Assignment 5

Kim Rune Solstad

November 4, 2013

1 Introduction

For this assignment, I modified the program delivered for assignment 4. A solution where there could be more than one queen per line where attempted, but not sucsessfully created. However the program is able to solve 30 - queens problem in a couple of secounds.

2 The algorithm

The algorithm implemented in the new solution is a local-search with constraint reasoning. First, a state is generated with partially randomized queens. Each row gets K-queens. Each domain in one column is then given a value that represents the numer of conflicts a potential queen would face if placed there. The queens in the selected column is then placed in the domains with the lowest amount of potential conflicts. This process is repeated until all columns are visited. The state is then evaluated. If none conflicts are found, a winner-state is returned. If conficts are found, the process repeat starting with the first column.
The algorithm stagnates when conflicts are found, but better domains for the queens are not. To detect stagnation, a counter is increased for each state generated, (that is not randomized), without decreasing the amount of conflicts. When this counter reach the max-value, 6 in this case, the state gets randomized.

3 Instructions

Compile all files in src with javac.
' javac *.java '
Run Test.class with java. Use option 'a' for algorithm. Use option 'n' to decide amount of queens. Use 'v' to display extra info.
' java Test {[option]}'
To run with 16 queens:
' java Test a n16 '

4 Output

4.1 8-queen

Example output from 8-queen puzzle:

```
Started with state:
...*....
*.....**
....*...
........
........
.*......
.....*..
..*.....
```

```
solving:
number of conflicts for each domain in the first column:
|1  |3  |1  |1  |1  |2  |3  |2  |
state after movement:
...*....
......**
....*...
........
*.......
.*......
.....*..
..*.....
```

```
solution:
..*.....
....*...
.......*
...*....
*.......
......*.
.*......
.....*..
```

Average time to complete 8-queen puzzle is 0.278 sek.

4.2 16-queen

Example output from 16-queen puzzle:

```
Started with state:
................
......*.........
...*.......*....
........*...*...
................
................
```

```
.........*....*.
................
***.......*.....
................
....*...........
.....*..........
................
.......*........
.............*.*
................
```

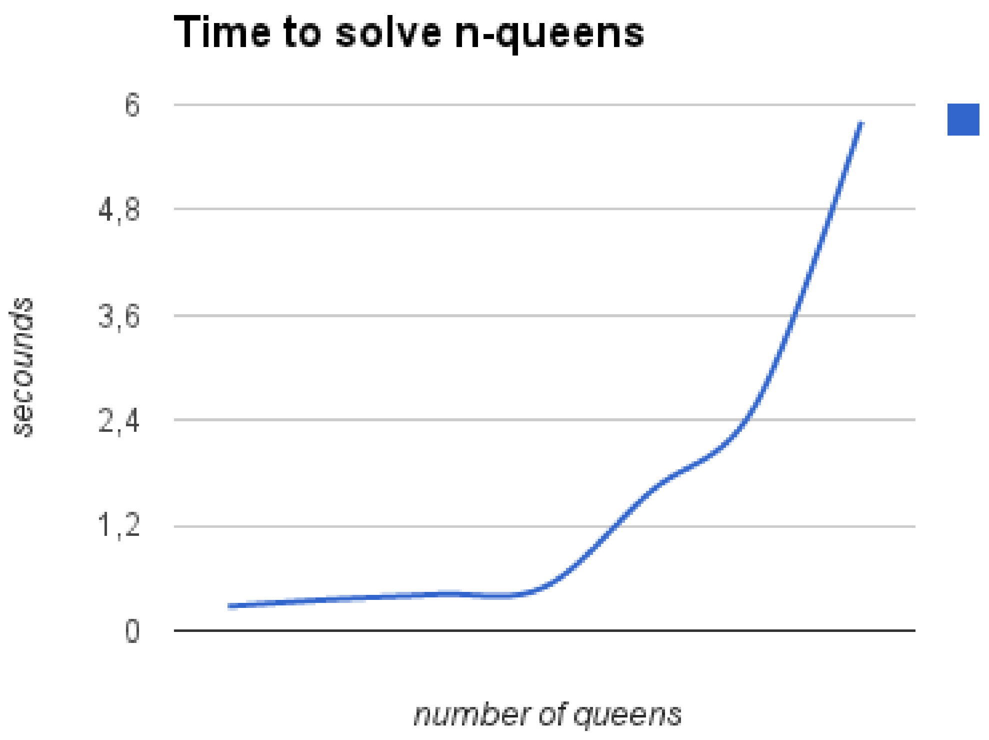
```
solving:
number of conflicts for each domain in the first column:
|0 |2 |2 |2 |0 |1 |6 |2 |3 |1 |2 |2 |0 |2 |3 |2 |
state after movement:
................
......*.........
...*.......*....
........*...*...
................
................
.........*....*.
................
.**.......*.....
................
....*...........
.....*..........
*...............
.......*........
.............*.*
................
```

```
solution:
........*.......
............*...
....*...........
......*.........
.*..............
...........*....
.....*..........
.......*........
.............*..
...*............
*...............
...............*
..........*.....
..............*.
.........*......
..*.............
```

Average time to complete 16-queen puzzle is 0.416 sek.

5 Plot

The graph shows average time spent on solving puzzles with different number of queens. 32 queens took an average of 6 secounds.



6 Changelog

1. Egg randomizing code moved from State.java to separate class called EggCartonEggRandomizer.java.
2. New eggrandomizing class created. This one places k eggs in each row of carton. This code is to be found in EggCartonRandomizer_v2.java
3. EggCartonObjectiveFunction.java has been supplemented with a method that counts obstacles in a single domain
4. MinConflictLocalSearchPuzzleSolver.java added. Constraint satisfying algorithm to solve n-queens puzzle.