CSE 150 Programming Assignment 3

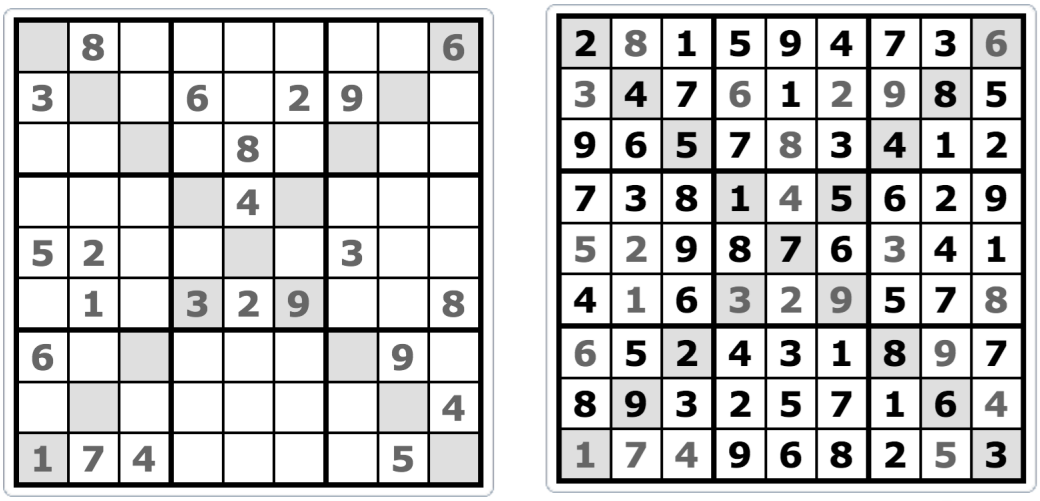
Due November 7, 2016 11:59 PM

1 Constraint Satisfaction

In this assignment, we attempt to solve a new set of problems - Constraint Satisfaction Problems (CSP). CSPs are problems where the goal is to find a solution that satisfies a set of certain constraints. The problem is defined by a set of variables, their domains and a set of constraints they need to follow. The constraints can involve one, two or multiple variables at a time. Examples of such problems are the Eight Queens Puzzle, Kakuro (Cross Sums), Map Coloring, Sudoku, Crossword etc. We explore one such problem in this assignment.

2 X - Sudoku

X - Sudoku is a variant of Sudoku. Just like Sudoku, it is usually played on a 9 x 9 grid. The goal is to fill up numbers 1 to 9 in the grid such that the rows, columns and each of the 3 x 3 grids have the numbers 1 to 9. In addition to the usual Sudoku constraints, the two diagonals should also have unique numbers. Some numbers will be given at the beginning to start off, the goal is to complete the board while satisfying these constraints.



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

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

Figure 1: An example 9x9 X-Sudoku board. Note the highlighted diagonals, they too need to be unique.

3 Provided Code

We have provided code to deal with the basic mechanics of the game, and some stub code for each problem already. In particular, the XSudoku class provides methods to parse the board and produce binary CSPs. The following is a brief description of each class provided in the assignment3.py code:

1. The `Variable` class represents a particular variable (such as X1). Each `Variable` has an associated domain, which in the case of X-Sudoku is either the integers 1 through N (the size of the board) or the number already displayed on the board.

2. The `Variables` class represents a collection of `Variable` objects with the ability to "begin transaction" and "rollback". The "rollback" method will revert any changes made to the variable domains (and assignments) that occurred since the last "begin transaction" method. You should find this method useful for implementing the backtracking search with the AC3 inference. (See problem 3 for more details.)

3. The `Constraint` class represents a binary constraint between two variables `var1` and `var2`. The constraint is satisfied (`is_satisfied(val1, val2) == True`) when the values of `var1` and `var2` (`val1` and `val2`) satisfy the relationship specified by `relation`. In the case of X-Sudoku, "not equal" (`operator.ne`) relation is used.

4. The `Constraints` are a collection of `Constraint` objects with the ability to look up constraints by the variables. **It also has the ability to return the neighbors of the node in the constraint graph, as well as the arcs involved in the constraints**. For example, to find all constraints involving a variable X_i (neighbors of X_i), you can use

   for constraint **in** constraints [x_i]:
   constraint.is_satisfied (....)

   For more details on how to use these classes, go through the comments above each of these classes in assignment3.py.

5. The `BinaryCSP` class defines a binary CSP problem, and it has the variables (a list of CSP variables) and constraints (an instance of Constraints). The `assignment()` method returns a dictionary of current variable assignments. (Note that this is provided for viewing purposes only, you probably do not need to use this method in your implementation.)

6. The `XSudoku` class has methods to parse as well as a method to produce a binary CSP (`to_binary_csp()`) and to use a CSP solver to solve an X-Sudoku puzzle `solve_with()`).

