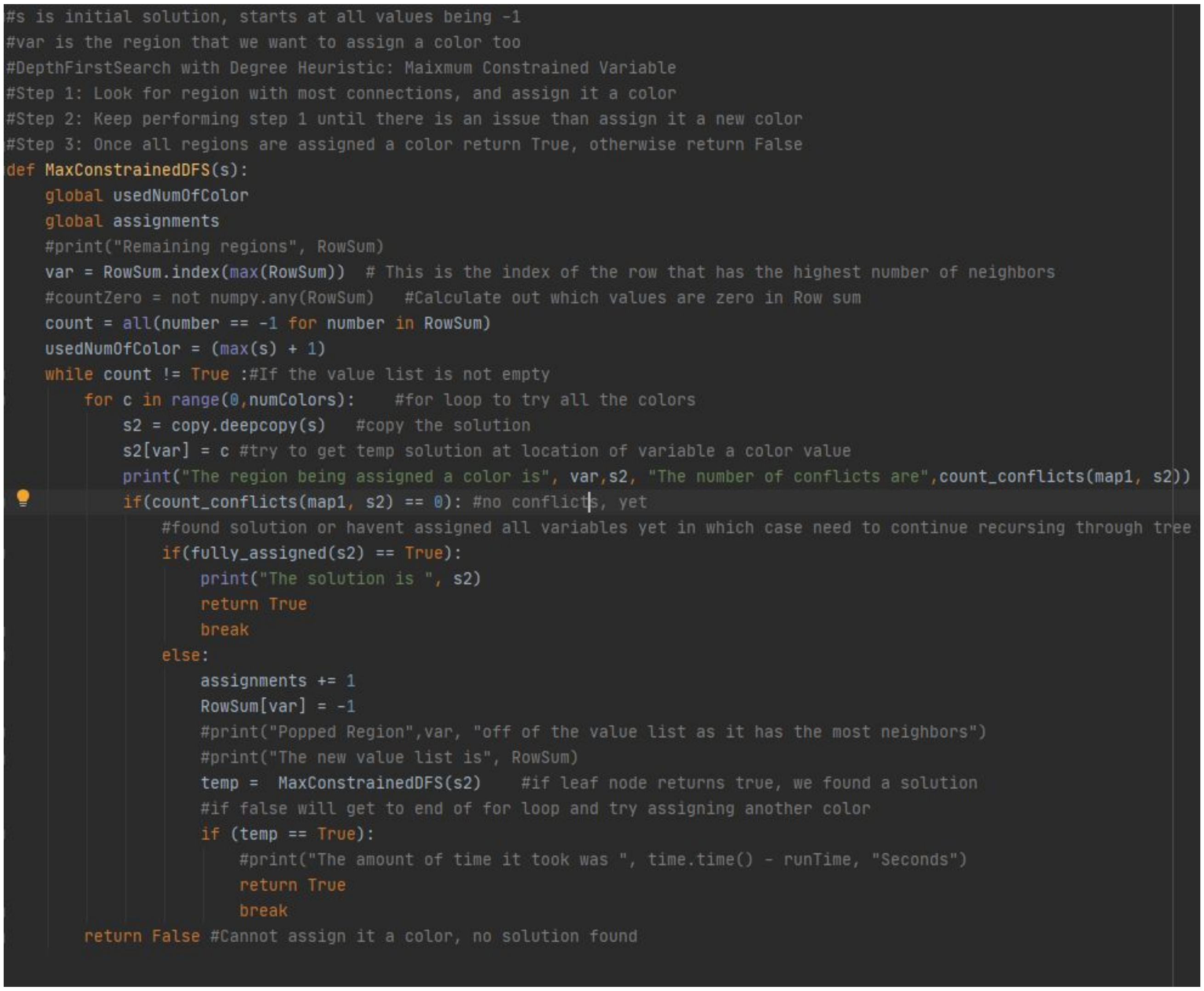
#Step 2: Loop through steps 3 and 4 until a solution is found or there is no more colors

