CISC 352: Artificial Intelligence

Assignment 1: N-Queens Problem

Authors: Alice (20111076), Jean (20094416), Renee (20113399), Zack (20124496), Teo (20100698)
All group members contributed equally

February 12, 2021

Introduction

The N-Queens problem involves placing N queens on an NxN chess board, in which N is an element of the natural numbers greater than 3, such that no two queens threaten each other. It was first composed in 1848, and has been a topic of interest for mathematicians and computer scientists ever since.

Algorithm Overview

In essence, if N < 128 the algorithm uses a greedy board initialization, otherwise it randomly generates a starting board. It then performs a minimum conflicts search, selecting random columns and moving the corresponding queen to a row with fewer conflicts until a solution is found. In order to scale this approach, the search is restarted with a new board initialization if a certain number of steps (based on the board dimensions) is exceeded.

Greedy Board Initialization

The algorithm works by placing each required queen in a position based on the number of preexisting conflicts on the board. It works alongside the *return_column_conflicts* method, which calls the *queens_conflicts* method. *Return_column_conflicts* provides a list of the amount of conflicts each board space in a given column gives, which we can use to determine the minimum conflict value thanks to *min(array)*. Towards the end of the board, this minimum may not be 0. However, we still place a queen in the minimum spot in this situation because our goal is not to place the queens in a proper board state without any position changes, rather just to have an efficient starting board and to reach a correct board state efficiently. After we figure out what the lowest conflict value in that column is, we then look through the column again to find all indexes with that minimum conflict value. We store those indexes in an array, and pick a random one where we will place the queen. This prevents the queen from always being placed in the first position containing the minimum conflicts. Although this algorithm has a relatively high time complexity, we found that it was still faster to use this greedy initialization approach as opposed to a random initialization approach for N < 128.

Conflicts Counting and Solution Method

A method for counting the number of conflicts a given queen is experiencing, relative to the other queens on the board, is used many times to solve the n-queens problem. The method for this is *queen_conflicts,* which takes as input a row, col position for a queen, and given the state of the rest of the board (stored as a list), it returns the number of conflicts the given queen is experiencing.

It does this in O(n) time, by doing a single pass on the list representing the state. We know,

based on the fact that each column has only one queen, that there are no column conflicts. Checking for row conflicts is simply a matter of ensuring that there are no repeated values in the list. Checking diagonal conflicts is the only somewhat tricky part, and this is achieved by checking if the row, col value of the queen we're checking, *(row, col)* and the row, col values of the current queen (*row_2, col_2)* have the same difference in absolute value pairwise. Explicitly, that is checked as follows: *if abs(row - row_2) == abs(col - col_2).*

This method is then used in the *check_solution* function to check if a current state is a valid solution, by checking that all the queens have 0 conflicts, i.e. for all queens on the board *queens_conflicts* returns 0.

Minimum Conflicts

A row with minimum conflicts is found using a simple for loop and the queens_conflicts method in the solve function. For a selected queen, each row in the corresponding columns is iteratively checked for the number of conflicts with the rest of the queens. The queen is then moved to the row resulting in the fewest conflicts.

Restart Board Initialization

We begin with a random board initialization method upon the first round of our search. If, however, we exceed the calculated maximum number of steps and require a restart we turn to a different board initialization method. In contrast to a randomized or greedy method, the next method we try is a numerical one. By exploiting the symmetry of the board and the numerical patterns emerging from the dimensions of the board, we can assign queens to more optimal positions and hope our search is able to improve the speed at which it finds a solution.

Conclusion

Before deciding on a randomized approach with calculated restarts, we tried a number of other methods including greedy board initialization, ordering the movement of queens based on the number of conflicts, and performing simulated annealing to escape plateaus and local optimums. Although these techniques reduced the number of steps, the overhead of their implementation was too much to scale n at a reasonable rate.

Future work could focus on reducing the number of steps by tracking queens which have been checked and avoiding picking the same queen more than once a round if no other queen has been moved.