Assignment 2

CPSC 322 Winter 2016, Term 1

Due: Friday, October 14, at 13:00

Name (last, first):
Student ID:

Partner's Name (last, first):
Student ID:

Late Days:

Please include the above information at the top of your first page; failure to do so will lead to a deduction of marks.

Assignments are to be submitted via **Connect** by the specified deadline. **We can not accept assignments sent via email or handed in in-person.**

Late Assignments

In accordance to the late policies discussed in course overview, every student is allotted four "late days", which allow assignments to be handed in late without penalty on four days or parts of days during the term. **If you want to use some of your late days** for this assignment, write the number of late days in the space provided above. No assignment can be more than two days late.

Collaboration

For this assignment, you may collaborate with *at most* one other student from this class. If you are working with someone else, you must submit **a single assignment for your pair,** writing the names and the student IDs for both of you in the space provided at the top of this page.

You may not, under any circumstances, submit any solution not written by yourself or your partner, or look at solutions of students not in your team or solutions from other sources. You may not share your own work with others outside your team.

Submission File Naming

For the non-coding questions, you will create a PDF file, and submit this PDF and your code file(s) as a ZIP archive file in Connect. Please name the submission ZIP file with your student ID or student ID of the pair similar to assignment 1 (see more details in Connect).

- Please try to start answering each question in a new page.

Question 1 (15 points) CSP representation: allocating developments

You are an architect who needs to decide how to position four new shops in a mall: a Japanese restaurant, a hairstylist, a clothing store, and a toy store. The floor plan can be represented as 3x3 grid (three rows 0,1,2 and three columns 0,1,2) and you need to place each shop in one cell of the grid (row, column). Marketing research has generated the following constraints on how to position the shops. Note that establishments are considered *close* to each other *if they share an edge on the grid*.

1) There is a fish and chips stand in cell (2,2).
2) There is a health store in cell (1,0).
3) The hairstylist and the clothing store should not be close to the fish and chips stand
4) The Japanese restaurant should be close to the health store.
5) The hairstylist and the clothing store should be close to the Japanese restaurant
6) The hairstylist and the clothing store should not be close to the toy store

[15 points] Represent this problem as a CSP. Be as precise as you can in specifying the constraints. Also do not forget some basic constraints that are inherent in allocating objects in space but are not listed above.

Question 2 (36 points + 5 bonus points)

Sudoku is a game played on a square grid of 9 large blocks, each containing its own 3x3 grid of cells. Here is an example:

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

Figure 2: An example Sudoku

The rules are simple:

- each cell contains a number from 1-9
- the number in a given cell (*i, j*) cannot match the number in any other cell in row *i* or column *j*
- each block can only contain one entry of each number from 1-9

Some cells are initially filled in and the goal is to find the unique assignment of numbers to the remaining cells that satisfies the above rules.

This game can be seen a straightforward system of constraints.

a. [8 pts] Design the CSP you will use to solve the Sudoku problem. Describe the variables, their domains and the constraints needed. We encourage you to use only **binary** constraints; it will make the implementation easier. You do not need to list all of them for this sub-question! Just specify **all** the constraints involving **one** variable; let's say the one in the center of the board.

b. [28 pts] Implement the Arc Consistency algorithm with Domain Splitting in Java and apply it to solve a game of Sudoku as a constraint satisfaction problem. We have provided a few utility classes and a class to be implemented as described below.

Download the file ***sudoku.zip*** from Connect
It contains the following files:

• SudokuUtil.java: This is a class that handles reading and writing the Sudoku boards to text files for you. The board is represented as a two dimensional array of integers.

• SudokuTester.java: This is a test script that will be used to test your code. You may want to modify it for testing/debugging of your implementation, but make sure that the code you submit works correctly with an unmodified copy of SudokuTester.

• SudokuSolver.java: This is the class where you put your implementation. Please, make sure your code is well commented and easy to read. Points will be deducted if this is not the case.

• *.sud and *Solution.sud: several sample Sudoku boards with solutions that will be used to test your algorithm. They are described in detail in SudokuTester.java.

⚠ If you are using Eclipse, the files in the "sudoku" folder should be in a package called "sudoku", otherwise comment out the package definition line (i.e., first line) in the source files.

For this question, you can assume that the given Sudoku board is valid, that is, it has exactly one solution. When testing, keep in mind that some Sudoku boards are not solvable with arc consistency alone.

A straightforward implementation is about 200 lines long. If you need many more lines of code, maybe you are not on the right path. Please make sure to have comments in your code. Marks may be deducted for lack of readability

SudokuTester.java includes time estimates for how long it should take to solve the given sample Sudoku boards. If your implementation needs significantly more time to solve these instances it is probably not correct. We will apply a cut-off time of **10 minutes** for your code to finish solving all 10 Sudoku problems.

To be submitted (via Connect): a single file ***SudokuSolver.java*** (Please, follow the instructions posted in Connect). If you decide to use additional classes, implement them as inner classes inside the ***SudokuSolver*** class. We will use a method similar to SudokuTester.java to mark your submission; therefore, your code's ability to solve a set of Sudokus is the primary metric it will be judged by.

c. Bonus [5 pts] Modify SudokuSolver.java so that it throws an exception when given an invalid Sudoku board. An invalid Sudoku board has no solutions or has more than one solution. Note that such a modified implementation may take significantly longer to solve valid boards.

To be submitted (by Connect): a single file ***RobustSudokuSolver.java***. If you decide to use additional classes, implement them as inner classes inside the ***RobustSudokuSolver*** class.