#Step 3: Generate new random state

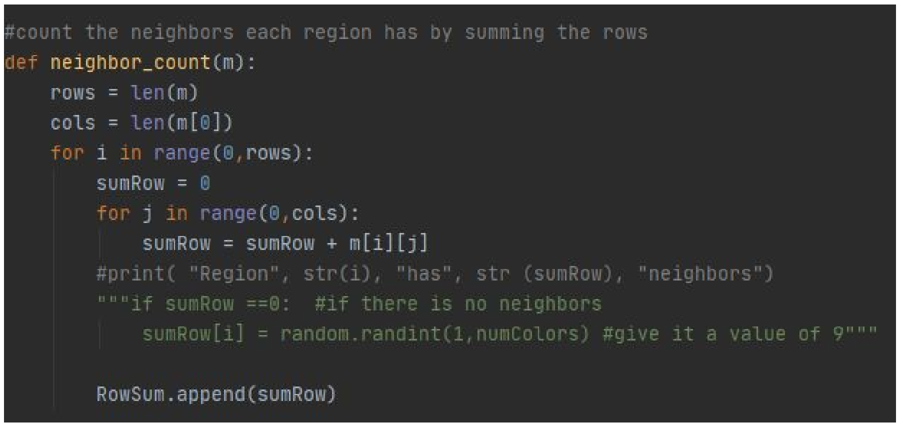
#Step 4: With the state in step 3, if its a goal state return, if it has less conflicts make it current state, if not better keep looping

Hunter Hawkins
CS-470 Artificial Intelligence
Dr. Soule Summer 2020
Project 3 CSP

DFS Algorithm + Supporting Functions:

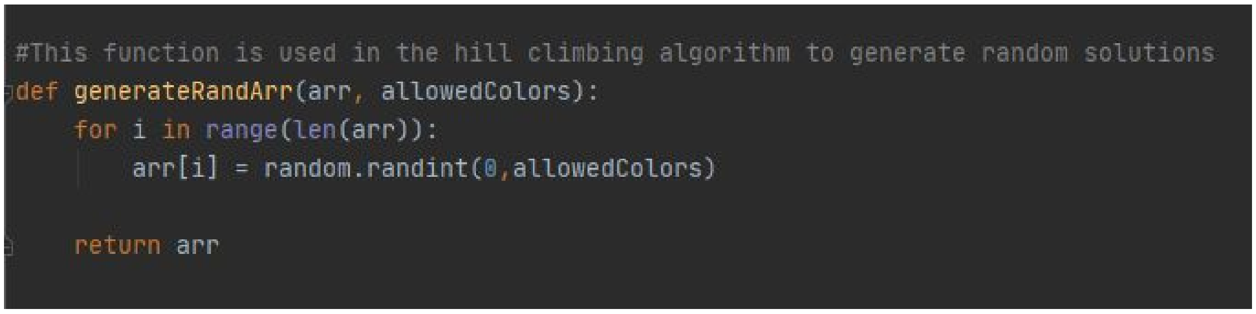


```python
#s is initial solution, starts at all values being -1
#var is the region that we want to assign a color too
#DepthFirstSearch with Degree Heuristic: Maixmum Constrained Variable
#Step 1: Look for region with most connections, and assign it a color
#Step 2: Keep performing step 1 until there is an issue than assign it a new color
#Step 3: Once all regions are assigned a color return True, otherwise return False
def MaxConstrainedDFS(s):
    global usedNumOfColor
    global assignments
    #print("Remaining regions", RowSum)
    var = RowSum.index(max(RowSum))  # This is the index of the row that has the highest number of neighbors
    #countZero = not numpy.any(RowSum)   #Calculate out which values are zero in Row sum
    count = all(number == -1 for number in RowSum)
    usedNumOfColor = (max(s) + 1)
    while count != True :#If the value list is not empty
        for c in range(0,numColors):    #for loop to try all the colors
            s2 = copy.deepcopy(s)   #copy the solution
            s2[var] = c #try to get temp solution at location of variable a color value
            print("The region being assigned a color is", var,s2, "The number of conflicts are",count_conflicts(map1, s2))
            if(count_conflicts(map1, s2) == 0): #no conflicts, yet
                #found solution or havent assigned all variables yet in which case need to continue recursing through tree
                if(fully_assigned(s2) == True):
                    print("The solution is ", s2)
                    return True
                    break
                else:
                    assignments += 1
                    RowSum[var] = -1
                    #print("Popped Region",var, "off of the value list as it has the most neighbors")
                    #print("The new value list is", RowSum)
                    temp =  MaxConstrainedDFS(s2)    #if leaf node returns true, we found a solution
                    #if false will get to end of for loop and try assigning another color
                    if (temp == True):
                        #print("The amount of time it took was ", time.time() - runTime, "Seconds")
                        return True
                        break
        return False #Cannot assign it a color, no solution found
```



```python
#count the neighbors each region has by summing the rows
def neighbor_count(m):
    rows = len(m)
    cols = len(m[0])
    for i in range(0,rows):
        sumRow = 0
        for j in range(0,cols):
            sumRow = sumRow + m[i][j]
        #print( "Region", str(i), "has", str (sumRow), "neighbors")
        """if sumRow ==0:  #if there is no neighbors
            sumRow[i] = random.randint(1,numColors) #give it a value of 9"""

        RowSum.append(sumRow)
```

