Vectorized Method for Solving the n-queens Problem using Bohrium
Larsen, Mads Ohm

Publication date:
2016

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Vectorized Method for Solving the $n$-queens Problem using Bohrium

Mads Ohm Larsen <ohm@nbi.ku.dk>

Introduction
On a normal $8 \times 8$-chessboard we find that 8 queens can be positioned in
different legal ways. This is too many to brute force, especially as we go for larger $n$.
For the $8 \times 8$-queens problem we only have 92 distinct solutions.

Backtracking Algorithm
Since only one queen are allowed in each row and column, we can skip placing
another queen in the same column or row and column, we can skip placing
8 queens in each column, we now know we can instead compute permuta-
tions of a

\[ \text{permutations of a } n \times n \text{ chessboard}. \]

Legal configurations of the board, which can be boiled down to
placements where we only check the diagonals, but only $8! = 256$
permutations of rows.
The $4 \times 4$-"identity" chessboard, which is shown in figure 2, can be built similar to
figure 1.

For the four-queens problem there are only two solutions. These are shown in
figure 3.

These two are reflections (Figure 4) or rotations (Figure 5) of each other.

To check if this is a solution, we can sum along each column, row, and diagonal, checking if the sum of any of these are
greater than 1. If not, then we have a solution.

Gathering all the traces (the sums along the diagonals and offset-diagonals) for the $4 \times 4$ identity board we get

\[ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \]

Here we see a 4, which means that the identity board isn't a solution to the four-
queens problem. In fact, the maximum trace must be equal to 4 to be a solution.

Adding a Dimension
Now that we know how to solve one board, we can create all permutations of that
board, to solve the entire $n$-queens prob-
lem for some $n$.

We can do this, by adding a dimension to our matrices above, creating a tensor of
$4 \times 4$-chessboards.

Normally we count two different types of solutions, distinct and fundamental.
The distinct solutions do not take reflection and rotation into consideration, but
the fundamental solutions do. For $4 \times 4$ we have 92 distinct solutions, but only
12 fundamental.

Matrix Representation
One solution for the four-queens problem can be represented as the following matrix:

\[ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \]

To check if this is a solution, we can sum along each column, row, and diagonal, checking if the sum of any of these are
greater than 1. If not, then we have a solution.

Gathering all the traces (the sums along the diagonals and offset-diagonals) for the $4 \times 4$ identity board we get

\[ \begin{bmatrix} 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1 \end{bmatrix} \]

Here we see a 4, which means that the identity board isn't a solution to the four-
queens problem. In fact, the maximum trace must be equal to 4 to be a solution.

Adding a Dimension
Now that we know how to solve one board, we can create all permutations of that
board, to solve the entire $n$-queens prob-
lem for some $n$.

We can do this, by adding a dimension to our matrices above, creating a tensor of
$4 \times 4$-chessboards.

Doing so yields

\[ 4 \times 4 \times 4 = 64 \]

boards, which all needs to be checked. Generalizing for the $n$-queens problem, this would be

\[ n^2 \times n! \]

For the four-queens problem, this yields $4^2 \times 4! = 256$ traces of which we only need
to count the number of 1s. As we have seen already, there is only two solutions, and we only have two max-traces which are 1.

Note that since there is always queens in the diagonals or offset-diagonals, we
can never have a max-trace of zero value.

Using NumPy and Bohrium
With our knowledge, we can construct a simple Python program using NumPy or Bohrium to calculate how many distin-
tict solutions are to the $n$-queens problem for a given $n$. In this program we
simply list all permutations of the $n \times n$ chessboard, flip it, than trace all diag-
agonals from $-n$ to $n$ and max that ma-
trix into a 10 vector. The result is then
found by counting is in this vector.

Future Work
Unfortunately we do not currently gain any performance boost using Bohrium, however this is largely due to the cre-
ation of the permutation boards.
We are planning to implement a permuta-
tion generator into the Bohrium runtime as a streaming generator for the
GPU, and will hopefully get a performance speed-up doing so.

```python
import numpy as np
def nqueens(n):
    # Generate all permutations of n identity boards
    boards = list(permutations(np.eye(n)))
    # Rotate each board
    rotation_boards = np.array([boards, np.fliplr(boards)])
    # Calculate all the traces, from -n to n for all the boards
    n_queen = np.max(
        np.trace( 
            rotation_boards, 
            1, 
            axis1=2, 
            axis2=0 )
    )
    for i in range(n, n+1)
    )
    # Count the number of 1s in the maximum of the traces
    print np.sum(n_queen == 1), "solutions for", n, ", by", n, ", board "

nqueens(8) # => 92 solutions for 8 by 8 board.
```