Optimization

THE N-QUEENS PROBLEM

David Coughlan
R00009964 | DCOM4A

Contents

Representation

For this problem initially, I thought of using a two-dimensional array to represent the board. Using one to represent the presents of a queen and zero an empty square on the board. Having thought more about it and considering later increasing the board and the search space exponentially I found the way I was going to represent the problem. Since queens can attack up, down, left, right and diagonally it made sense to only have one queen per column. With this I could use a one-dimensional array with each index representing the position the queen holds in that column. This shortens the search space dramatically.

I made a board at a size of n and then passed it to a method for randomly populating each column of the array using numbers from zero to the size of the board.

Scoring

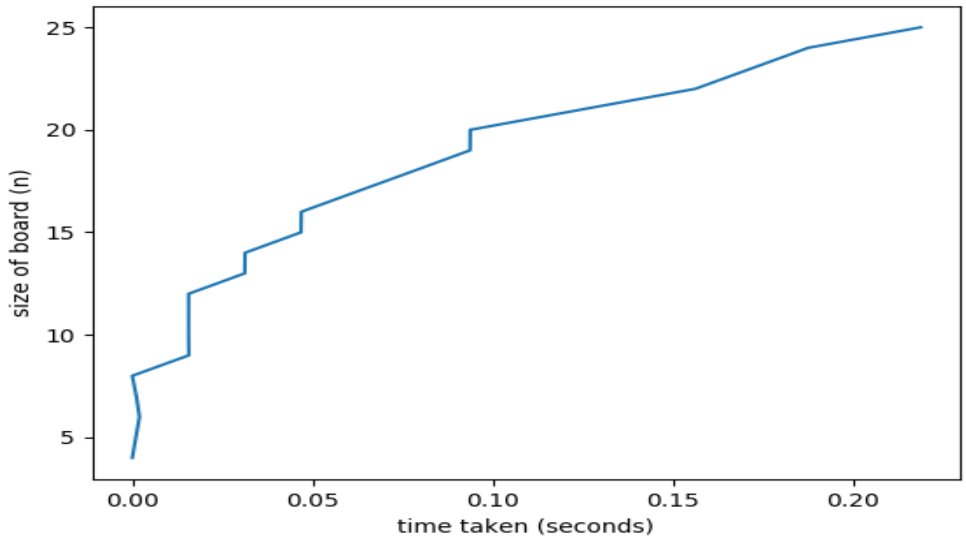
Picking the heuristic took a little bit of time. Knowing the allowed moves of a queen I now had to apply these to a one-dimensional array. I decided it was easier to count the pairs that are in an attacking state. This means that as my scorer returns an integer the lower the score the better the state of the board.

Hill climber

The hill climber is an optimization technique that belongs to the local search family. It is an iterative algorithm that starts with an initial solution (random state) and then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, one with a better heuristic score, then that is saved as the best current solution and the process continues until no further improvements can be found.

In my hill climber I ran two loops and in these I changed one index of the state at a time incrementing through all the different combinations for that index. I scored each as they were generated and then compared. If the state was better, I saved it as the one to beat as well as its score. I counted a move as every time I found a better score. I then returned the moves, best score and the best state.

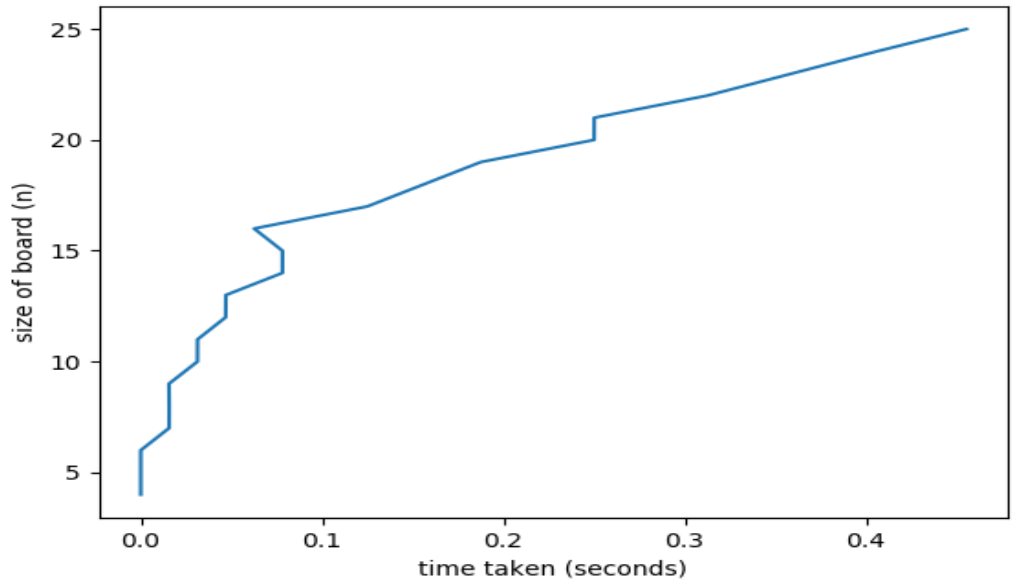
I then tested this algorithm using time as a measurement. I increased the board size in a loop and graphed the results of the size of the board against the collection of times it took to complete on each board size.



