CSC384 Assignment 1 : Sudoku

Introduction

Sudoku is the Japanese word for “single numbers”, and refers to the newest numerical puzzle game that has become popular in newspapers and game publications all over the world. Although the rules for this puzzle are simple to understand, the solutions can range from the very simple to the agonizingly difficult. This is why it is the perfect domain for the first assignment of the course.

	6		2		4		5	
4	7			6			8	3
		5		7		1		
9			1		3			2
	1	2				3	4	
6			7		9			8
		6		8		7		
1	4			9			2	5
	8		3		5		9	

Puzzle Background

The Sudoku puzzle is a 9x9 grid of squares, some of which contain number values from the start. Your goal is to add additional digits so that each row, column and 3x3 square contains the digits 1 to 9, inclusive.

Solving Sudoku

The brute force method of solving Sudoku involves a pencil and a large eraser. Filling in the squares one at a time with number values, you continue until one of the rows, columns or 3x3 squares contains a duplicate value. This method can take a while, and is NP-complete for the generalized nxn puzzle case.

More elegant techniques exist of course, usually involving heuristics of some sort. In most cases, Sudoku puzzles have a unique solution, so with careful analysis of the numbers in the grid, the solution can be determined without a single use of the eraser.

Design

Objective

To create a Prolog program that takes in an incomplete Sudoku grid and returns the same grid with all the completed values. The assignment will be done in two stages:

- Brute force search: devise a complete search that discovers and returns the puzzle solution. If multiple solutions exist, return the first one found. If no solutions are possible, then indicate that as well.
- Heuristic search: building on the brute force search, add heuristics that speed up the solution process.

Work on the brute force search first, and then move onto modifying it for the heuristic search. Both should employ techniques described in lectures. Only the heuristic search version of your program is needed when handing in your final product. Hand in the brute force version only if the heuristic version could not be completed in time.

Grid Layout

Your program will read the layout for each Sudoku puzzle from a file that contains a 9x9 grid of single characters. The characters will be the digits from 1 to 9 inclusive, plus the blank character for empty squares. For example, the file contents for the grid layout above are show in the box on the right.

```
 6 2 4 5
47  6  83
  5 7 1
9  1 3  2
 12  34
6  7 9  8
  6 8 7
14  9  25
 8 3 5 9
```

Deliverables

Submit `sudoku.pl`, the Prolog file containing the program that completes the objectives. Include in this file a predicate called `solve(X,Y)`, where:

- `X` is the name of the file in which the Sudoku layout can be found
- `Y` is the name of the file into which the completed Sudoku puzzle is written

The grid layout must conform to the specifications outlined above. Other Prolog files may be submitted as well, if needed. Be sure to document your code thoroughly, as marks will be deducted for poor or unreadable designs.

Prolog files are submitted electronically at the following site:

`http://www.utm.utoronto.ca/submit`

In addition to the electronic submission, a short report (max 4 pages, 12pt font, double-spaced) will also be handed in during the class following the due date. The report will outline the general approach and heuristics used in the program design, as well as other design details you feel are pertinent for the marker to know when reading your program. Also include some basic analysis of the effectiveness of the heuristics, using the search criteria outlined in the lectures.

Mark Breakdown

40% Correctness
20% Style & Documentation
40% Report (Technique & Analysis)

Hints and Tips

- The course web page has useful links to Prolog resources, including a downloadable version of SWI-Prolog and related documentation for it.
- The website `http://www.sudoku.com` has many example grids to use in testing your design. To create grids with multiple or no solutions, either remove a value from the grid, or add an incorrect value to the grid.

Solution to example puzzle:

8	6	1	2	3	4	9	5	7
4	7	9	5	6	1	2	8	3
3	2	5	9	7	8	1	6	4
9	5	8	1	4	3	6	7	2
7	1	2	8	5	6	3	4	9
6	3	4	7	2	9	5	1	8
5	9	6	4	8	2	7	3	1
1	4	3	6	9	7	8	2	5
2	8	7	3	1	5	4	9	6