CP468 – ARTIFICIAL INTELLIGENCE: N QUEENS

BENJAMIN SEGALL (151403050), YITONG WANG (150757470), MENGDAN WAN (146802060), JOEL KIPFER (150771770) [1]

2019-12-01

CONTENTS

LIST OF FIGURES

[1] *CP468 Artificial Intelligence, Fall 2019, Wilfrid Laurier University*

1 PART A

1.1 Constraint Satisfaction Problem: N Queens

Problem Statement: Given an N by N chessboard and N Queens, place the Queens to create a stalemate.

1.1.1 *Goal State: The Queens are placed such that there are no two Queens that share a row, column, or diagonal.*

1.1.2 *Initial State: An N by N board and N Queens placed at random. They must be rearranged to a state where they cannot attack one another.*

1.1.3 *Type: This is a closed world problem.*

1.2 MIN-CONFLICT Algorithm

Description: this algorithm selects a value for a variable that leads to the minimum number of of conflicts with other variables by local search. In the N Queens problem, the initial state is a randomized board with Queens, where each Queen's number of conflicts can be counted by the number of constraints it violates (i.e. number of Queens that can be attacked).

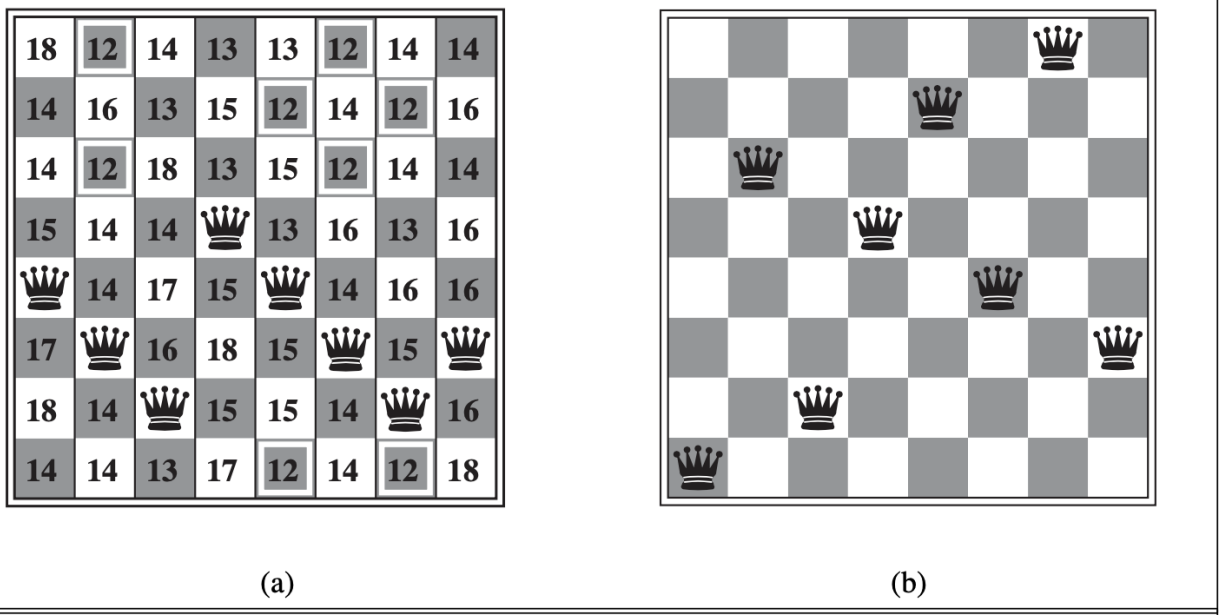

Figure 4.3 (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.

Figure 1: An example of the minimum conflict heuristic applied to the 8 Queens problem. Source: Russell, S. J., Norvig, P. (2010). Artificial intelligence: a modern approach. Hoboken: Pearson.

2 PART B

2.1 Choice of Language

The solution we propose is offered in both Python and C++. Initially, Python was chosen for its simplistic structure, however it proved to be inefficient to solving large data sets as we approached one hundred thousand Queens. This solution was kept as a reference and an alternative solution in C++ was written with the MIN-CONFLICT algorithm, which is much faster. At the time of this submission, both solutions can solve the N Queens problem, though only the C++ solution is optimized. The C++ solution was able to complete this task in approximately 4 hours on an AWS computer.

2.1.1 *Python Solution:*

This application is derived from the previous assignment's CSP solution. AC3 was used as a naive approach, however it proved to be too slow. On top of Python's dynamic typing, this solution was eliminated along with the language.

