CSE 150 Programming Assignment #2

Date assigned: April 24, 2007
Date due: 11:59:59 May 3, 2007

In this programming assignment, you will be asked to solve Sudoku puzzles of various sizes. Traditionally Sudoku puzzles are 9×9 grids filled with numbers between 1 and 9 such that:

- Each digit appears once in each row.
- Each digit appears once in each column.
- Each digit appears once in each of 9 non-overlapping 3×3 boxes

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

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

Here we expand the problem somewhat to work on more general cases, such as the (simpler) 4×4 case (with 2×2 boxes) and the more complicated 16×16 case (with 4×4 boxes). (There are versions that allow non-square boxes, but we won't be generalizing to those). As is often the case in AI, it's often best to test with the small trivial cases and scale up to the harder ones (inefficient implementations may make 16×16 impossible).

Simple sudoku (.ss) files are text files that describe a Sudoku puzzle. The format is pretty simple, but there seems to be some flexibility. Blanks are marked with '.', filled in numbers with their corresponding digit. Interior lines for boxes are always marked with symbols, sometimes spaces between characters are added, and sometimes additional info is provided. Test cases will be in the first form given below, but many other formats are used on the web. Probably the simplest and most robust method is to simply ignore all lines starting with a letter and all characters that are not digits, '.', or new lines. Typically .ss files are only for 9x9 puzzles, but your program should be capable of reading any n x n puzzles where n is 4, 9, or 16. For 16x16, hexadecimal will be used. Here's a sample of the form we'll be testing with:

```
...|...|...
2..|8.4|9.1
...|1.6|32.
-----------
...|..5|.4.
```

```
8..|423|..6
.3.|9..|...
-----------
.63|7.9|...
4.9|5.2|..8
...|...|...
```

While here's another sample from the web using a different format:

```
Puzzle: X-wing031
+-------+-------+-------+
| 3 4 5 | 1 9 2 | 7 6 8 |
| 9 2 8 | 4 7 6 | . 3 1 |
| 6 7 1 | 5 3 8 | 2 4 9 |
+-------+-------+-------+
| 2 1 3 | 9 6 4 | 8 . 7 |
| 5 8 9 | 3 2 7 | 6 1 4 |
| 7 6 4 | 8 1 . | 9 2 3 |
+-------+-------+-------+
| 4 9 2 | 7 5 1 | 3 8 6 |
| 1 3 6 | 2 8 9 | 4 7 5 |
| 8 5 7 | 6 4 3 | 1 9 2 |
+-------+-------+-------+
```

Logistics:
You can work alone or in groups of at most 3. Code must be submitted (with comments) by the due date. Use of Java for this assignment is strongly encouraged – if you'd rather use something else, please discuss it with the TA (if you would like to practice Matlab (you won't need it this quarter), this is an assignment where it should work fine). A copy of your write up as a pdf should be included. Turn in the code using the turnin script.

Hard copies of write-ups are due at the beginning of class the following class, printed and stapled. Code should be included in the printout. Write-ups should include a brief description of the approach taken, answers to any questions posed, and sample output on some examples. See below for more details on the content of the write up.

Half of the points are based on the quality of the write-up and the comments. The other half are based on the performance of your code, both as reported in your write up and on a test set of problems. Bonus points may be awarded at the TA's discretion for going above and beyond the HW's description.

Points will be deducted for not following these instructions. Note that no late programming assignments will be accepted!

Project specifics

1. Write a puzzle class. At least one constructor should take the name of a file. This file will be a .ss file as described above. From this, the entire puzzle state should be formed. This should include at least the domain of each variable and your set of restraints. It should also include methods to set a variable to a value (with forward checking to remove the set variable from the domain of each other variable in row, column, and box), remove a single value from the domain of a single variable, and provide the needed info for each of the heuristics below.

   Note that all methods here will use forward checking.

   A variable's domain can be represented in a couple ways. Each variable can have an associated list of possible values. Or you can store a n x n x n matrix where (i, j, k) is 1 if the variable at (i, j) can assume value k, 0 otherwise. (If using Matlab, you'll have to use the latter.)

2. Write a blind backtracker class. It should pick a variable at random or using some blind ordering (such as left-to-right-top-to-bottom). This should be able to be run from the command line. For example, in Java to load the puzzle in samplePuzzle.txt this should be run via "java CSPSolver samplePuzzle.txt". Your program should display to the screen the following information once a puzzle is solved:
   a. Number of nodes visited
   b. Sum of the branching factors for each node visited
   c. Total execution time

3. As with (2), but select the **minimum remaining values** (aka most constrained variable). This is triggered with the flag command line flag –mrv. Just for comparison, also have a version that selects the **maximum remaining values** with the flag –xrv.

4. As with (2), but select the variable that follows the **degree heuristic**, using the flag –dh. Test it on its own, and in conjunction with –mrv (where this technique is used to break ties between variables that are equally constrained by –mrv).

5. As with (2), but once a variable is selected select the **least constraining value**. Trigger it with –lcv. Run it with the various combinations of other flags.

6. There are a number of other techniques that can improve performance. 20 points of your performance score will be based on a competition between students based on # of nodes visited (10 pts) and total execution time (10 pts). Your submission (which is the same for both competitions) should be triggered with the flag –best (which will likely include using the three above heuristics implicitly).

   The book covers a number of possible general techniques to improve performance (arc consistency, strong k-consistency, special handling of the Alldiff constraint, conflict-directed backtracking, etc.) but we encourage you to try other things as well. For instance, in general checking for strong k-consistency can be intractable for even small values of k, but there are a number of well-known patterns (check the discussion board or

Google for pointers) that capture some of these higher order dependencies and can trim your search immensely. There will likely be some trade off, as better pruning will improve # of nodes visited but time-intensive pruning will eventually start increasing runtimes.

Write up:
Please describe the approach taken and how your program should be executed. What kind of testing did you do, and how did your methods fare? Often, Sudoku puzzles are rated by difficulty, so this is a variable you may want to measure over.

Also, perform some analysis on the efficacy of the different approaches with different puzzles. Your write up should include at least one graph.

Testing:
Your programs will be run on several puzzles for each of the flags described above. Test set files will be in the first of the .ss formats described above (but we encourage you to accept the more general formats). Your own reported testing should be thorough enough that there are no surprises here.

We'll be including pointers to some .ss repositories on the discussion board. Lots of websites have downloadable .ss files, and some will even spawn novel puzzles for you on the fly.

Programs will also be run using the –best flag and the scores here will be compared by a) # of nodes visited b) the total execution time. The best program(s) on each metric will receive 10 pts, the worst 1, and the remainder spread uniformly between.

Hints:
-Start testing on small, trivial problems.
-You may want to introduce additional command line arguments for your testing. Be sure to document this.