Machine Learning : Optimization

Guidelines and Submission Instructions:

1. The project is worth 30% of your overall module grade. You will produce a **.py file** containing all your code and a **short document** detailing your evaluation of the various algorithms.

2. Upload your solution python file (**.py file**) and report as a single .zip file to Blackboard before **20:00 on May 6th**.

3. Go to the "Project Optimization" folder in Blackboard to upload your file.

4. Once you have submitted your files you should verify that you have correctly uploaded them. It is your responsibility to make sure you upload the correct files.

5. Please make sure you **fully comment your code**. You should clearly be explain the operation of important lines of code.

The marks will be distributed as follows:

- Hill Climbing and Random Restart Hill Climbing Algorithm **(50%)**
- Simulated Annealing Algorithm **(20%)**
- Evaluation Report **(30%)**

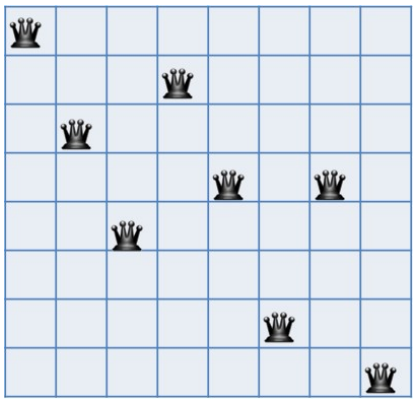
Optimization for N-Queens:

The objective of the n-Queens puzzle is to place *n* queens on an *n*×*n* chessboard so that no two queens attack each other.

The image below depicts the problem when n=8 (standard chess board). The 8-Queens can be formulated as an optimization problem. A state consists of a specific placement of the 8 queens on the chess board. The goal state is the arrangement of the 8 queens on the board in such a way that none can be attacked. The objective of this assignment to write the following optimization algorithms to solve this problem and produce a short report showing your evaluation of their performance:

1. Hill Climbing Algorithm (Steepest Descent/Ascent)
2. Random Restart Hill Climbing (RR-HC)

3. Simulated Annealing Algorithm (RR-SA)



The implementation of your algorithms should give special consideration to the following issues:

- How will you represent a specific state? (refer to Week 11 lecture notes for an example of an appropriate representation for this problem)
- What action will your program take in order to generate one or more neighbouring states (one method is suggested below in the Instructions section below in in the Instructions).
- How will you evaluate the quality of a given state? (refer to Week 11 lecture notes for appropriate heuristic functions for this problem)

Instructions:

1. Generate a **random initial state**. An initial state in your problem will be n queens positioned randomly on the board. Make sure that you parameterize the number of queens so that you can expand the size of the board. You should have decided on an appropriate representation before generating a state.

2. A **heuristic function** that will take in a given state and return a heuristic value for that state. Refer to the lecture notes for an appropriate heuristic function for this problem.

3. You will need to provide code that will generate neighbouring states. There are many different ways you can generate a neighbour. Implement a simple mechanism. For example, you could generate a new neighbouring state by moving the position of a single queen by one place. It is important in hill climbing that you make only small alterations to the current state to produce a neighbouring state.

4. Develop a basic steepest ascent/descend hill climbing algorithm. This function will use the code you developed in part 3 to generate a number of neighbouring states. It will then search all neighbouring states. It will return the neighbour state with the best heuristic value, assuming the best neighbouring state is better than the current state. If none of the neighbours have a better heuristic value than the current state then the algorithm stops and returns the current state (we refer to this as convergence). Again see Week 11 notes.

5. You will have noticed that the hill climbing algorithm can often converge to a suboptimal solution. Use the code you developed in part 4 to implement a random restart hill climbing (RR-HC) algorithm. Initially set the maximum number of restarts to 500. Each time the algorithm restarts it should generate a new random configuration for the initial state of the board. If the algorithm identifies a solution to the problem it should exit and provide the details of the solution obtained. Please make sure that the code keeps track of the total number of moves taken by the hill climber to arrive at an optimal solution.

6. Create a random restart simulated annealing (RR-SA) algorithm. You should be able to reuse much of the code you wrote for the hill climbing version in part 5.

7. When running optimization algorithms on a problem such as the n-Queens we are interested in their relative performance. For this experiment we will based performance on the total number of cumulative moves taken in order to reach an optimal solution for boards ranging in size from n= 8 to n=25. This will involve repeatedly running your RR-HC and the RR-SA algorithm for various board sizes and recording the number of moves taken to reach an optimal solution. What is the relative performance of the RR-HC compared to the RR-SA? What is the impact of various parameters that impact the overall performance? Do changes in the method of neighbour generation make any significant difference? These are the questions your evaluation should answer. Please note that the best way to present this information is on graphs. You report should include a short explanation interpreting the results of each graph.