Assignment 2: Local Search

Worth: 7.5% of total grade
Due Date: Friday September 25th 2020 23:59pm
Submit two files named
For assignment 2a submit $your_upi.py and
for assignment 2b submit $your_upi.pdf respectively
to the CANVAS dropbox (where $your_upi is replaced by your upi).

Download the code that comes with the assignment. The hill_climbing.py code has 5 places where you need to write code. The test.py file has tests you can run against your code to make sure they work correctly. For full marks in Part A your code will need to pass the provided tests, as well as continue to perform correctly with some additional states. Make sure that you name your python code <your upi>.py when submitting it!

Part A

Task 1 [1 mark]

Complete the n-queens implementation provided by implementing the 2 missing methods in the NQueensHillClimbing code; actions and result. Refer to action_and_result_examples and the test_action_and_result function in tests.py to ensure your method output is correct.

Task 2 [1 marks]

Write a function called hill_climbing_instrumented, which performs hill climbing exactly as the provided hill_climbing function does, but outputs additional information. It must return a dictionary containing entries for the number of expanded nodes and whether the problem is solved or not, and lastly the board state found with the highest hill climbing value. Refer to the function signature and dummy return value in the provided function skeleton, and check the corresponding examples and tests in the test.py file to make sure you return the correct information.

Task 3 [1 marks]

Write a new hill climbing function called hill_climbing_sideways, that implements the sideways moves discussed in the lectures. It should return the same information as task 2, and additionally the number of sideways moves taken. This must allow the user to specify the maximum number of sideways moves used. Again, refer to the function skeleton and test.py for the exact format required. Make sure you continue to use the provided method for selecting the best neighbouring state (Node.best_of, as used in the hill_climbing function) to ensure your output matches the test examples.

Task 4 [1 marks]

Write a new hill climbing function called hill_climbing_random_restart, that implements the random restarts discussed in the lectures. It should return the same information as task 2, and additionally the number of restarts taken. This must allow the user to specify the maximum number of random restarts used. Again, refer to the function skeleton and test.py for the exact format required. Use the provided NQueensProblem.random_state method to ensure your output matches the test examples. When debugging your code, don't forget to use random.seed for reproducible results.

Part B

Task 5 [3.5 marks]

Part B should be a maximum of 1 page including the table.

Complete the table that includes a) the number of nodes expanded when solving the problem b) the amount of time used when solving the problem c) the probability that each of these will solve the problem. You will need to run more than one problem to see a pattern, so these should probably be averages and standard deviations. You should try at least 3 different number of queens (X, X+1 and X+2) so you can see how the problem grows with the number of queens.

In the report you should discuss a) how you think the number of expanded nodes, the amount of time, and the probability will change as the number of queens increases b) how you determined the number of queens to run c) how you determined the number of problems you need to run for each of them.

There are a number of ways to time your code in python, and which you use is up to you. If you are using ipython (or a Jupyter notebook) then the easiest way is probably the %timeit command. This is basically a syntactic sugar over the timeit module from python's standard library, which you can use instead. Or you could also time the code by calling time.time() to get the start and end times. Make sure you compute averages and standard deviations!

You will need to determine how many runs you need to run to get a stable answer for your averages and standard deviations for the table. The table should look as below:

# queens	method	Average # of nodes +/- std dev	Average time to solve +/- std dev	Probability of solving
X queens				
	hillclimbing			
	sideways			
	random restart			

X+1 queens				
	hillclimbing			
	sideways			
	random restart			
X+2 queens				
	hillclimbing			
	sideways			
	random restart			

Report Rubric:
2 points for the table
1.5 points for the report