Various Approaches to the N-Queens Problem

Scheme Week 3

Esty Thomas

Goal

The goal of this project was to produce multiple methods written in Scheme by which one might produce solutions to the "N-Queens problem" for any N, and then to answer certain questions, mainly "How does the work grow with problem size?" The methods produced were to be a simple backtracking method and also a minimum-conflicts method. Minor questions asked include: When backtracking, which column order is most efficient? When taking a minimum-conflicts approach, is it more efficient to have a random initial state, or one created with a 'greedy' algorithm, selecting the least conflicting value for each new column? Does choosing the column with the most conflicts instead of a random column in the min-conflicts approach do anything useful?

Methods

Backtracking

The backtracking algorithm goes approximately thus: for each column, if it is not the final column, check if a queen can be legitimately placed in the next column, according to some fixed column order: either left to right (0 to N), inside out, or outside in. If so, recurse down another level using the next column as the current column. If not, see if there is any other valid position in this column. If so, try again with the new assignment. If not, fail. If this is the final column, check if it is a valid state. If yes, return. If not, fail.

Minimum-conflicts

The minimum-conflicts algorithm chooses a random column with conflicts (there is a check against simply choosing the same column over again unless it is the only one left) and then selects the row in that column that causes the least conflicts to place the queen. It does this until either a valid state is reached or it hits the provided maximum number of movements.

General

Many of the helper methods, in particular, are made up of a wrapper function and a recursive core function: the 'wrapper' initializes everything and passes it in to the recursive function, which is what does all the work.

The functions are generally split into as many sub-functions as possible in an attempt to make everything understandable.

Results

How does the work grow with problem size?

Backtracking and min-conflicts grow very differently. Backtracking grows enormously quickly, while min-conflicts grows fairly slowly.

