Hunter Johnston
12/2/15
Scheme 3&4 Writeup
"N QUEENS"

Backtracking

This method of calculating the n queens solution is very systematic, tackling the problem column by column. For each column, the algorithm finds the next row in which no conflicts are created. It places the queen and then iterates to the next column. If there are no "legal" spaces remaining in that column, it returns false to the parent call of backtrack, which will then move the queen in the previous column to the next legal space and continue the algorithm.

Minimum Conflicts

This method is more random than backtracking, and it works because the solution space is evenly distributed along the problem space. It doesn't always return an optimal solution though, as it can get stuck in a local optima. This is why it is necessary to draw a line and have your program restart itself from the beginning if it has been computing for too long, or made too many queen moves. The method is given a starting board which has been greedily assigned. It chooses a random column and moves the queen in that column to the row with the least amount of conflicts (additionally, it only counts each direction as a threat once. So 2 queens in the same row threatening the queen in question only count as 1 threat). The algorithm keeps picking columns to reassign until a solution is found, or a local optima has been found. This method's runtime should be almost constant.

Minimum Remaining Value

This algorithm enhances the backtracking algorithm. when iterating through the columns, instead of choosing the next closest column to recurse into, you choose the column that has the least number of legal spaces remaining.

Sample Output

```
=> (nq-bt #t 17)

board:
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0

#(0 2 4 1 7 10 14 6 15 13 16 3 5 8 11 9 12)
number of steps: 91222
```

```
=> (nq-mc #t 10)

board:
0 0 0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1


number of steps: 1

#(8 5 3 1 7 4 6 0 2 9)
```

```
=> (nq-mc #f 100)

number of steps: 1533

#(28 5 40 77 49 52 85 12 25 58 90 27 54 26 38 48 16 73 88 86 4 29 0 13 65 83 62 93 17 84 44
53 11 79 15 74 97 34 56 89 59 76 6 42 7 1 47 9 68 81 32 45 23 87 61 99 82 30 98 10 8 91 21
78 60 33 57 94 66 41 2 36 64 31 18 50 3 24 19 67 46 14 92 35 71 80 63 96 55 75 70 20 95 69
43 37 39 51 72 22)
```

This is a sample from my attempt at mrv before i mutilated it trying to fix it. it worked for sizes 4 and 5, but there is a bug where it skips columns because of the nature of the MRV algorithm choosing the next least conflicting column. I can't get it to go back and check the unchecked columns it left behind.

```
=> (nq-mrv #t 6)

board:
0 0 0 1 0 0
1 0 0 0 0 0
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0


number of steps: 144

#(1 3 5 0 -1 4)
```

This is an example of my all-solutions-backtracking function. it prints all the solutions instead of just one.

```
=> (define testboard (vector -1 -1 -1 -1 -1))

=> (backtrack-all-sol #t testboard 5 0)

board:
1 0 0 0 0
0 0 0 1 0
0 1 0 0 0
0 0 0 0 1
0 0 1 0 0

board:
1 0 0 0 0
0 0 1 0 0
0 0 0 0 1
0 1 0 0 0
0 0 0 1 0

board:
0 0 1 0 0
1 0 0 0 0
0 0 0 1 0
0 1 0 0 0
0 0 0 0 1

board:
0 0 0 1 0
1 0 0 0 0
0 0 1 0 0
0 0 0 0 1
0 1 0 0 0

board:
0 1 0 0 0
0 0 0 1 0
1 0 0 0 0
0 0 1 0 0
0 0 0 0 1

board:
0 0 0 0 1
0 0 1 0 0
1 0 0 0 0
```

```
0 0 0 1 0
0 1 0 0 0

board:
0 1 0 0 0
0 0 0 0 1
0 0 1 0 0
1 0 0 0 0
0 0 0 1 0

board:
0 0 0 0 1
0 1 0 0 0
0 0 0 1 0
1 0 0 0 0
0 0 1 0 0

board:
0 0 0 1 0
0 1 0 0 0
0 0 0 0 1
0 0 1 0 0
1 0 0 0 0

board:
0 0 1 0 0
0 0 0 0 1
0 1 0 0 0
0 0 0 1 0
1 0 0 0 0
```

Data

```
#<procedure:timed_nq-bt>
size avg_milliseconds
4    0.1403076171875
5    0.0197021484375
6    0.273486328125
7    0.0611083984375
```

```
8    1.427978515625
9    0.8757080078125
10    2.3846923828125
11    1.3320068359375
12    10.306591796875
13    5.5557861328125
14    110.1611083984375
15    95.8731201171875
16    858.301708984375
17    539.818310546875
18    7293.266918945313
19    674.68369140625
20    51325.875
```

```
#<procedure:counted_nq-bt>
size avg_steps
4    26.0
5    15.0
6    171.0
7    42.0
8    876.0
9    333.0
10    975.0
11    517.0
12    3066.0
13    1365.0
14    26495.0
15    20280.0
16    160712.0
17    91222.0
18    743229.0
19    48184.0
20    3992510.0
```

```
#<procedure:timed_nq-mc>
size avg_milliseconds
4    0.2978271484375
5    0.0382080078125
6    2.4904052734375
7    2.3499755859375
```

```
8   2.86630859375
9   2.65556640625
10   7.2
.
.
.
90   1027.744482421875
91   704.702294921875
92   1407.769873046875
93   932.7626220703125
94   746.1849609375
95   619.868212890625
96   1058.04150390625
97   841.2904052734375
98   852.0585205078125
99   1000.2313720703125
100   1495.20048828125


#<procedure:counted_nq-mc>
size avg_steps
4   8.3
5   3.2
6   75.0
7   18.0
8   82.0
9   91.4
10   101.6
.
.
.
90   877.1
91   915.4
92   682.2
93   975.4
94   766.9
95   1037.0
96   693.5
97   1060.1
98   1173.2
99   960.2
100   1002.0
```

Discussion

The data shows that backtracking is faster for the first few trials, but quickly increases, and min-conflict increases it's runtime slowly. If it was perfect, it would not increase. min-conflict has drastically fewer steps in every case, compared with backtracking.