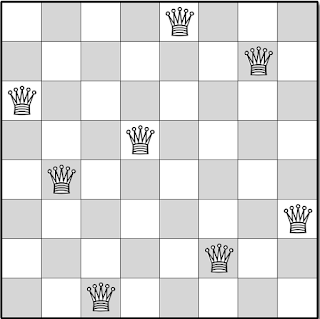
****

**N-Queens Puzzle**

Zachary Gay & Jialin Wang

*University of South Florida CAP 4621 Fall 2019 Final Project*

*Abstraction*

The N-Queens is a puzzle where the goal is to place N number of queens on an NxN chessboard without any of the queens threatening each other. A queen can move any number of squares vertically, horizontally, or diagonally. So if a queen is placed on the same row, column or diagonal as another then the solution is invalid. Solutions exist for all natural numbers N with the exception of N = 2 or N = 3. While there are many different ways to represent and solve this problem, the most efficient method is to model it as a constraint satisfaction problem (CSP) using a minimum-conflict heuristic.

**1 Description of your system**

Our system contains one stand alone program which is written in C++ and can be compiled with the standard g++ compiler using the command ‘*g++ \*.cpp*’. The program accommodates two different algorithms for solving the N-Queens puzzle; a Backtracking algorithm & a Local Search algorithm.

When the program is executed, it displays a welcome message and prompts the user to enter a positive integer, which will be assigned to the number of queens to solve for and the dimensions of the chessboard. The user is then prompted to select which algorithm they would like to use to solve the puzzle. The user can enter ‘1’ for the Backtracking algorithm or ‘2’ for the Local Search algorithm. Both of these inputs are checked to ensure a valid number was entered. If anything else was entered by the user the program will loop until an accepted integer is entered. Once all the inputs are entered and validated, the program will execute the corresponding algorithm and calculate a solution.

The Backtracking algorithm implements a two dimensional array of size N x N as its data structure in order to keep track of where the queens are currently located. It initializes all values in the two dimensional array to ‘0’, which shows we have a clear board with no queens placed yet. The algorithm enforces the rule that only one queen can be placed per column in order to recursively iterate through the data structure from left to right. During each recursive call, every row in the column is tested to see if a valid move is available. If a move is valid in the column before all the rows have been tested then we place the queen by assigning a value of ‘1’ to the appropriate location in the two dimensional array and recursively move to the next column. Otherwise it will return false and backtrack to the previous column and continue to test the remaining rows for a valid move. Since we are iterating through the two dimensional array from left to right, when checking if a move is valid, it is only necessary to check the left side for attacking queens because it is known that no queens have been placed to the right side. Once all columns have been successfully traversed it must be true that all constraints are satisfied and a solution has been found.

On the other hand, the Local Search algorithm implements a one dimensional array of size N as its data structure in order to keep track of where the queens are currently placed in each row. It initializes all values in the one dimensional array to a random value, which is used to calculate the first number of conflicts. The algorithm will then select a random row to reassign its queen. Before attempting to reassign the queen, every column in the row is tested using a minimum conflict heuristic. To calculate the heuristic, since we are only keeping track of which column each queen is located in for each array element, it is only necessary to check if any of the same values are in the array or if any are diagonal from each other using absolute values. When a conflict is found, the total number of current conflicts is incremented by one. If the number of conflicts are lower at the new column rather than the current column, then the queen is assigned to the new column. This process is repeated until no conflicts exist, which it is then known all constraints are satisfied and a solution has been found.

After a solution is discovered and regardless of which algorithm was selected, a visual representation of a N x N chessboard is printed to the terminal with ‘Q’ in place of where the queens are for the solution. This provides a quick way for the user to visually check if the solution is valid.

**2 Summary of experimental results**

Although both algorithms are guaranteed to eventually find a solution for all N > 3, they each differ in many ways.

As shown in Table 1.1, the Local Search algorithm has better time and space complexities over the Backtracking algorithm. However, the Local Search algorithm is far more unpredictable due to assigning the queens in random positions at the beginning and then randomly selecting the rows to reassign the queens. This makes the algorithm a bad choice if you are desiring to print every solution to the puzzle but a good choice for solving one solution for small and large numbers of N due to its increased speed. On the Contrary, the Backtracking algorithm is ideal if you desire to print every solution due to how it iterates through all possible paths and backtracks once that path leads to no solution, but is a bad choice for larger values of N. To put into perspective on how these time complexities differ, the backtracking algorithm solves *N = 29* in ‘*106 seconds*’ while the local Search algorithm solves *N=100* in ‘*10 seconds*’.

Both algorithms are considered to be successfully implemented as they pass the original test of solving the puzzle for *N=8*, which is the size of a professional chessboard.

**3 Main conclusions**

The main conclusion constructed from this project was how using a minimum conflict heuristic can significantly increase performance when solving constraint satisfaction problems similar to the N-Queens Puzzle.

**4 Discussion of any surprising discoveries**

It was surprising to discover just how many solutions there are for a given value of N and just how quickly the number of solutions grows as N increases. For example, if N = 20 then there are ‘39,029,188,884’ solutions, and is eight times less than if N=21, which has ‘314,666,222,712’ solutions. Also, it was surprising to see just how implementing a minimum conflict heuristic and decreasing the space complexity can significantly increase the performance of the program.

**5 References**

* Algorithm Speed - Results - N-queens Solver

https://sites.google.com/site/nqueensolver/home/algorithm-results

* Backtracking Algorithm - N-queens Solver

https://sites.google.com/site/nqueensolver/home/algorithms/2backtracking-algorithm

* Eight Queens Puzzle

https://en.wikipedia.org/wiki/Eight\_queens\_puzzle

* Min-conflicts Algorithm

https://en.wikipedia.org/wiki/Min-conflicts\_algorithm