Hunter Hawkins
CS-470 Artificial Intelligence
Dr. Soule Summer 2020
Project 3 CSP
Hill Climbing Algorithm + Supporting Functions:



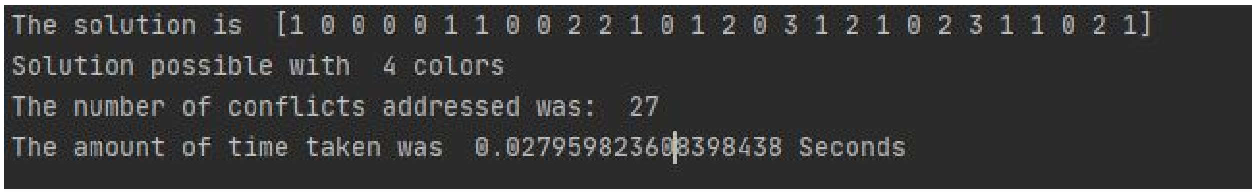
```
#This function is used in the hill climbing algorithm to generate random solutions
def generateRandArr(arr, allowedColors):
    for i in range(len(arr)):
        arr[i] = random.randint(0,allowedColors)

    return arr
```

Hunter Hawkins
CS-470 Artificial Intelligence
Dr. Soule Summer 2020
Project 3 CSP

Results

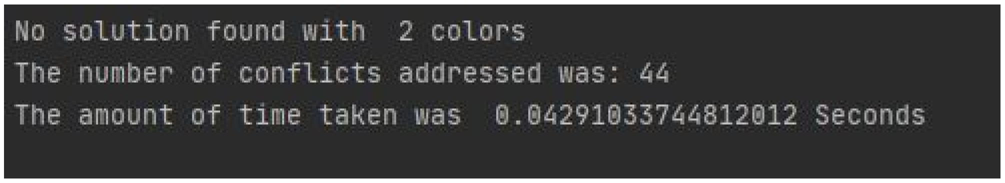
The results of my algorithms varied a large amount. It relates to how the algorithms were designed. The DFS was taking the hardest problems first which resulted in a more front loaded workload whereas the hill climbing algorithm was evenly spread in terms of workload due to it choosing a random state. Both algorithms will come up with a correct result given enough time so they are complete but they may not be optimal in finding the fastest or best solution.

