TDT4136: Introduction to AI 2023

Assignment 3: Constraint Satisfaction Problems

Mario Jaimez Vila / 105590 / Marta Almagro / 106011

a) Our program's solution for each of the boards, as well as the number of times the backtracking function was called and the number of times it returned failure for each of the broads.

```
------------> Solving Easy
Times the backtrack function was called in board Easy: 1
Times the backtrack function failed in board Easy: 0
Sudoku solution:
7 8 4 | 9 3 2 | 1 5 6
6 1 9 | 4 8 5 | 3 2 7
2 3 5 | 1 7 6 | 4 8 9
-------+--------+------
5 7 8 | 2 6 1 | 9 3 4
3 4 1 | 8 9 7 | 5 6 2
9 2 6 | 5 4 3 | 8 7 1
-------+--------+------
4 5 3 | 7 2 9 | 6 1 8
8 6 2 | 3 1 4 | 7 9 5
1 9 7 | 6 5 8 | 2 4 3

------------> Solving Medium
Times the backtrack function was called in board Medium: 4
Times the backtrack function failed in board Medium: 0
Sudoku solution:
8 7 5 | 9 3 6 | 1 4 2
1 6 9 | 7 2 4 | 3 8 5
2 4 3 | 8 5 1 | 6 7 9
-------+--------+------
4 5 2 | 6 9 7 | 8 3 1
9 8 6 | 4 1 3 | 2 5 7
7 3 1 | 5 8 2 | 9 6 4
-------+--------+------
5 1 7 | 3 6 9 | 4 2 8
6 2 8 | 1 4 5 | 7 9 3
3 9 4 | 2 7 8 | 5 1 6

------------> Solving Hard
Times the backtrack function was called in board Hard: 16
Times the backtrack function failed in board Hard: 4
Sudoku solution:
1 5 2 | 3 4 6 | 8 9 7
4 3 7 | 1 8 9 | 6 5 2
6 8 9 | 5 7 2 | 3 1 4
-------+--------+------
8 2 1 | 6 3 7 | 9 4 5
5 4 3 | 8 9 1 | 7 2 6
```

```
9 7 6 | 4 2 5 | 1 8 3
-------+--------+------
7 9 8 | 2 5 3 | 4 6 1
3 6 5 | 9 1 4 | 2 7 8
2 1 4 | 7 6 8 | 5 3 9

------------> Solving Very hard
Times the backtrack function was called in board Very hard: 84
Times the backtrack function failed in board Very hard: 61
Sudoku solution:
4 3 1 | 8 6 7 | 9 2 5
6 5 2 | 4 9 1 | 3 8 7
8 9 7 | 5 3 2 | 1 6 4
-------+--------+------
3 8 4 | 9 7 6 | 5 1 2
5 1 9 | 2 8 4 | 7 3 6
2 7 6 | 3 1 5 | 8 4 9
-------+--------+------
9 4 3 | 7 2 8 | 6 5 1
7 6 5 | 1 4 3 | 2 9 8
1 2 8 | 6 5 9 | 4 7 3
```

We added a new board from a Sudoku application so we could check the solutions and if our algorithm was doing it right.

```
------------> Solving My example (from app 'Sudoku Club', expert mode)
Times the backtrack function was called in board My example (from app 'Sudoku Club'):
101
Times the backtrack function failed in board My example (from app 'Sudoku Club'): 71
Sudoku solution:
9 2 8 | 7 3 4 | 1 6 5
4 1 5 | 6 9 2 | 8 7 3
7 6 3 | 8 1 5 | 9 2 4
-------+--------+------
5 4 9 | 2 8 7 | 3 1 6
8 3 1 | 4 6 9 | 7 5 2
6 7 2 | 3 5 1 | 4 8 9
-------+--------+------
2 8 4 | 5 7 3 | 6 9 1
3 9 6 | 1 2 8 | 5 4 7
1 5 7 | 9 4 6 | 2 3 8
```

b) Brief comments about the results in the consistency checks (point (b) just above). For example, you could compare the performance of the algorithm on the different boards, or try to relate the results to the theory, or something else that you find relevant. What is important here is that you are able to show understanding through reflection about these results.

As we can see in the results, as the board got harder, more calls of the backtracking algorithm were needed to solve it, and the chance of failure went up. This can be explained via understanding the algorithm and the nature of the sudokus.

The algorithm starts by eliminating from the legal values the values that don't fit the constraints via the arc consistency algorithm**. Then, if there still are variables with more than one possible value, the algorithm assigns them one of their possible values and runs the AC-3 algorithm again until every variable has 1 possible value.

The increase of the number of calls just means that there were more legal values per variable or the acotation of the legal values was slower as the board got harder. On the other hand, failures of the algorithm mean that, sometimes, when a value was assigned to a variable outside the AC-3 algorithm, a situation was created in which another variable had no legal values, which is one of the challenges of hard sudokus that backtracking cannot solve because of its inability to predict future situations.

**Arc consistency is a constraint satisfaction algorithm that follows the following steps (using as an example variables A and B):

- First, it takes the constraints of both variables and puts them in a queue, making sure that they are bidirectional (if there is A = B, then there must be B = A).
- Then it examines the first constraint, let's say it's A = B, then we are looking for a value in B that can't satisfy the constraint, if there is, it is removed, and the program removes the constraint from the queue.
- In the second round it does the same thing but with the addition the if any value of any variable is removed, the queue is refilled with every arc that is not already on the queue referring to those variables, even if it had already been checked them because now the values have changed.
- This process until the queue is empty.