Due Date: February 12th, 2015 at 11:59pm

Submission Instructions: Assignments are to be submitted through LEARN, in the Dropbox labelled *Assignment 3 Submissions* in the Assignment 3 folder. Late assignments will be accepted up until February 14th at 11:59pm. Please read the course policy on assignments submitted after the official due date. *No assignment will be accepted, for any reason, after 11:59pm on February 14th.*

Lead TA: Milad Khaki (milad.khaki@uwaterloo.ca). Office hours are Wednesdays 10:00-11:00 in DC2306C.

Announcements: The following exercises are to be done individually. For the programming questions you may use the language of your choice.

1. Adversarial Search [10 pts]

1. Prove the following assertion: For every game tree, the utility obtained by MAX using minimax search against a suboptimal MIN will never be lower than the utility obtained playing against an optimal MIN.

2. Can you come up with a game tree in which MAX can do better using a suboptimal strategy (that is, not using minimax search) against a suboptimal MIN? If yes, then draw the tree and explain how MAX can do better. If no, then explain why not.

2. Constraint Satisfaction [20 pts]

In a cryptarithmetic puzzle, each letter stands for a distinct digit and the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, with the additional assumption that no leading zeroes are allowed. In the above puzzle, we can represent the

$$\begin{array}{r} \text{ZEROES} \\ + \quad \text{ONES} \\ \hline \text{BINARY} \end{array}$$

constraints on the letters by a series of binary constraints: $S \neq E$, $S \neq O$, $S \neq N, \ldots$ or by *Alldiff*$(Z, E, R, O, S, N, B, I, A, Y)$. The addition constraints on the four columns of the puzzle also involve several variables and can be written as

$$\begin{aligned} S + S &= Y + X_1 \cdot 10 \\ E + E + X_1 &= R + X_2 \cdot 10 \end{aligned}$$

$$\begin{aligned} O + N + X_2 &= A + X_3 \cdot 10 \\ R + O + X_3 &= N + X_4 \cdot 10 \\ E + X_4 &= I + X_5 \cdot 10 \\ Z + X_5 &= B \end{aligned}$$

where X_1, X_2, X_3, X_4 and X_5 are auxiliary variables representing the digit (0 or 1) carried over to the next column.

Solve the above cryptarithmetic puzzle by hand, using backtracking, forward checking, and the MRV and least-constraining heuristics. Show your work.

3. Constraint Satisfaction [10 pts]

Suppose you have a relation $v(N, W)$ that is true if there is a vowel (one of: a, e, i, o, u) as the Nth letter of word W. For example, $v(2, \text{cat})$ is true because there is a vowel ("a") as the second letter of the word "cat", however $v(3, \text{cat})$ is false since the third letter ("t") is not a vowel. Note also that $v(5, \text{cat})$ is also false since there is no fifth letter in "cat".

Suppose that the domain of $N = \{1, 3, 5\}$ and the domain of $W = \{$*added, blue, fever, green, stare*$\}$.

1. Is the arc from N to W arc consistent? If so, explain why. If not, show what element(s) can be removed from the domain to make it arc consistent.

2. Is the arc from W to N arc consistent? If so, explain why. If not, show what element(s) can be removed from the domain to make it arc consistent.

4. Programming Questions [40 pts]

In the question you are asked to implement a CSP algorithm for for solving Sudoku puzzles. Sudoku is a simple game of deduction, usually played on a 9x9 grid. In the goal state, each number from 1 to 9 appears exactly once in each row and column on the grid. Additionally, the grid is subdivided into 9 3x3 non-overlapping subgrids, and each number must appear exactly once in each subgrid. A Sudoku puzzle usually starts with some numbers filled in, so that there is a single unique solution. An example of such a puzzle is given below:

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

a. [5 pts] Write a formal description of Sudoku as a CSP, giving a list of variables, their domains, and the constraints between them.

b. [15 pts] Implement a CSP solver for Sudoku using forward checking and backtracking with the Most Restricted Variable and Least Constraining Value heuristics. You should use the Most Constraining Variable heuristic as a tie breaker. Have your solver count the total number of variable assignments it makes (including backtracking).

c. [20 pts] You can find a set of Sudoku problems with different numbers of initial values in the problem.zip file included in the A3 folder on learn.

Each file contains 9 rows of exactly 9 numbers. Numbers within a line are separated by whitespace. The value 0 is used to indicate a blank space in the grid. For example, the first line of the example puzzle above would be written:

5 3 0 0 7 0 0 0 0

Run your solver on each problem instance, and plot mean number of steps against number of initial values. To avoid spending a (very) long time collecting data, **your solver should 'give up' if it needs to take more than 10,000 steps**. Describe your findings and provide an explanation for what you observe.

Submit the following

- A well documented copy of your code
- A written description of your problem representation

- The plot of the average number of variable assignments against the number of initial values the sample Sudoku problems
- A discussion of your findings and an explanation for why your program behaves as it does.