AI Intro - Assignment 5

Hans-Kristian Bruvold, Thuvarahan Yogalingam

November 2015

1 Introduction

The assignment 5 of TDT4136 AI Intro is about implementing constraint satisfaction problem solver algorithm. This algorithm will consist of backtracking search and AC-3 algorithm. The algorithm will solve four different sudoku boards.

2 Methodology

To implement this algorithm we started of by using the skeleton code provided to us and then implemented the backtrack, inference and revise code by using the pseudocode described in the book. The solve_sudoku.py will use the algorithm on all sudoku boards and print the result.

2.1 AC-3

The AC-3 algorithm will check the state of the problem and reduce the size of domains as much as possible only by removing a value if it does not fit into any constraints between current variable and a neighbouring variable. The algorithm will continue doing this until all values in domains of variables fit into some constraint for that variable.

2.2 Backtrack

The backtrack algorithm will run AC-3 and then make a choice if a solution is not found by AC-3. The choice will set a variable to one of the values in its domain and run AC-3. If successful it will make another choice and do the same again, or if a solution is found it will return the solution. If AC-3 fails, it means that the choice was not a choice that is suitable for the final solution. It will then go back to a previous choice and make another choice.

3 Result and Discussion

This is the results from running the algorithm on the boards

3.1 easy.txt

```
8 1 7 | 6 9 3 | 2 5 4
5 2 3 | 7 4 1 | 9 8 6
9 6 4 | 5 2 8 | 3 1 7
------+-------+------
6 7 1 | 8 5 9 | 4 2 3
3 4 8 | 2 6 7 | 1 9 5
2 9 5 | 1 3 4 | 7 6 8
------+-------+------
7 5 6 | 3 1 2 | 8 4 9
1 3 9 | 4 8 5 | 6 7 2
4 8 2 | 9 7 6 | 5 3 1
backtrack count:  1
backtrack fail count:  0
```

Backtrack count is 1 and backtrack fail count is 0. This means that the sudoku was so simple to solve that one run of AC-3 algorithm was enough to find a solution.

3.2 medium.txt

```
6 3 5 | 9 2 7 | 1 4 8
4 8 2 | 1 6 5 | 9 7 3
9 7 1 | 3 8 4 | 2 6 5
------+-------+------
5 2 9 | 7 1 6 | 3 8 4
8 4 6 | 2 9 3 | 5 1 7
7 1 3 | 5 4 8 | 6 9 2
------+-------+------
2 9 4 | 8 5 1 | 7 3 6
1 6 7 | 4 3 2 | 8 5 9
3 5 8 | 6 7 9 | 4 2 1
backtrack count:  1
backtrack fail count:  0
```

The backtrack count and fail count are the same as in easy.txt. The same reason applies here too.

3.3 hard.txt

```
8 9 2 | 3 5 1 | 7 6 4
1 3 4 | 8 7 6 | 5 2 9
5 7 6 | 4 9 2 | 3 1 8
------+-------+------
7 1 5 | 6 2 9 | 4 8 3
4 6 3 | 5 1 8 | 2 9 7
2 8 9 | 7 4 3 | 6 5 1
------+-------+------
3 5 1 | 2 8 4 | 9 7 6
9 4 7 | 1 6 5 | 8 3 2
6 2 8 | 9 3 7 | 1 4 5
backtrack count:  3
backtrack fail count:  0
```

On hard.txt the backtrack count is 3 without any failure returns. This means that the algorithm had to choose values from the domain 3 times before it found a solution. If a human were to solve the sudoku, one would assume that he would have to make around 3 guesses before a solution was found.

3.4 veryhard.txt

```
3 4 7 | 9 6 1 | 8 2 5
6 5 8 | 4 2 3 | 9 7 1
9 1 2 | 5 7 8 | 4 3 6
------+-------+------
5 3 4 | 2 9 7 | 6 1 8
7 6 9 | 1 8 4 | 2 5 3
2 8 1 | 3 5 6 | 7 9 4
------+-------+------
1 7 6 | 8 3 2 | 5 4 9
8 9 3 | 7 4 5 | 1 6 2
4 2 5 | 6 1 9 | 3 8 7
backtrack count:  14
backtrack fail count:  9
```

he

On veryhard.txt the backtrack count is 14 and with 9 failure returns. This means that the algorithm had to do several guesses and on several occations had to go back to a previous state of the board before making another guess. A human would have a lot of trouble tying to solve this sudoku. It would require lots of guesses and possible several retries before a solution is found.