CIS 391/521: HW 4 - Constraint Satisfaction (Ch. 6)

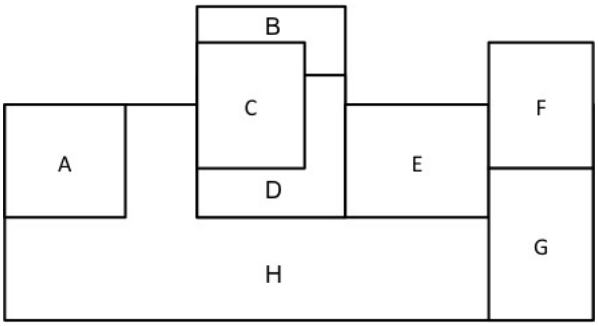
Due at 10:30 a.m. on Thursday, Feb 23rd. All submissions will be via Blackboard. There are two parts to this homework. Part 1 is short answer questions that are to be written up individually (.pdf or .txt format). Part 2 is python programming that may be completed in pairs (no more than 2 per group). For the python programming you will submit (a) answers to all the questions as described below (b) a .zip file of the code you used to generate it (c) as "solutions.txt" file as described below.

Let us know if you have any questions, check the discussion board, and remember that you can always come to Office Hours, even if only to keep the Professor and TAs company.

1 Written Portion (21 points)

1.1 AC-3 and CSPs

Suppose you are tasked with coming up with a coloring scheme for family homes. Their requirements are that no two adjacent rooms are the same color. They also want to keep the paint job relatively cheap, so they have agreed to only use 3 colors of paint: X, Y, and Z. The floor plan of the first home is below:



1. Cast this problem as a CSP, what are the variables, constraints, and corresponding domains?
2. You decide that you can solve this problem with a backtracking algorithm. Your initial heuristic will be to assign colors to rooms in lexigraphical order. And to try paint in X-Y-Z order. Draw out the resulting tree, where each node is labeled by the room that is getting assigned a color and the color it is being assigned.
3. This seems a little inefficient, especially if you were to scale this to a more complicated floor plan. You decide to try modifying your heuristic function. What would the tree be if you used most constraining variable first? Any ties are broken by lexigraphical ordering.
4. What about if you did most constraining variable combined with least constraining value?
5. What would the table be if you tried to use forward checking?
6. After hearing about the wonders of the AC-3 algorithm, you decide to run the algorithm on this question. What happens when you run AC-3 on this problem as is (all domains have all possibilities still)? Do you arrive at a full solution? Why or why not?

7. The owner of the house says that room A should be Y and room B should be color X. If we run AC-3 this time with the arc queue [(A,H),(H,A),(B,C),(C,B),(B,D),(D,B),(C,D),(D,C),(C,H),(H,C),(D,H),(H,D)...] which variables have reduced domains? What are their corresponding domains now? Which arcs are added to the queue and in what order?

2 Code Portion (20 points)

2.1 AC-3

Implement AC-3 for Sudoku in Python. Pseudocode for AC-3 as applied to a general CSP can be found on page 209 of AIMA. The specifics of the coding are up to you, but you may find it useful to start by doing something similar to the following:

- Modify the SudokuBoard class from HW1 such that the board is a dictionary, mapping each board location, such as `(0,0)`, to a set of possible values, such as `set([1,2])`. When reading in the board, this can be implemented by mapping the location of each "*" to `set(range(1,10))`, and the location of each number `x` to a singleton set, `set([x])`.
- Create a structure mapping each constraint to a set of its uncertain members. For example, for sudoku-board2.txt, this structure would initially map the top row constraint `set([(0,0), (0,1), (0,2), (0,3), (0,4), (0,5), (0,6), (0,7), (0,8)])` to `set([(0,3), (0,4), (0,5), (0,6), (0,7)])`.

Test AC-3 on the boards sudoku-board1.txt through sudoku-board5.txt. AC-3 should only be able to solve one of these boards. Report which of these boards your implementation of AC-3 was able to solve, and print its solution.

2.2 Singularity of Values

Even though the requirement that each row, column, and block contain each number exactly once is implicit in the formulation of the problem as a binary CSP, AC-3 doesn't take advantage of this. For example, if a row has only one square with the value 2 in its domain, then this square should be assigned the value 2, regardless of how many other values it currently has in its domain. AC-3 doesn't make this type of inference.

Augment your Sudoku solver (which is currently just the AC-3 algorithm) with this other type of inference. Now which of the 5 boards can it solve? For each of the boards it's able to solve, print its solution.

To make it easier for us to check your solutions, Please submit 3 files:

1. a file .pdf or .txt answering the above questions
2. a file .py or .zip with the code you wrote
3. A file called `solutions.txt`. In this file, include the solutions this Sudoku solver found. Format of the solutions should be just like that of the input board files (9 lines with 9 numbers per line, no spaces), with a blank line between each of the different board solutions.