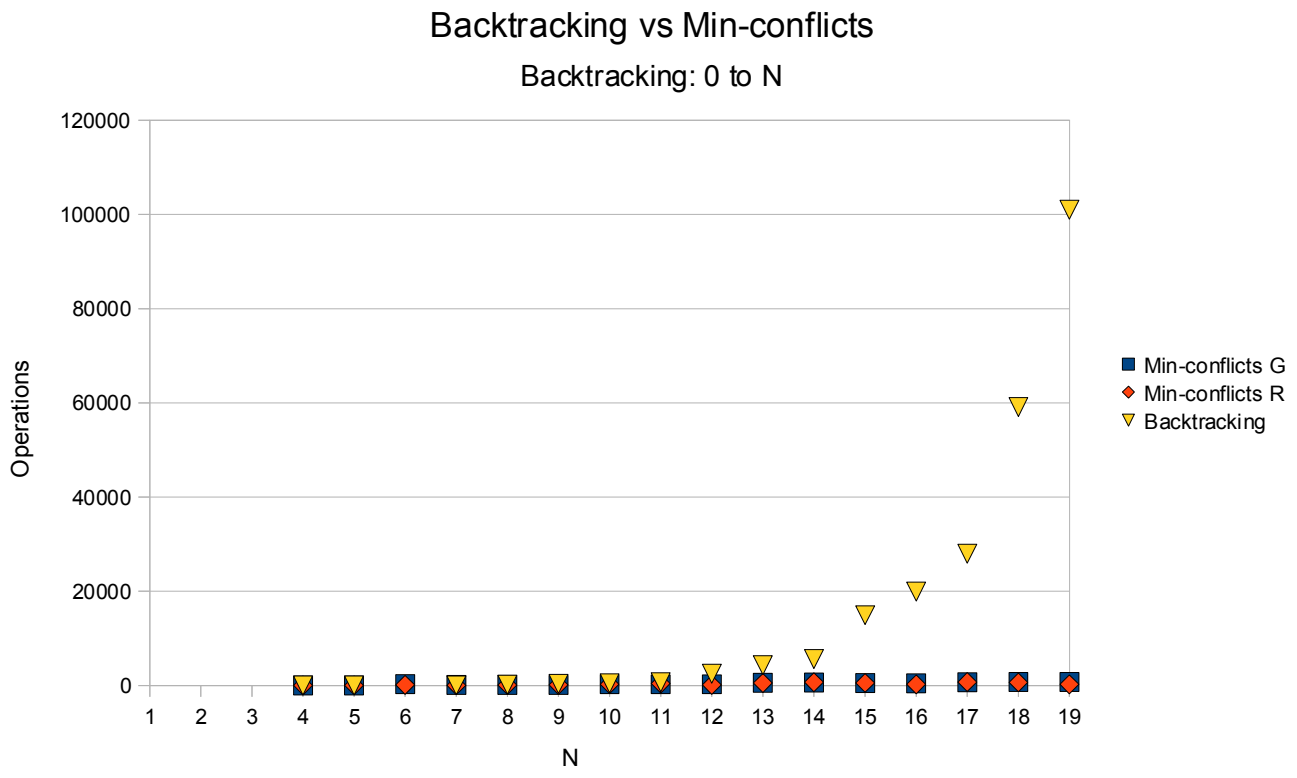


Fig. 1: Comparison of backtracking and minimum-conflicts

The graph suggests that backtracking is exponential in complexity, while min-conflicts is not. For further discussion of the relative complexities, see the Discussion section.

What column order is most efficient for backtracking?

The hypothesis presented was that inside-out would be more efficient than left-to-right, which would be more efficient than outside-in. Here is the data:

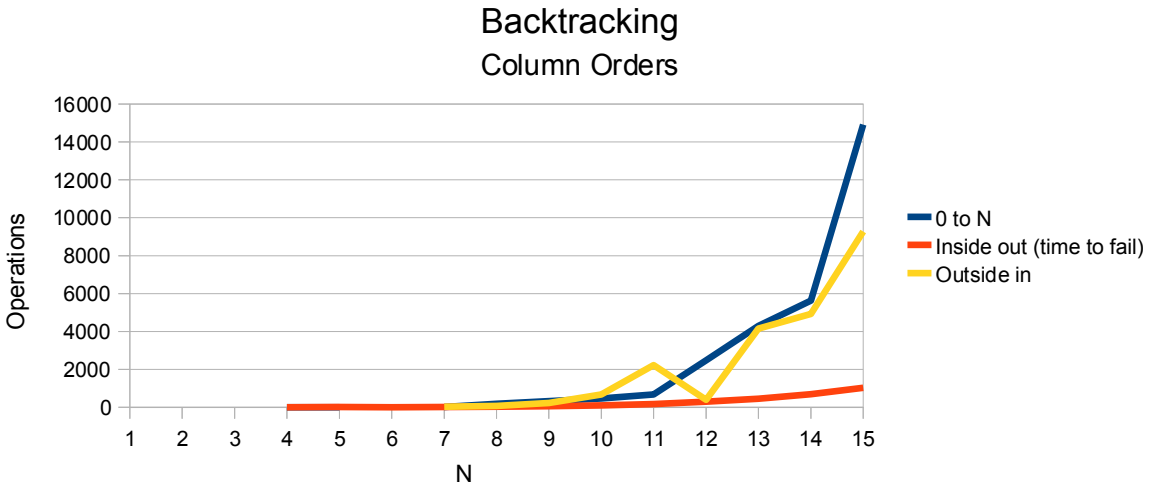


Fig. 2: Comparison of order of columns for backtracking

It is impossible to draw any conclusions about the inside-out order, because it inevitably decided that each problem had no solution, although it did take less time to fail than the other methods took to succeed. Surprisingly, and contrary to the hypothesis, the outside-in column order was overall as efficient or more efficient than the left-to-right column order. It is not clear why this should be so.

Does the initial state for min-conflicts matter?

The initial state for the minimum-conflicts approach may or may not make a difference. Two methods were tested: 'greedy' initialization, where each new column was selected to create the fewest conflicts, and completely random initialization.

