# N-Queen's Problem

Technical Document

CISC 352

Winter 2020

Professor Sidney Givigi

**Group 26**

Aubrey McLeod 20030642

Duncan Clarke 20056561

Lianne Orlowski 10204124

Taylor Jones 20020146

Jiaoqi Liu 20013693

# Introduction

The goal of this project is to solve the N-Queens problem by using an iterative repair method paired with a minimum conflicts heuristic. The n-queens puzzle originated in 1848, where *n* queens are placed on an *n* by *n* chessboard, such that no two queens are able to threaten each other. Traditional approaches to the n-queen problems have been based on backtracking, where search techniques systematically generate all possible solutions. However, this approach requires exponentially increasing time and resources for larger values of *n* and is therefore untenable beyond a certain point. In solving the N-Queens problem, we applied a min-conflict heuristic to board generation (restricting the number of possible conflicting queens greatly) followed by an iterative repair algorithm.

# Description of Algorithm

Our solution implements the QS4 algorithm described by Sosič and Gu (1994). In this algorithm the first $n-c$, pieces are placed such that no conflicts arise, while the remaining $c$ pieces are placed randomly (random rows, in their individual columns). Through this placement method, conflict among queens is minimized to only those that are derived from pieces $c$ through $n$.

After the board is generated, our iterative repair algorithm iterates through our set of conflicted pieces and attempts to swap the row of the first conflicted piece and a random other piece. If this swap resolves conflict for both pieces, then the algorithm moves on to the next conflicted piece, otherwise the swap is undone and a different random piece is selected. This is repeated until all conflicts are resolved.

### Greedy initialization using min-conflict

Internally, we represent all the queens on the board as a permutation of all the numbers between $1$ and $n$. This guarantees that no two queens will attack each other on the same row or the same column. The problem then remains to resolve any collisions among queens that may occur on the diagonals. This is important because the initial assignment had a great effect on the number of steps necessary to solve n-queens problems on the net (Minton et al. 1991).

Our initial assignment is generated using a greedy algorithm that iterates through the columns, placing each queen on the row where it diagonally conflicts with no other placed queens. A function is called to check the number of potential collisions if the selected queen were to be placed at a given row. The queen is immediately placed on the board if there are zero conflicts, otherwise the function will randomly select a different unused row and attempt to place the queens again.

To avoid working for too long on an unideal initialization, it is necessary to limit the number of steps. If no solved board is generated after $3.08 \times$ problem size iterations (Sosič and Gu), the remaining $c$ pieces are placed.

### Heuristic repair

The heuristic value used is equal to the number of queens with a possible conflict, denoted as `c`. Due to the nature of our initialization algorithm, all queens before column `c` are inherently not in conflict, and thus we only need to resolve all pieces in columns greater than `c`.

To resolve the board, the algorithm selects the first element in the set of conflicted pieces and a random other piece in the set of all pieces. If the swapping of their positions leads to neither piece being in conflict then the swap is applied, and the process is repeated with the next piece in the conflicted set (this can be seen as a more restrictive ruleset for minimum conflict, where the only valid conflict is no conflict, not the lowest). Otherwise, no action is taken, and the search continues until a valid swap is found. In this way, all `c` pieces are iterated through and resolved.

It is of course possible, to run into local optima, where no valid swap would result in no conflict. In this regard, we limit the number of possible swaps to `2(n/ln(n))`.

## Detailed Methodology

**`main()`**

The main function takes two variables as its input. The first variable is the name of the input file, here it's specified to be "nqueens.txt". The second variable is the name of the output file, which is "nqueens_out.txt". This function reads the input file, takes the integers and stores in a list in order to determine the number of queens and size of the board. After solving the solutions to the nqueens.txt list, it outputs the solutions line by line to the nqueens_out.txt file using the `generate_output_string(solution)` function, which simply generates a string to represent the solutions found.

**`solver(size)`**

This function runs the QS4 algorithm, described above. It takes a parameter called `size`, which will be the number of queens and the size of the chess board. This is read from the input file from `main()`. First it calls `init_all()` to initialize the global variables from the previous solution, then calls `final_search(initial_search(), math.ceil(2 * SIZE / math.log2(SIZE)))`. The second parameter here is used for time constraint purposes and sets a max amount of steps to take before returning the function. It is used to minimize the time searching through conflicts and getting stuck in a local optima.

**`initial_search()`**

This function uses the heuristic min conflict to place at least `n - c` queens without conflict and with at most `c` queens with conflict and returns the number of conflicting pieces. It picks a random row that has not had a queen placed yet and safely places it. Then the conflicts in that position on the board increases and if there are no other conflicts then it moves on to the next queen. Once all queens that can be placed without conflicts are placed, then it randomly assigns the remaining

conflicting queens in their unplaced rows. Finally, it returns the number of conflicting queens to `final_search(k, steps)`.

**`final_search(k, steps)`**
This function takes two parameters, `k` being the number of conflicting queens and `steps` being the maximum number of steps taken before the program runs into local optima. Essentially, `final_search` performs a version of the iterative repair. It goes through our region of conflicting queens and randomly selects one and swaps it with another queen on the board. It only completes the swap when both queens have zero conflicts for both pieces. Once the swap has proven to be successful, our unsorted space then minimizes. If the solution takes too long (has too many swaps) within our given number of steps, this means that the function has hit a local optima and it is unable to complete the solution and the function returns none.

**`total_collisions(i)`**
This function calculates the total number of collisions of a given queen and returns that value.

**`partial_collisions(i)`**
Determines the current angular collision for a given queen.

**`swap(i, j)`**
Swaps two queens in the array changing their positions.

**`swap_with_conflict(i, j)`**
This function clears the conflicts of two queens, then calls the `swap()` function to change their position, and then updates their new conflicts.

**`add_conflict(i)`**
Increments the row, column, and left and right diagonal conflicts at a given index in the pieces array.

**`remove_conflict(i)`**
Decrements the row, column and left and right diagonal conflicts at a given index in the pieces array.

**`init_all()`**
Initializes the global variables from previous iteration through program. This is needed when going through the input file nqueens.txt to find the solution for the next line containing a new *n*.

# References

Minton, Steve, et al. "The Min-Conflicts Heuristic: Experimental and Theoretical Results." *NASA Ames Research Center*, 1991.

Sosič, Rok, and Jun Gu. "Efficient local search with conflict minimization: a case study of the n-queens problem." *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, no. 5, 1994, pp. 661-668.

Sosič, Rok, and Jun Gu. "3,000,000 Queens in less than one minute." *ACM SIGART Bulletin*, vol. 2, no. 2, 1991, pp. 22-24.