2.1.2 *C++ Solution:*

However, arrays were declared in the style of pointers, therefore speeding up the computation process. Memory was dynamically allocated by size of N, making this solution more space efficient as well.

2.2 Design

There were two options when generating the initial state: greedy or random. After comparing both options, it was obvious that the greedy approach was more costly. Greedy also provided just a small advantage, therefore a randomized initial state was chosen. Due to this decision, the end state from this solution also tends to be more randomly spread.

The biggest challenge was designing an MIN-CONFLICT algorithm that was well suited for this problem statement. This solution check's each Queen's constraints, and picks a random Queen with the max number of conflicts if there are several. Else, just the Queen with the largest conflict is chosen. Then, this Queen is moved to a row with minimum conflicts within that column. If there are multiple columns with the same minimum conflicts, then ties are broken randomly.

Another improvement from a naive MIN-CONFLICT algorithm was that conflicts is computed once at the beginning for each Queen. A Queen's individual number of conflicts is only updated when a Queen is moved and her move affects said Queen, then the affected Queen(s)' conflicts are updated. The total number of conflicts is not computed each time.

It should also be noted that number conflicts is found of iterating every row in a column, therefore reducing the run time from cubic to quadratic.

2.2.1 *Data Structure*

The implementation chose pointer formatted arrays that can be reused as data updates. Therefore, less memory needs to be allocate, which makes the memory footprint manageable for even large N values.

3 PART C

3.1 Installation Instructions

Clone repository from by clicking here or by going to this URL: https://github.com/bentekkie/CP468_FinalProject. This repository contains both the Python and C++ solution (under the folder titled "cpp"), as well as sample result sets and graphics.

3.2 Executing Python Naive Solution

*Ensure you have Python 3.8 installed.

Change directory to where *solve.py* is stored. Run this command in the command line or terminal of your computer:

```
>> python3 solve.py
```

3.3 Executing C++ Solution

*Ensure you have a compiler installed.

Change directory to where *solve.cpp* is stored. Run this command in the command line or terminal of your computer to compile:

```
>> ‘g++ solve.cpp
```

Ignore the warnings. Then, run the program using

```
>> ./a.out <n> <option>
```

The options for <n> and <options> are as follows:

- n: the number of queens
- option: an integer option for output customization. The possibilities are:
  - 0. Solve n-queens problem for n queens, output will be ‘n,<execution time(sec)>,<did it solve the board? 1: 0 (bool)>‘
  - 1. Solve n-queens problem for i queens where i increments from 100 to n by 100 each step, output will be same as ‘0‘ with a line for each iteration ‘i‘
  - 2. Same as ‘0‘ but also will print the board state as ‘n‘ comma separated integers where the ‘i‘th integer ‘j‘ represents a queen at column ‘i‘ and row ‘j‘
  - 3. Same as ‘2‘ but only print the board state

4 PART D

4.1 Results

Note: A sample of the results of 100 to 1 million Queens is available here or at this URL: https://github.com/bentekkie/CP468_FinalProject/tree/master/results. Due to the size the results, it is not included here.

4.1.1 *10 Queens sample solution*

```
10,0.000123315,1
7,4,0,3,8,6,2,9,5,1
```

4.1.2 *Interpreting results*

The first line is in the format
(n, time to solution in seconds, result? 1:0 (bool))

Each index in the second line is a column, it indicates which row the Queen resides in a solved board. For example, index 0 in the example above has a value of 7, meaning there is a queen at the 8th row of the first column.

Results for 100 to 1 million Queens can be interpreted the same way.