DFS W/ Minimal Colors: In the figure below I have ran the program with the DFS algorithm. By design the DFS algorithm should use the minimal number of colors possible, which in this case is 4. It found the solution in a .0279 seconds and addressed 27 conflicts.



```
The solution is  [1 0 0 0 0 1 1 0 0 2 2 1 0 1 2 0 3 1 2 1 0 2 3 1 1 0 2 1]
Solution possible with  4 colors
The number of conflicts addressed was:  27
The amount of time taken was  0.027959823608398438 Seconds
```

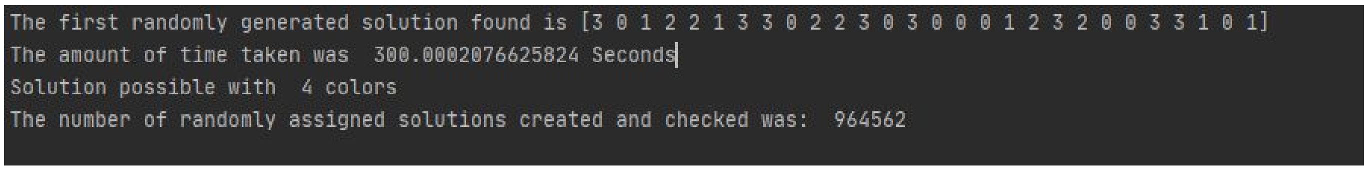
DFS W/ Non possible solution: In the figure below we have the results of a DFS algorithm with no possible solution. We tried to solve the problem with 2 colors, it addressed 44 conflicts and took .0429 seconds to realize there wasn't a solution to the problem.



```
No solution found with  2 colors
The number of conflicts addressed was: 44
The amount of time taken was  0.04291033744812012 Seconds
```

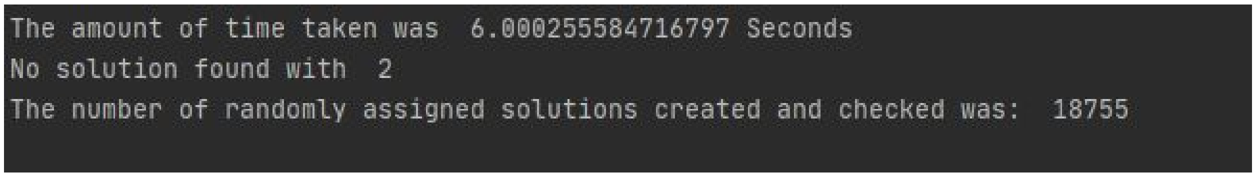
Hill Climbing W/ minimal colors: In the figure below I limited the Hill climbing algorithm to a maximum of 4 colors. When doing so it generated 964,562 random potential solutions, which took 300+ seconds. This shows that it did eventually find a solution but not necessarily in an efficient manner. I gave it 100 seconds to search before adding in another color.

Hunter Hawkins
CS-470 Artificial Intelligence
Dr. Soule Summer 2020
Project 3 CSP



```
The first randomly generated solution found is [3 0 1 2 2 1 3 3 0 2 2 3 0 3 0 0 0 1 2 3 2 0 0 3 3 1 0 1]
The amount of time taken was  300.0002076625824 Seconds
Solution possible with  4 colors
The number of randomly assigned solutions created and checked was:  964562
```

Hill Climbing W/ Non possible solution: In the figure below the hill climbing algorithm was given 6 seconds before adding in additional colors, which peaked at 2 colors (0,1). It found no solution out of 18,755 randomly generated attempts which took 6+ seconds.



```
The amount of time taken was  6.000255584716797 Seconds
No solution found with  2
The number of randomly assigned solutions created and checked was:  18755
```

Conclusion

Overall this project has increased my knowledge with python, as I was really struggling to read in the excel spreadsheet, and the different strategies used to solve Constraint Satisfaction Problems (CSP). I chose to implement the Depth first search with the degree heuristic as it was the first thing that came into my mind when learning about the map coloring problem during lecture. Along with that I was wanting to better understand the hill climbing algorithm so I chose to implement a hill climbing algorithm that was not necessarily the most efficient algorithm but it did work by finding solutions to the problem. This project overall was enjoyable and helped further my learning about AI, Python, and CSP.