N queens problem

<http://aima.cs.berkeley.edu/python/csp.html>

wiki says there are 45k solutions for n = 14. Since it's an expotential increase, you can bet it's even more for n = 15. That *will* take a while, regardless of the algorithm (even with an optimal algorithm, this is a complex problem - and yours is propably not optimal). Try for a much smaller n (say, 8).

In case anyone is wondering, I did the math. With it taking 60m, the average time to compute a single correct solution with N=15 (2,279,184 correct solutions) is 1.579513ms.

The backtracking algorithm to the N-Queens problem is a factorial algorithm in the worst case. So for N=8, 8! number of solutions are checked in the worst case, N=9 makes it 9!, etc. As can be seen, the number of possible solutions grows very large, very fast. If you don't believe me, just go to a calculator and start multiplying consecutive numbers, starting at 1. Let me know how fast the calculator runs out of memory. Fortunately, not every possible solution must be checked. Unfortunately, the number of correct solutions still follows a roughly factorial growth pattern. Thus the running time of the algorithm grows at a factorial pace.

The number of solutions can be estimated using Donald Knuth's randomised estimation method.

Starting from no queens placed the number of allowed positions for the next row is n. Randomly pick one of the positions and calculate the number of allowed positions (p) for the next row and multiple this by n and store it as the total number of solutions (total = n \* p) , then randomly choose one of the allowed positions.

For this row calculate the number of allowed positions (p) for the next row and mutiple the total number of solutions by this (total \*= p). Repeat this until either the board cannot be solved, in which case the number of solutions equals zero, or until the board is solved.

Repeat this many times and calculate the average number of solutions (including any zeros). This should give you a quick and pretty accurate approximation of the number of solutions with the approximation improving the more runs you do.

Least constraining value heuristics :

Iterating over each possible values, and backtracking if its not a solution, Order the values from the least constraining value to the most constraining value. How much does it affect the possible values of other variables.

<https://github.com/darius/aima-python-mirror/blob/master/csp.py>

<https://www.sanfoundry.com/python-program-solve-n-queen-problem-recursion/>

<https://developers.google.com/optimization/cp/queens>

<https://gist.github.com/justjkk/372468/6903af265950cf59ea33301002954024d6d7e82f>

<https://notebooks.azure.com/john-lam/libraries/aima-python/html/csp.py>

<https://cs.stackexchange.com/questions/47870/what-is-least-constraining-value>

<https://people.cs.pitt.edu/~wiebe/courses/CS2710/lectures/constraintSat.example.txt>