5 PART E

5.1 Number of Queens and Runtime

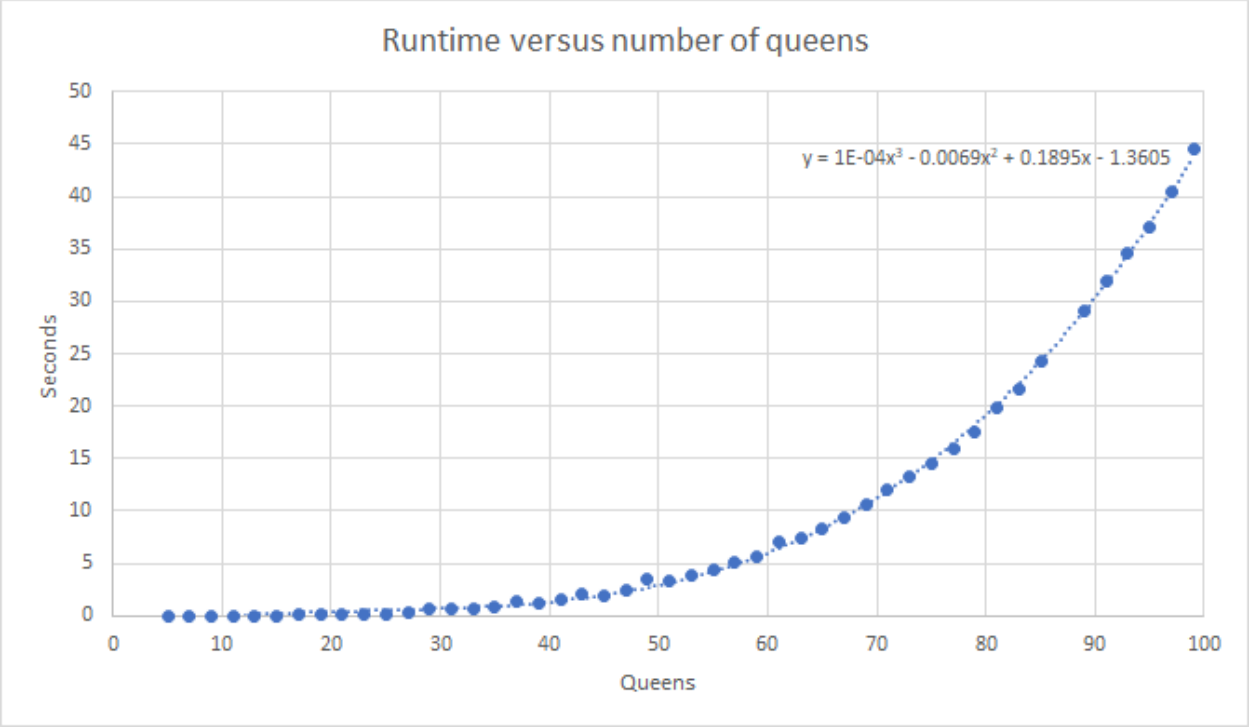

Figure 2: The amount of time taken to find a solution based on the number of Queens.

