CMPSCI 383 Homework 2

Shivangi Singh

Assigned: Feb 21 2018; Due: March 5 2018 @ 11:59 PM EST

Abstract

To submit this assignment, upload a pdf to Gradescope containing your responses to the written response questions below. You are required to use LaTeX for your write up. We suggest using sharelatex.com or overleaf.com, since they do not require you to install anything on your computer. When submitting your answers use the template LaTeX code provided and put your answers below the question they are answering. Do not forget to put your name on the top of the pdf. To submit the coding portion of the assignment upload your `my_nqueens.py` file to the Gradescope programming assignment. Do not change function definitions or return types unless otherwise specified. Your work for all parts of this assignment must be your own (do not collaborate with other students in any way when completing this assignment).

1 N-Queens problem (60 pts)

In class, we discussed various ways to tackle the classic N-Queens problem. In this section, you will first manually go over the hill climbing solution for this problem and then implement different solutions to the problem in python.

1.1 Hill climbing (20 pts)

Given below is a random placement of four queens on a 4×4 chess board. We will use the representation from class, where the board is represented by one number per column, denoting which row the queen is in.

Find the next two moves for the board using hill climbing with the number of attacking pairs heuristic. As your answer to this question, provide the new board configurations represented as a list where the index represents the column and the value represents the row where the queen is present in that row. For reference, the representation of the initial board configuration is given.

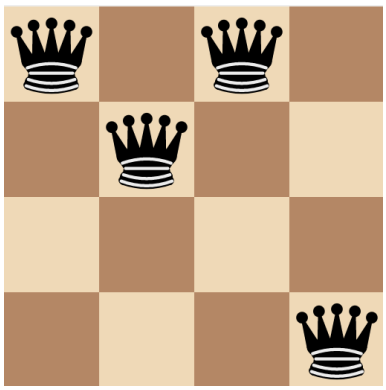

Figure 1: Initial configuration: [3,2,3,0]

Answer: Intital State: [3,2,3,0]
Next Move: [3,1,3,0] [2,1,3,0]

1.2 Implementation (40 points)

You will implement two different solutions to the N-Queens problem here using Python 3.6 using the anaconda distribution. Information about setting up the environment is given in the previous assignment.

1.2.1 Hill Climbing (20 points)

Start from the template code in `my_nqueens.py` and fill in the code for the required functions. Your program will take n as an input and return the N-Queens solution for an $n \times n$ board. The solution should represented as a list of row indices as described in 1.1. You will use the hill climbing method here with the number of attacking pairs heuristic.

1.2.2 Simulated Annealing (20 points)

Start from the template code in `my_nqueens.py` and fill in the code for the required functions. Your program will take n as an input and return the N-Queens solution for an $n \times n$ board. The solution should be represented as a list of row indices as described in 1.1. You will use the simulated annealing method here with the number of attacking pairs heuristic. The temperature and annealing rate will be implemented as parameters to the annealing method.

Note that a solution might not always be found with a given starting board configuration using both these methods. You will therefore need keep track of whether a solution has been found in each run.

2 Genetic Algorithm for Sudoku (20 pts)

In class, we looked at genetic algorithms that are designed to evolve candidate solutions into better solutions by combining them. Think about how you can apply this to solving the sudoku puzzle.

In your own words, describe how you would formulate the Sudoku puzzle as a GP search problem. Your answer should at least include information about the representation of the state as a string, the selection (fitness function), the mutation, and the cross-over methods you would use.

Answer:

A genetic Algorithm has the following pattern:
1) Initialization
2) Evaluation
3) Selection
4) Crossover
5) Mutation

Firstly we can create a board that is a list of lists 81 numbers where each list inside the outer list contains 9 elements/numbers ranging from 1-9 eg : board=[[1,2,3,3,4,5,2,5,6],[1..],[4..],...] len(board) =9. (This is a state representation of the board). We can evaluate the fitness of the generated board using the fitness function and use the boards with the best two fitness function to perform selection on. The fitness function returns the number of clashes/repetitions of numbers in the inner list (row clashes) + the number of clashes/repetitions of numbers in the columns i.e. number of clashes when the inner lists element at index i are compared across all the inner lists, also we could similarly find the number of clashes in the smaller box of the Sudoku, by taking 3 lists at time and finding the non unique numbers in sets of 3 i.e. (0-3,4-6,7-9) and add them.

Therefore our fitness function is = number of row clashes+ number of column clashes + number of clashes within the smaller box.

Then we can generate new children by crossing over and mutations. Crossing Over:- we could take the 1st 5 rows of one state with and fuse it with the last 4 rows (last 4 lists) of the other list.
Eg: S1= [[a], [b], [c], [d], [e], [f], [g], [h], [i]]
S2= [[1], [2], [3], [4], [5], [6], [7], [8], [9]]
New off springs by crossing over =
o1=[[a], [b], [c], [d], [e], [6], [7], [8], [9]]
o2=[[1], [2], [3], [4], [5], [f], [g], [h], [i]]

We could also mutate the lists inside lists at random spots. Like if a=[1, 3, 4, 5, 6, 2, 4, 5, 7] in the off spring it could be mutated/changed as a=[2, 3, 3, 5, 6, 2, 4, 5, 7]. Then evaluate the fitness to classify the children generated as the solution or perform steps (3-5) till you find the solution or overrun the number of times you want the algorithm to be run.

3 Constraint Satisfaction Problems (20 pts)

Read chapter 6 from the textbook about constraint satisfaction problems until section 6.3. Do exercise questions 6.1 and 6.2.

Answer:

6.1) How many solutions are there for the map-coloring problem in Figure 6.1? How many solutions if four colors are allowed? Two colors?

There are 18 solutions for the map-coloring problem in Figure 6.1 if we can choose 3 from colors and 768 solutions if we can choose from 4 colors and no solution if we are limited two colors given our constraint that no two states can be colored the same.

6.2) Consider the problem of placing k knights on an n n chessboard such that no two knights are attacking each other, where k is given and k n2.

a. Choose a CSP formulation. In your formulation, what are the variables?

b. What are the possible values of each variable?

c. What sets of variables are constrained, and how?

d. Now consider the problem of putting as many knights as possible on the board with- out any attacks. Explain how to solve this with local search by defining appropriate ACTIONS and RESULT functions and a sensible objective function.

A CSP problem has three properties X= set of all the variables in this case which is the Knights so each variable in X represents the location of the knight on the board, D= the domain possible for all the variables in X which is any coordinate on the n by n board (0,0),.....(n-1,n-1), and C is the constraint which are the set of conditions that are placed to make solving the problem easier. In our case the constraint is that every pair of knights are constrained such that no two knights can be in the same square or in a square to where the knight can move.

Solving the Problem using Local Search algorithm : We could use Simulated Annealing by first creating an array of positions for the k knights. Eg: pos=[(0,1),(9,8),(a,b)....] where the a,b can take up any value between 0 and n-1 (where n is the size of the board). We could make the stepCost function (heuristic function) to return the number of attacking pairs in the given orientation of the board. Then we can try to minimize the heuristic function by making moves that minimize the stepCost. The moves can be made by moving one of the knights under attack to a new position that is not already occupied.