Wednesday, 22 May 2019

Lab2 - del 1

Sudoku

- Depth first graph search; easy. —> Failed. 1.12.94
- Depth first graph search; medium. —> Failed. Took forever
- Backtracking search; easy —> succeed. 00.00.012
- Backtracking search; medium —> succeed. 00.00.020
- Backtracking search; harder —> succeed. 00.00.086
- AC3; easy —> succeed. 00.00.035
- AC3; medium —> failed. 00.00.033
- AC3; harder —> failed. 00.00.031
- min_conflicts; easy —> failed. 01.25.28
- min_conflicts; medium —> failed. 01.27.03

Backtracking is the best, didn't failed. Backtracking textbook example is the 8 queens puzzle (no queen should not attack each other). The solution is build upon recursion and if during the process of running the algorithm we come upon a problem that cannot be solved, we simply backtrack it one step and try another way. Backtracking solution is perfectly adapted for sudoku but maybe not much else. For the sudoku, it is all about perfect placement of the tile. One tile has one place and it depends on the surrounding.
Both min_conflicts and depth first graph search failed on the easiest, so they aren't that good.

Min_conflicts solves CSP (constraint satisfaction problem). It randomly selects a variable from a set of variables with conflicts violating CSP. It assigns to the variable the value to minimise the conflict. For this to fail, a pre-selected nr of iteration will be reached.

The depth first search is a data structure. It starts at a root and explores the graph for the solution and then backtracking. For a sudoku with a lot of different solution and with a lot of different graphs depending on the moves, the hard part would be the amount of data.

Sudoku - backtracking

- BT; easy; first-unassigned-variable; no_inference —> succeed 00.08.60
- BT; medium; first-unassigned-variable; no_inference —> failed. Took forever

- BT; easy; mrv; no_inference —> succeed. 04.34.81
- BT; medium; mrv; no_inference —> failed. Took forever

- BT; easy; mrv; forward checking —> succeed. 00.00.01
- BT; medium; mrv; forward checking —> succeed. 00.00.027
- BT; hard; mrv; forward checking —> succeed. 00.00.064

- BT; easy; first-unassigned-variable; forward checking —> succeed. 00.00.022
- BT; medium; first-unassigned-variable; forward checking —> succeed. 01.11.00
- BT; hard; first-unassigned-variable; forward checking —> succeed. 00.00.698

- BT+FC; easy; first-unassigned-variable; no_inference —> succeed 00.00.08.60
- BT+FC; medium; first-unassigned-variable; no_inference —> Failed. Took forever

- BT+FC; easy; mrv; no_inference —> failed

- BT+FC; easy; mrv; forward checking —> succeed. 00.00.01
- BT+FC; medium; mrv; forward checking —> succeed. 00.00.024
- BT+FC; hard; mrv; forward checking —> succeed. 00.00.062

- BT+FC; easy; first-unassigned-variable; forward checking —> succeed. 00.00.019
- BT+FC; medium; first-unassigned-variable; forward checking —> succeed 01.11.151
- BT+FC; hard; first-unassigned-variable; forward checking —> succeed 00.00.598

- BT+MRV; easy; first-unassigned-variable; no_inference —> Failed. 01.13.648
- BT+MRV; medium; first-unassigned-variable; no_inference —> Failed. Took forever

- BT+MRV; easy; mrv; no_inference —> failed. Took forever

- BT+MRV; easy; mrv; forward checking —> succeed. 00.00.011
- BT+MRV; medium; mrv; forward checking —> succeed. 00.00.012
- BT+MRV; hard; mrv; forward checking —> succeed. 00.00.065

- BT+MRV; easy; first-unassigned-variable; forward checking —> succeed. 00.00.019
- BT+MRV; medium; first-unassigned-variable; forward checking —> succeed 01.11.344
- BT+MRV; hard; first-unassigned-variable; forward checking —> succeed 00.00.622

————————-

- BT+MRV+FC; easy; first-unassigned-variable; no_inference —> Succeed. 00.08.544
- BT+MRV+FC; medium; first-unassigned-variable; no_inference —> Failed. Took forever

- BT+MRV+FC; easy; mrv; no_inference —> failed. Took forever

- BT+MRV+FC; easy; mrv; forward checking —> succeed. 00.00.01
- BT+MRV+FC; medium; mrv; forward checking —> succeed. 00.00.012
- BT+MRV+FC; hard; mrv; forward checking —> succeed. 00.00.047

- BT+MRV+FC; easy; first-unassigned-variable; forward checking —> succeed. 00.00.019
- BT+MRV+FC; medium; first-unassigned-variable; forward checking —> succeed 01.11.344
- BT+MRV+FC; hard; first-unassigned-variable; forward checking —> succeed 00.00.59

For this, forward checking as a parameter works best across all algorithms. No interference was unstable when it came to a harder sudoku.
MRV^5 was the better option for the variable but not by alot.

————————————-

Part 2 - Queen

queensS.py **********

n = 30 -> vad händer sjukt långsam

n = 50 -> en kaffepaus senare och inget resultat

n = 10

Solution: [9, 7, 4, 2, 0, 5, 1, 8, 6, 3]

Elapsed time 0:00:00.003000

problem with n = 20

Solution: [19, 17, 15, 18, 16, 7, 5, 8, 2, 0, 3, 11, 4, 1, 12, 10, 13, 6, 14, 9]

Elapsed time 0:00:13.952000

problem with n = 25

Solution: [24, 22, 20, 23, 21, 16, 14, 12, 10, 6, 4, 1, 5, 0, 2, 19, 17, 15, 18,

11, 9, 7, 13, 8, 3]

Elapsed time 0:00:04.593000

Sammanfattning: Klarar inte att köra efter n = 26, går generellt snabbare under

20 men 25 var snabbare än 20

queensCSP.py **********

Backtracking:

problem with n = 10

Elapsed time 0:00:00.053000

Sammanfattning: n = 3 verkar vara det lägsta utan error och den kan ta n = 500 men

tar bara längre tid, verker vara linjär

---

problem with n = 30

Elapsed time 0:00:00.417000

n = 3 -> fungerar inte

AC3

n = 10 -> fail