AI: Principles & Techniques
Programming Task 2; N Queens Problem

Elianne Heuer, s4320514
Jordy Aaldering, s1004292

October 16 2020

1 Introduction

The N Queens Problem is the problem of placing N queens on an $N \times N$ chessboard so that no two queens threaten each other; thus, a solution requires that no two queens share the same row, column, or diagonal. A solution exists for all natural numbers N except for two and three.

In this report we will describe and give a solution to this problem using linear programming and the Python package PuLP.

2 Methods

The decision variables of this problem are all possible positions to place the queens. This corresponds to a $N \times N$ grid containing binary values. Where a zero means that there is no queen and one means that there is a queen in that position. So this means we have N^2 decision variables.

There are several constraints the solution must follow. Firstly there must be exactly N queens in total. Where every row and column must contain at most one queen, and where every diagonal must also contain at most one queen. In our solution we combined the first constraint with the second and third ones. Meaning that we get that there must be exactly one queen per row and column. This saves us a constraint and simplifies the solution. There are N rows, N columns, and $4N - 2$ diagonals. So in total we have $6N - 2$ constraints.

It does not matter whether we minimise or maximise. This is because a solution is always correct when there are exactly N queens, and there are no better or worse solutions.

3 Results

Below is our solution to the N Queens Problem, following the constraints explained above.

```
prob = LpProblem(f"{N}_Queens")
board = [[LpVariable(f"{x},{y}", cat=LpBinary) for x in range(N)] for y in range(N)]

for i in range(N):
    # Exactly one queen per column
    prob += lpSum([board[x][i] for x in range(N)]) == 1

    # Exactly one queen per row
    prob += lpSum([board[i][y] for y in range(N)]) == 1

    # At most one queen per positive diagonal
    prob += lpSum([board[x][i - x] for x in range(i + 1)]) <= 1
    prob += lpSum([board[x][N - x + i] for x in range(i + 1, N)]) <= 1

    # At most one queen per negative diagonal
    prob += lpSum([board[x][x - i - 1] for x in range(i + 1, N)]) <= 1
    prob += lpSum([board[x][N - 1 - i + x] for x in range(i + 1)]) <= 1

status = prob.solve()
if LpStatus[status] == "Optimal":
    for row in board:
        print(" ".join(["Q" if value(v) else "." for v in row]))
```

In our code we initialise our board in N^2 steps. After this we loop N times through all the N columns, N rows and 2 times through the N diagonals. This gives us a complexity of:

$$\mathcal{O}(N^2 + N^2 + N^2 + N^2 + N^2) = \mathcal{O}(N^2) \tag{1}$$

4 Discussion

Our solution to the N Queen Problem using linear programming is optimal and also easily understandable. Which is why we believe that this is a very good solution. The code for looping through the diagonals maybe could be written in one line, but we did not see how and we chose to place all the constraints inside one for-loop.