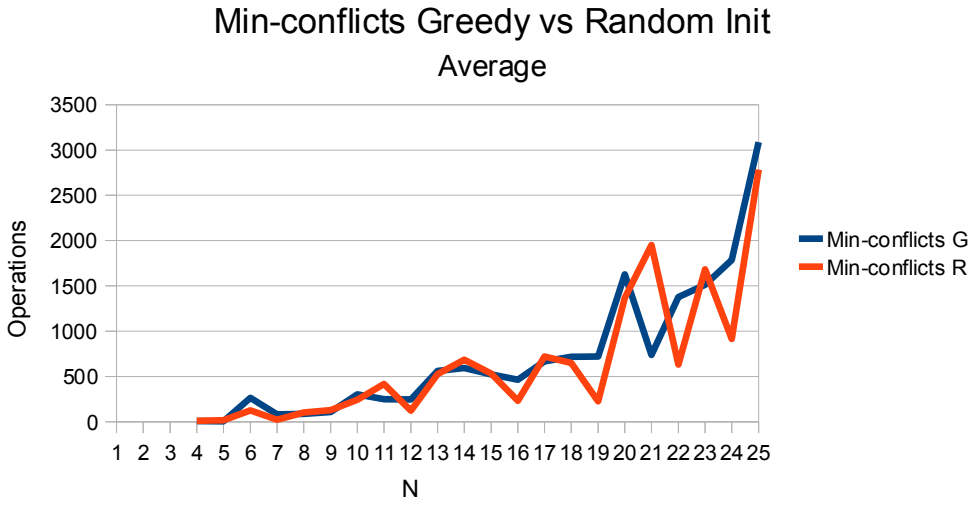


Fig. 3: Comparison of the averages (out of 6 or 4 results) for each N

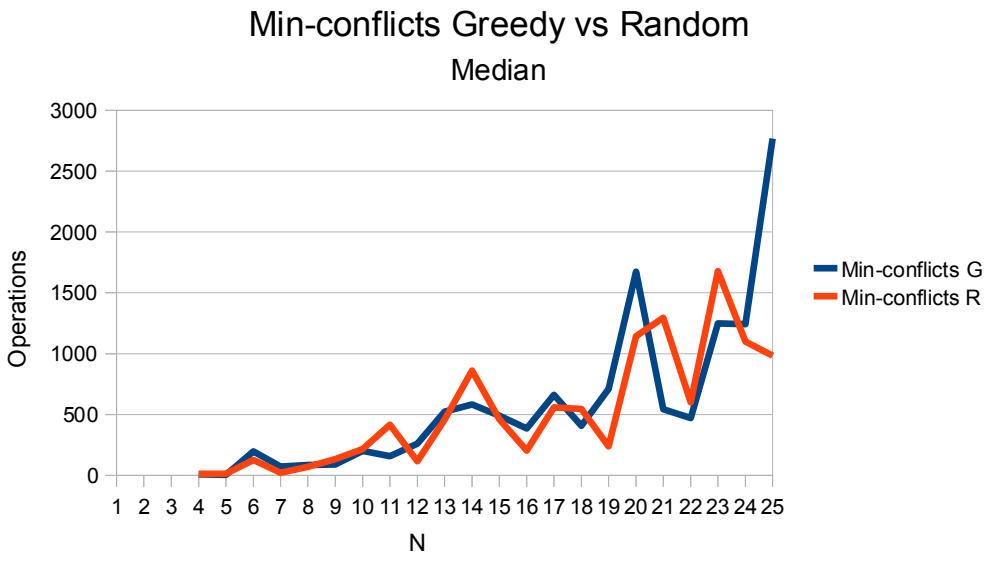