For the problems 3 and 6, the input files contain an unsolved X-Sudoku problem. In each input file, the first line contains an integer N denoting the size of the board. In the next N lines, the N x N board for X-Sudoku is represented with 0 for the "empty" cells and other numbers for pre-filled cells. The boxes are represented in the next N lines where each line represents the cells in a box. In the input, the ordering of the cells in the boxes and the ordering of the boxes do not matter. A typical initial board like the one shown below is represented as shown in input1.txt

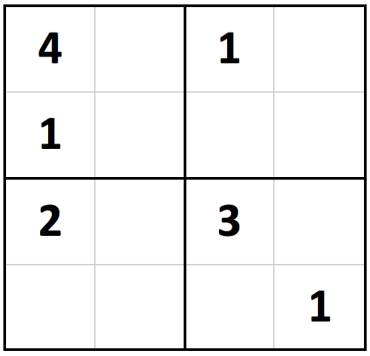


Figure 2: An example of the representation of a 4 x 4 board

input1.txt

```
4
4 0 1 0
1 0 0 0
2 0 3 0
0 0 0 1
(0,0) (0,1) (1,0) (1,1)
(0,2) (0,3) (1,2) (1,3)
(2,0) (2,1) (3,0) (3,1)
(2,2) (2,3) (3,2) (3,3)
```

Test Cases

You can also perform a subset of automated testing by running `test_problems.py` in the `tests` directory:

```
$ cd tests
$ python test_problems.py
```

This executes the test corresponding to problems 1 to 6 by giving some input in the in directories and comparing the output against ones in out directories. The tests will be reported as a "failure" if the output of your code does not match the text files the out directories. A good practice is to run the tests before doing the problems and observe that they fail. Then, once you implement the problems correctly, your tests should pass. You are encouraged to add more of your own inputs and outputs in the problems directory! If you want to run test cases individually, you can find instructions to do that in the respective test script files.

Submission

You will be submitting your assignment on Vocareum. Submit the **solutions** folder (along with all their files) as well as your **Report** in PDF on Vocareum by 11.59 PM on November 7, 2016. You do not need to submit any other files.

4 Problems

The problems that you need to complete are found in the **solutions** folder. You can import/copy the code from one file to another whenever you need to reuse the code in other problems.

Problem 1 (2 Points)

Implement the `is_complete` method that returns True when all variables in the CSP have been assigned.
Hint: The list of all variables for the CSP can be obtained by `csp.variables`. Also, if the variable is assigned, `variable.is_assigned()` will be `True`. (Note that this can happen either by explicit assignment using `variable.assign(value)`, or when the domain of the variable has been reduced to a single value)

Problem 2 (2 Points)

Implement the `is_consistent` method that returns `True` when the variable assignment to value is consistent, i.e. it does not violate any of the constraints associated with the given variable for the variables that have values assigned.
For example, if the current variable is X and its neighbors are Y and Z (there are constraints (X, Y) and (X, Z) in `csp.constraints`), and the current assignment is $Y = y$, we want to check if the value x we want to assign to X violates the constraint $c(x, y)$. This method would not not check $c(x, Z)$, because Z is not yet assigned.

Problem 3 (4 Points)

Implement the basic backtracking algorithm in the `backtrack()` method. It is "basic" in a sense that the variable ordering, value ordering and inference heuristics are not implemented yet. In other words, you only need to implement the `backtrack()` method in this problem. (But you should of course make calls to the `select_unassigned_variable()` and other methods.)
Hint: As noted earlier, you may find it necessary/useful to be able to revert any changes that have been made to the variable assignments and domain changes. To do this, you can use the transaction-inspired methods in the `Variables` class:

```
csp.variables
csp.variables.begin_transaction()
# Do whatever you need with the variables (assignment , domain reductions)
csp.variables.rollback()        # This undoes everything from the start of the transaction
```

Your backtracking search should take approximately 1 second for all the 4 test cases. Note that if you test this algorithm on a 9x9 board, backtracking search takes very long to return a solution, so we move on to improve our backtracking search.

Problem 4 (2 Points)

Implement the AC3 algorithm in the `ac3()` method. Depending on the arc parameter given, it should also act as the Maintaining Arc Consistency (MAC) algorithm described in p.218 of the textbook. When the arc parameter is empty, it performs AC3 on all the arcs in the problem. If the arc parameter has a specific value, it performs AC3 only on those arcs given in the parameter. (This will be used later in Problem 6).

Problem 5 (2 Points)

Implement the variable and value ordering heuristics in `select_unassigned_variable` and `order_domain_values` methods. For the variable heuristics, implement the minimum remaining values (MRV) heuristic using the degree heuristic as the tie-breaker. For the value ordering, implement the least-constraining-value (LCV) heuristic.

Problem 6 (5 Points)

Complete a faster backtracking search algorithm by augmenting the basic backtracking algorithm with the MAC inference and the variable and value ordering heuristics written so far. **Your improved backtracking search should take approximately 1 to 10 seconds for all 5 test cases.**

Report (8 Points)

Submit the solutions folder and a write-up for this assignment in PDF. You should include the following in the write-up:

1. Description of the problem and the algorithms used in the problems (culminating in the solution in Problem 6).
2. Run the solver you developed on different types (easy, medium, hard) of puzzles from the test cases as well as on your own, and measure how the solution time varies with the board size and difficulty ratings. Show your results in **two separate graphs** (one for the board size and another for the difficulty rating). Summarize your finding in a paragraph.
3. Analyze the effect of each type of heuristic (inference, variable and value ordering). Compared to the basic solver in p3, how much does each type of heuristic speed up the solver for this problem? For example, check the effect of adding inference, followed by inference and variable ordering, and finally inference, variable and value ordering.
4. A paragraph from each author stating what their contribution was and what they learned.

Your writeup should be structured as a formal report, and we will grade based on the quality of the writeup, including the analysis of the algorithms as well as the structure and clarity of explanations.