I noticed how fast it was taking only seconds to handle the larger size boards. It did start to take longer once the size got over 15. But it is hard to say due to the hill climber usually converging on its closest approximation of the optimal solution.

Random Restart Hill climber

Random restart hill climbing is a meta-algorithm built on top of the hill climbing algorithm. It runs the hill climber algorithm for a set number of times. At each iteration checking the score and again saving the best state and best score. If the hill climber's results start to plateau, we stop getting a new best score, then a restart is performed. This is done by randomly generating a new board and then checking the score of the board. The loop of this happening increments to 500. Then function then returns the best score, state, total number of moves and the number of restarts.



From my testing of this I noticed obviously similar behaviour to the hill climber as it is based on that algorithm. Once again, the results seemed to start to get bottlenecked when the board size was up to 15. Times after this started to get much greater as the search space grew exponentially.

Simulated Annealing

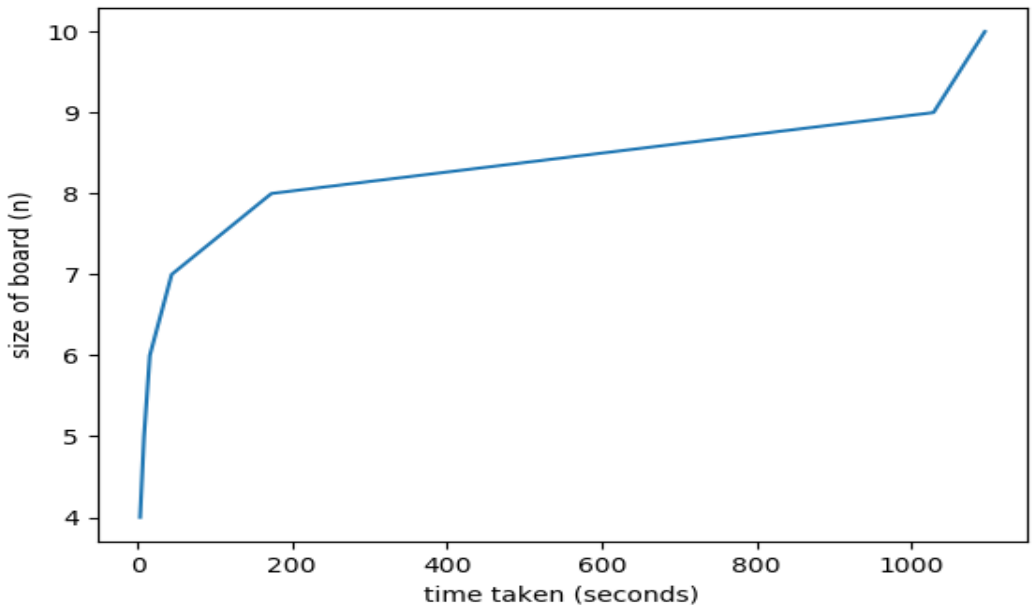
Simulated annealing uses two tricks. The first is where in some trades it can accept a move with a score that is not better than its best so far. These serve to allow the solver to explore more of the possible space of solutions. These bad moves are allowed through a certain formula.

$$e^{-\Delta D/T} > R\,(0,\,1),$$

Where the Delta D is the change of the distance from the trade. Negative for a good trade and positive for a bad. T is a synthetic temperature and R (0,1) is a random number between 0 and 1. D is cost function and corresponds to the free energy in the case of an annealing metal. If the temperature is large the algorithm is more likely to accept bad trades and a large area of search space is accessed.

The next trick that it uses is to apply an annealing rate. This is used to lower the temperature of the process. After making many trades and observing that the cost function declines only slowly the temperature is lowered, and this limits the number of bad trades allowed to occur. When the temperature is lowered numerous times, it reaches a low value. Then the algorithm may only accept good trades to find the local minimum of the cost function. It is given 500 attempts and if the scores have plateaued I randomize the board.

This algorithm takes a long time for me to run making it very hard to graph it for the same magnitude as I have done previously with the hillclimber and the random restart hill climber. Due to this I decided to cap the board increments at ten. This still illustrates its performance on the problem.



From the graph you can see that from the size of eight up it really starts to skew dramatically. Making the cost of running the algorithm on greater numbers too costly. I had initially written all methods and tested them on a small board however when run on larger boards every flaw is amplified in the results. Even though the cost of this algorithm is very high it does produce an optimal solution and with a little bit more time could be tweaked run more efficiently for larger number boards.

Conclusion

Looking at the trade-off between the size of the board and the time taken to run. I believe that the hill climber offers the best solution. It is a less resource heavy algorithm than the random restart hill climber and the simulated annealing. It often comes out to be the best once it is tweaked. That is why it is so popular for this problem.