Fig. 4: Comparison of the medians (out of 6 or 4 results) for each N

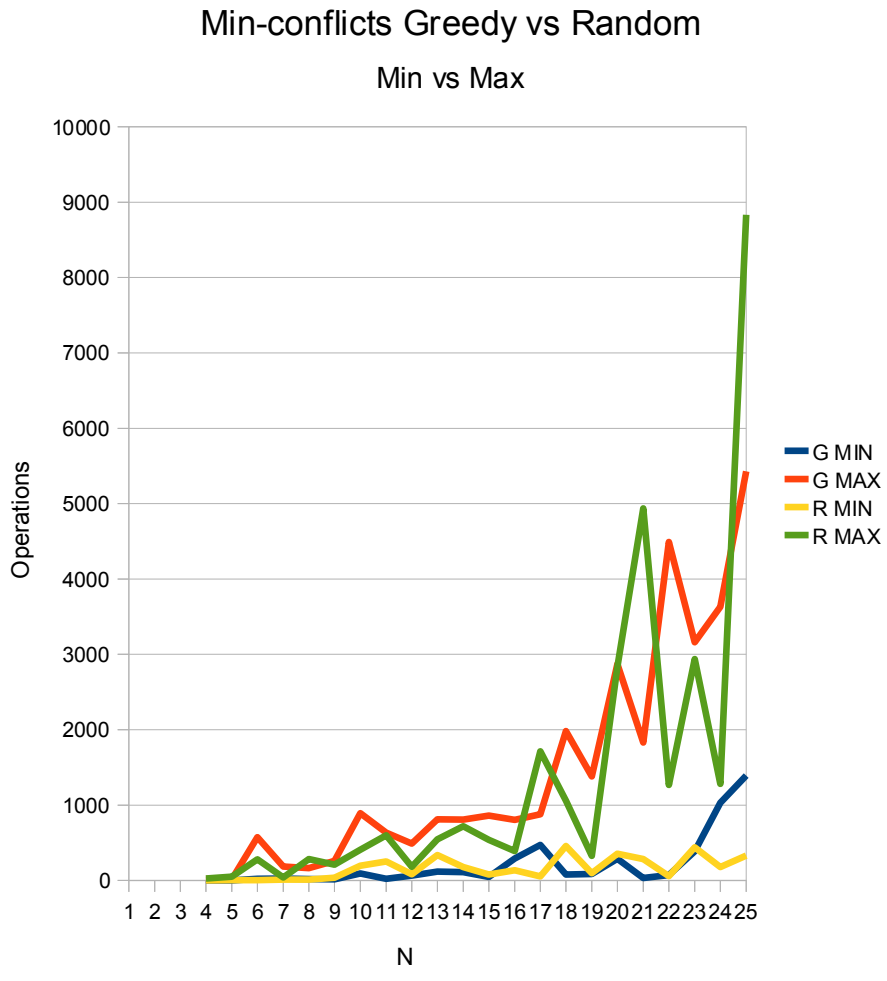
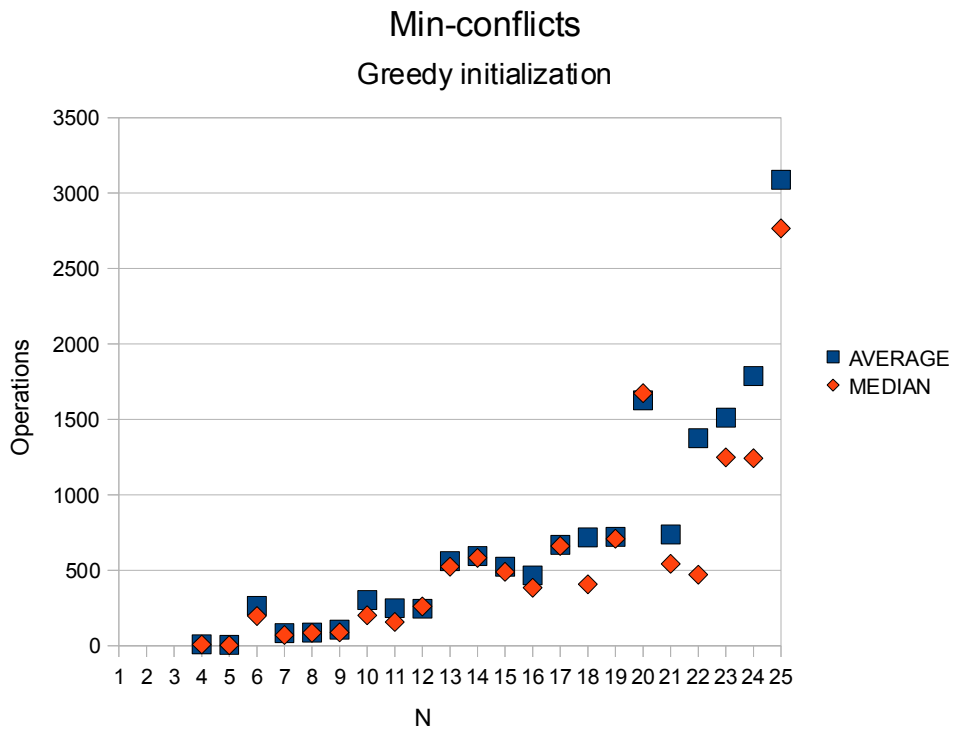