5.2 Graphic representation of 100, 1,000, and 10,000 Queens

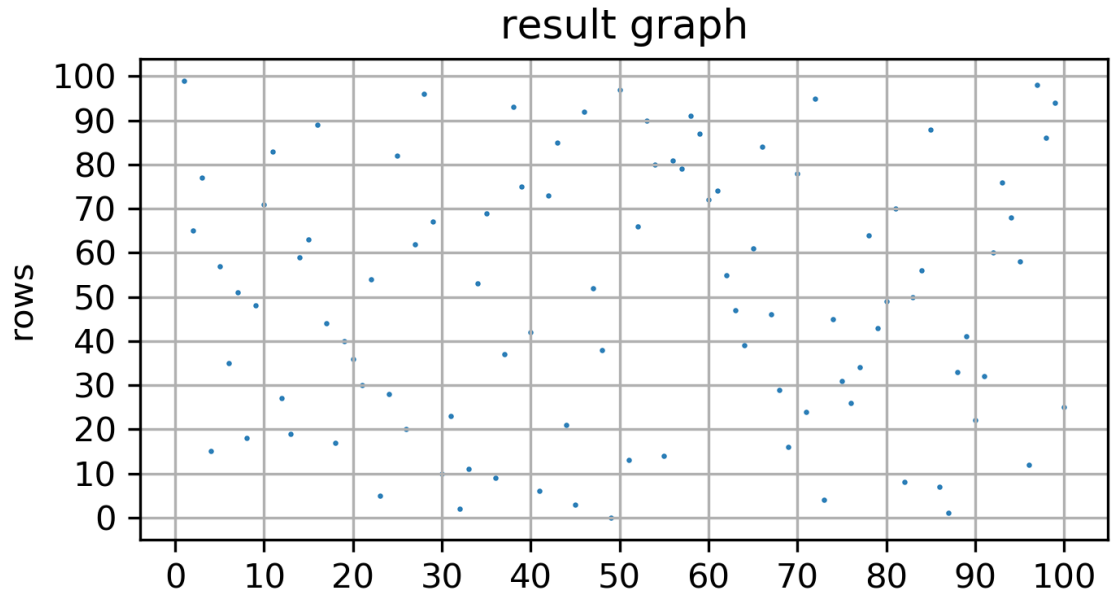

Figure 3: 100 Queens

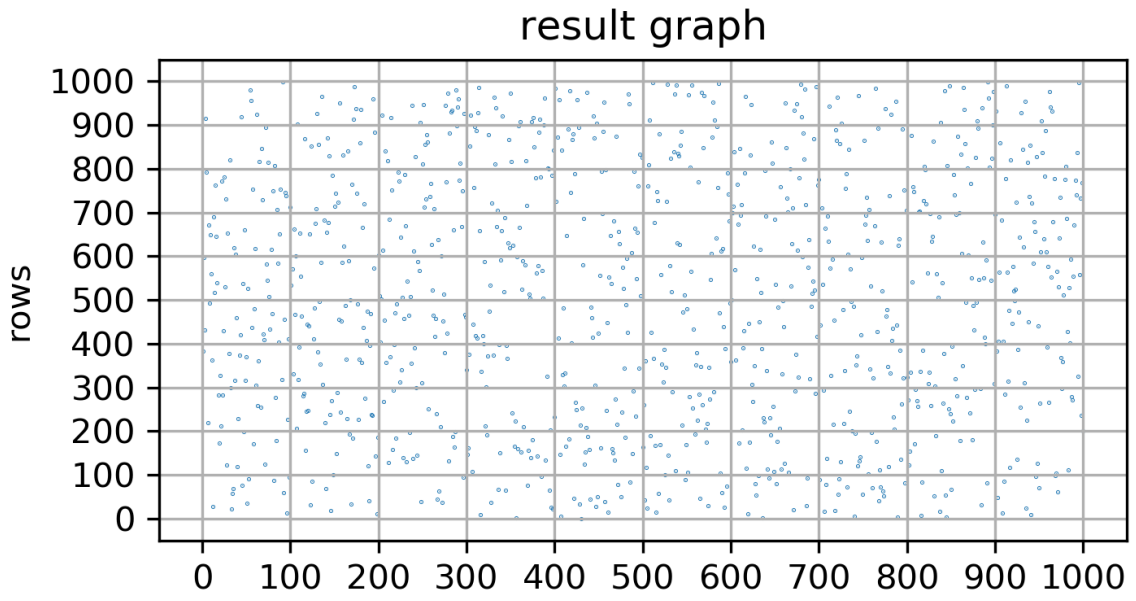

Figure 4: 1,000 Queens

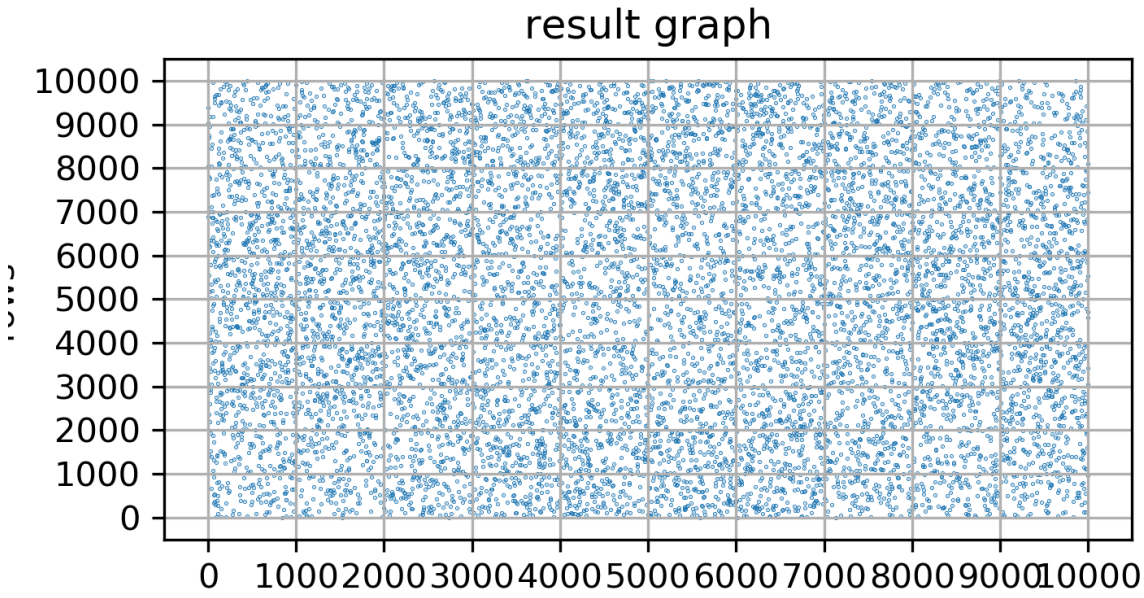

Figure 5: 10,000 Queens