TNM096 – Laboratory 2

Anders Glad, andgl432
&
Michaela Rabenius, micra753

Task 1

Which algorithm works (in a timely manner, say a couple of minutes max) and which doesn't? Explain your results in terms of the capabilities of the algorithms and nature of the problems.

Failed:
Depth-first search (level: easy)
AC3: (level: hard) (level: hardest)
Min conflicts: (level: easy) (level: harder)

Cleared:
Backtracking search (level: easy, Time: 0.016 seconds) (level: harder, Time: 0.015 seconds) (level: hard, Time: 0.046 seconds)
AC3: (level: easy, time: 0.031 seconds)

Answer:
The AC3 can solve the easier puzzles by reducing the domain of every variable to a single value, but this does not work for the more difficult ones.

The regular 'depth first' can search very deep without finding a solution, which is time consuming. The 'backtracking search' in contrast can if it finds non-valid values immediately go back up the tree to look for a solution in another branch, which saves time and is much quicker, even for the hardest sudoku.

DFS is slower since it will not evaluate the configurations until it reaches a filled state. Backtracking will check if the assignments are consistent before continuing down a branch.

Min-conflicts search doesn't work well with non-commutative problems. That is problems with few or only one solution, like most sudoku problems.

(b) What effect does configuring the settings for Backtracking have on the results? Try the following. Open the file aima\csp.py on line 227, and remove the comment symbol # For the option you want to use; whether BT, BT+FC, BT+MRV, or BT+FC+MRV

- Set the variable
Select unassigned variable to:
∗ first-unassigned-variable
∗ =mrv

- For each of the two options above, set the parameter inference to:
∗ No inference
∗forward-checking

Which of these settings work best for sudoku?

Answer:

Configuration: BT -> takes longer time to solve (level: easy) time: 6.626 seconds.

Configuration BT + FC -> faster, time: 0.016 seconds; (level: easy)

Configuration: BT + MRV -> much slower, cancelled for taking too long (level: easy)

Configuration: BT + FC +MRV -> fast, time: 0.016 seconds (level easy).

The best settings are those using forward checking

	Fuv + no in	Fuv + fc	Mrv + no in	Mrv + fc
BT	T: 6.626 s	0.015 s (0.0)	1 min, 53.4 s	0.015 s
BT + FC	6.6 s	0.016 s	3 min, 22.3 s	0.015 s (0.0)
BT + MRV	6.66	0.015 s	Cancelled	0.016
BT + FC + MRV	7.6 s	0.015 s	42 s	0.016

Task 2

Answer the following questions:

a. **How large can n be for each of the algorithms? Why?**

Depth-first_ n = 27. It cannot handle a higher number of queens since the number of possible solutions also increase. Therefore, the search tree grows faster than we are able to search through.

AC3: This algorithm cannot find a solution for any size. All variables (queens) value (position) differs from each state which in turn affects the constraints on the current state. Therefore, we cannot remove values from the domain.

Back-tracking: same reasoning as in depth-first, but it can handle a size of 29.

Min-conflicts: can handle enormous sizes, even larger than n=10000. This is the case since it is not dependent on searching through a search tree. Simply taking each variable and assigns it the value which return and lowest amount of conflicts.

b. **What Backtracking settings work the best? Why?**

BT+FC+MRV.

Minimum Remaining Value selects the variable with the smallest domain to receive the first and next assignment rather than a random one. This minimizes the number of inevitable dead ends which otherwise would have to be explored.

Forward Checking keeps track of the variables which have not yet been assigned and terminates the search when a variable has no consistent values left.

Together, these minimizes the number of dead ends and the time needed to explore them. Therefore, they are a powerful combination in CSPs.

c. How many steps does Min-Conflicts require to do its work?

Around 50 iterations are enough to converge to a fitting solution.

d. Compare the nature of the heuristics deployed in traditional and constraint-based problem solving.

The difference is that the traditional heuristic methods compute a certain state and compares it to a goal state.
CSP keeps one state and makes modifications to it and applies changes dependent on the given constrains. This results in no illegal moves being made.