Fig. 5: Comparisons of the minimum and maximum results for each N



ig. 6: Comparison of the average vs the median for greedy initialization

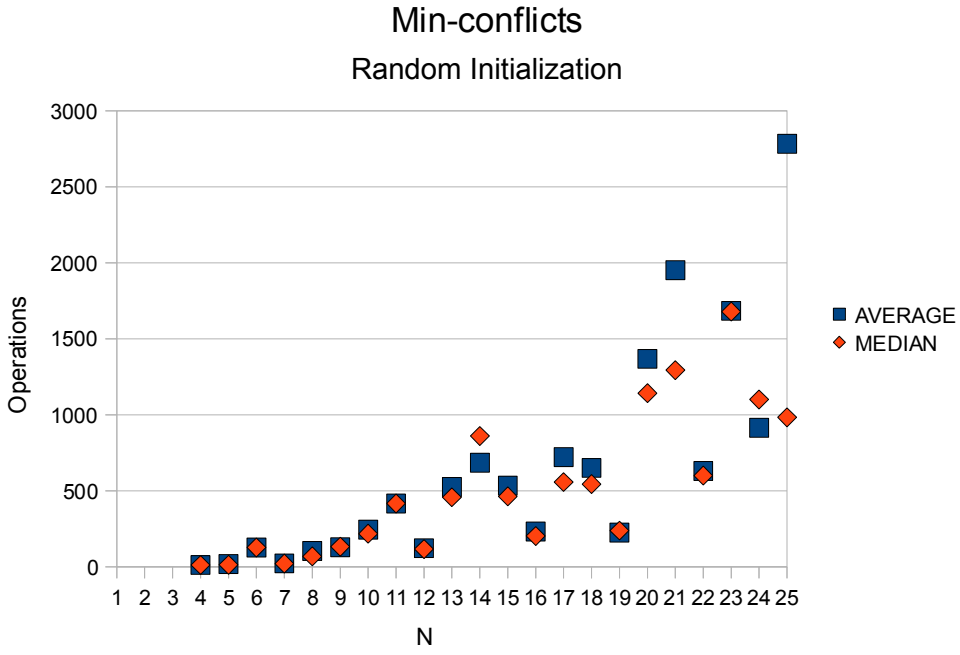


Fig. 7: Comparison of the average vs the median for random initialization

While the differences as compared to backtracking are very small, as can be seen in Fig. 1, overall the main difference seems to be that greedy initialization is more consistent, while random initialization is potentially much faster or much slower, which seems logical.

Does choosing the column with the most conflicts help at all in min-conflicts?

No. The random-column-selection was tested on the initial state #(0 1 2 3) with 50 steps as the max and gave the results (fail, fail, 8, 42, 19, 19, fail). If selecting the column with the most conflicts were helpful, one would expect it to take, on average, fewer than 50 steps. However, the results gotten were four successive failures, and the state logs indicated that it kept trying to change the same column and got nowhere. This would seem to indicate that such a device is unhelpful at best and impossible to work with at worst.

Discussion

The backtracking algorithm is obviously flawed, because the inside to outside column order fails on every attempt and the backtracking algorithm is unable to find a solution for N = 6 using any column order. Most probably, it is needlessly convoluted and needs to be rewritten, for which there is unfortunately no time. It is much improved from its original form, in which N = 4 took 91 steps, most intermediate states were repeated, and it failed more often – the problem there was not checking for the validity of a new placement before recursing further.

The question was raised of whether or not Russell & Norvig's claim that the runtime for min-conflicts is roughly independent of N, and that it can solve the 1,000,000-queens problem in 50 steps. The latter claim seems possible, given the element of randomness, but unlikely.

Complexity of backtracking and min-conflicts

In the worst case scenario, the backtracking algorithm will place n! * n queens. This suggests that it is $O(2^n)$, which is to say, of exponential complexity. This is certainly borne out by the data presented in Fig. 1.

The worst-case for min-conflicts is somewhat more difficult to work out, because the column into which it places a queen is selected randomly. The data suggest that it is $O(n^2)$. The following chart lists the maximum result divided by n^2, for reference purposes.

n	Greedy (max/n^2)	Random (max/n^2)	n	Greedy (max/n^2)	Random (max/n^2)
4	1.13	1.56	15	3.83	2.4
5	0.64	2	16	3.14	1.52
6	15.92	7.75	17	3.03	5.93
7	3.73	0.82	18	6.12	3.27
8	2.58	4.44	19	3.83	0.91
9	3.17	2.57	20	7.18	7.09
10	8.93	4.08	21	4.15	11.19
11	5.22	4.94	22	9.28	2.62
12	3.4	1.25	23	5.98	5.55
13	4.8	3.23	24	6.31	2.23
14	4.12	3.67	25	8.68	14.13

Fig. 7: Comparing max results to n^2

The only n for which the maximum result from min-conflicts was n^3 or greater was n = 6, which is an outlier in all data sets for difficulty. In the worst-case scenario, then, min-conflicts is polynomial, but in most cases it is quadratic.

Appendices

Some solutions

4 Queens

```
. . Q .
Q . . .
. . . Q
. Q . .
```

5 Queens

```
. . . Q .
Q . . . .
. . Q . .
. . . . Q
. Q . . .
```

6 Queens

```
. . . Q . .
Q . . . . .
. . . . Q .
. Q . . . .
. . . . . Q
. . Q . . .
```

7 Queens

```
. Q . . . . .
. . . . Q . .
. . Q . . . .
Q . . . . . .
. . . . . . Q
. . . Q . . .
. . . . . Q .
```

8 Queens

```
. . Q . . . . .
Q . . . . . . .
. . . . . . Q .
. . . . Q . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
```