Question 3 (29 points) CSP Crossword Puzzle

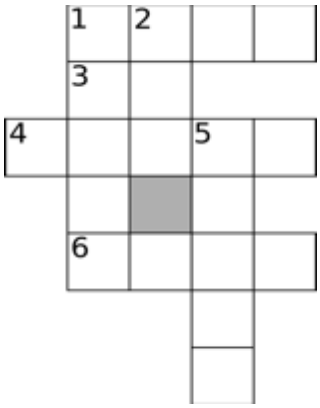

Figure 1. A crossword puzzle

Consider the crossword puzzle shown in Figure 1. The available words that can be used are:

at, eta, be, hat, he, her, it, him, on, one, desk, dance, usage, easy, dove, first, else, loses, fuels, help, haste, given, kind, sense, soon, sound, this, think

Consider the corresponding CSP in Figure 2 with nodes for the positions (1-across, 2-down etc.) and words for the domains (not all words are shown in the figure, but each domain contains all the words with the same number of letters corresponding to that node).
In this question, you will perform arc consistency with domain splitting, as well as stochastic local search, to solve the crossword.

a. [2 points] Perform arc consistency. What is the network after arc consistency has halted? Report the remaining value(s) in the domain of each variable.

b. [4 points] Use domain splitting to solve the problem. What is (are) the solution(s)? Show the domain splits you performed to reach your results, and for each domain split what the remaining variable domains were after running arc consistency.

You can use the "Consistency for CSP" applet in ***AIspace.org*** to verify your results, but you should be able to trace the algorithm by hand on paper. For use with AIspace, you can get this network in file named ***Crossword.xml*** from Connect.

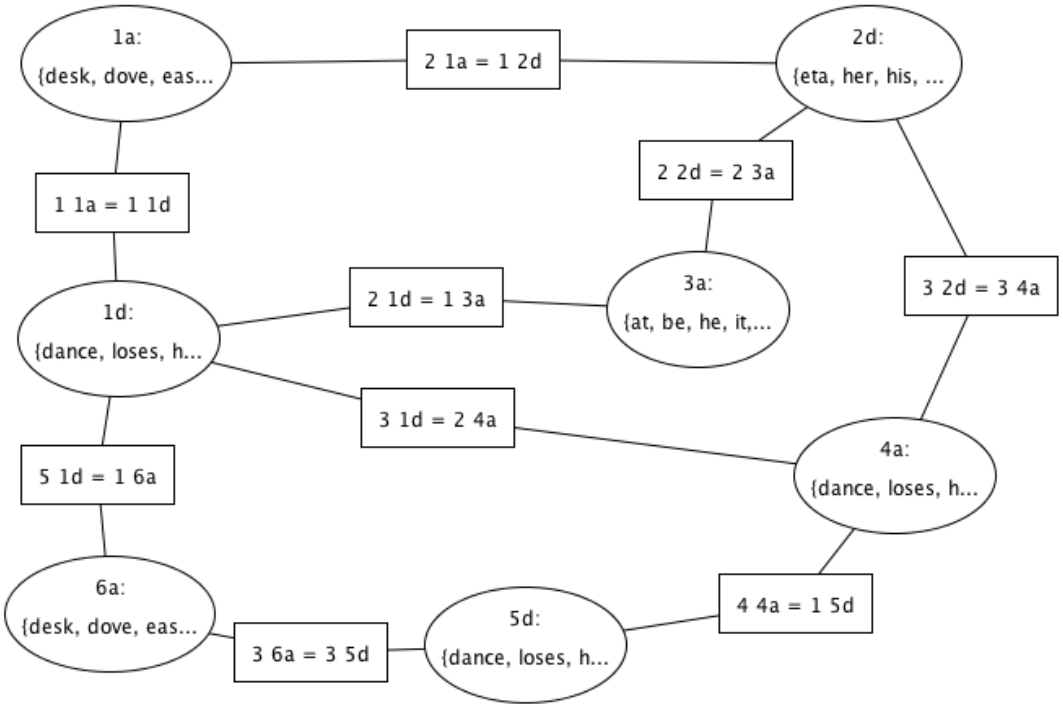

Figure 2. A CSP representation of the crossword puzzle

Now consider how stochastic local search can solve the crossword. You should use the "stochastic local search" ***AIspace.org*** applet to answer questions ***c-f*** below. Start with the arc consistent network, saved as ***CrosswordConsistent.xml***, which can be downloaded from Connect. You can make use of batch runs to get statistics, but remember to observe the actual algorithms and discuss it in your answers.

When using batch runs, change the options to terminate each run after 10,000 instead of 100 steps. Note that when the runtime distribution plot in AIspace ends before the specified number of steps that means after the point it ends none of the runs were successful (they ran up to the specified number of steps without finding a solution).

c. [3 points] Consider random walk with a one stage selection (chooses both variable and value together). How many steps does this require to solve the problem in 50% of its runs? Given a very large number of steps, would this algorithm solve the problem every time?

d. [3 points] Consider greedy descent with a one stage selection (chooses both variable and value together). How many steps does this require to solve the problem in 50% of its runs? Given a very large number of steps, would this algorithm solve the problem every time?

e. [3 points] Consider greedy descent with random walk (also using a one stage selection), with 90% greedy steps and 10% random walk steps. How many steps does this require to solve the problem in 50% of its runs? Given a very large number of steps, would this algorithm solve the problem every time?

f. [4 points] Give a set of parameters for the combination of greedy descent and random walk (using a one stage selection), that yields good results in terms of the 80% quantile of the runtime distribution (that is, a parameter setting achieving 80% successes in as few steps as possible). What is the 80% quantile of the runtime distribution for your parameter setting? Is this better than when using no random noise at all? Why or why not?