CS 4365 Artificial Intelligence
Spring 2021
Assignment 2: Local Search, Problem Encoding, and Constraint Satisfaction
Part I: *Due electronically by Sunday, March 14, 11:59 p.m.*
Part II: *Due electronically by Sunday, March 28, 11:59 p.m.*

Instructions:

1. Your solution to this assignment must be submitted via eLearning.
2. For the written problems, submit your solution as a **single PDF** file.
   - We prefer that you type your solution. If you choose to hand-write your solution, only use **blue or black pen** (black is preferred). Scan your PDF using a **scanner** and upload it. Make sure your final PDF is **legible**. **Regrades due to non-compliance will receive a 30% score penalty**.
   - Verify that both your answers and procedure are **correct, ordered, clean, and self-explanatory** before writing. Please ask yourself the following questions before submitting:
     - Are my answers and procedure legible?
     - Are my answers and procedure in the same order as they were presented in the assignment? Do they follow the specified notation?
     - Are there any corrections or scratched out parts that reflect negatively on my work?
     - Can my work be easily understood by someone else? Did I properly define variables or functions that I am using? Can the different steps of my development of a problem be easily identified, followed, and understood by someone else? Are there any gaps in my development of the problem that need any sort of justification (be it calculations or a written explanation)? Is it clear how I arrived to each and every result in my procedure and final answers? Could someone describe my submission as messy?
3. **You may work individually or in a group of two**. Only one submission should be made per group. If you work in a group, **make sure to indicate both group members when submitting** through eLearning.
4. **IMPORTANT:** As long as you follow these guidelines, your submission should be in good shape; if not, we reserve the right to penalize answers and/or submissions as we see fit.

Part I: Written Problems (50 points)

1. **Constraint Satisfaction (16 points)**

   Consider a constraint satisfaction problem where there are six variables A, B, C, D, E, and F, each with domain 1, 2, 3, 4, 5, 6. There are six constraints: A $>$ F, F $>$ E, F = D, B $>$ A, A = C, and B $>$ D.

   (a) **(8 pts)** Show how backtracking can be use to solve this problem. To select variables, use the most constrained variable heuristic, breaking ties using the most constraining variable heuristic. If ties still exist, break them alphabetically. To select values, use the least constraining value heuristic, breaking ties by preferring smaller values. Indicate the first 30 branches visited in the search tree (or stop when the solution is reached). Write them in text form with each branch on one line. For example, suppose we had variables X, Y and Z with domains 0, 1, and constraints X$\neq$Y, Y=Z. The first two branches visited in the search tree should be written as:

Y=0, X=0	failure
Y=0, X=1, Z=0	solution

   (b) **(5 pts)** Show how forward checking can be used to solve this problem. To select variables, use the most constrained variable heuristic, breaking ties using the most constraining variable heuristic. If ties still exist, break them alphabetically. To select values, use the least constraining value heuristic, breaking ties by preferring smaller values. Indicate the first 30 branches visited in the search tree (or stop when the solution is reached).

   (c) **(3 pts)** Apply constraint propagation to eliminate values from the initial domains of the variables. Show the resulting domain of each variable after constraint propagation is applied.

2. **Problem Encoding and Local Search (20 points)**

   Recall that the Pigeonhole problem concerns putting $n + 1$ pigeons into n holes. The constraints on this problem are: (1) each pigeon is assigned a hole and (2) each hole can accomodate at most one pigeon.

   (a) **(9 pts)** Assuming that the binary variable x_{ij} is *True* if and only if pigeon i is in hole j, give the clauses that encode the constraints of the problem. Make sure that each set of clauses is accompanied by an English description.

   (b) **(4 pts)** How many clauses do you need? Express your answer in big-O notation.

   (c) **(7 pts)** Describe how local search can be applied to the Pigeonhole problem, using a direct formulation (i.e., **not** using the Boolean encoding of part (a)). Discuss a way of improving the local search strategy. (Note: Since there is no solution to the Pigeonhole problem, any local search strategy will not return a solution, but that doesn't mean we cannot apply local search to it.)

3. **Adversarial Search (14 points)**

   Consider the game tree below.

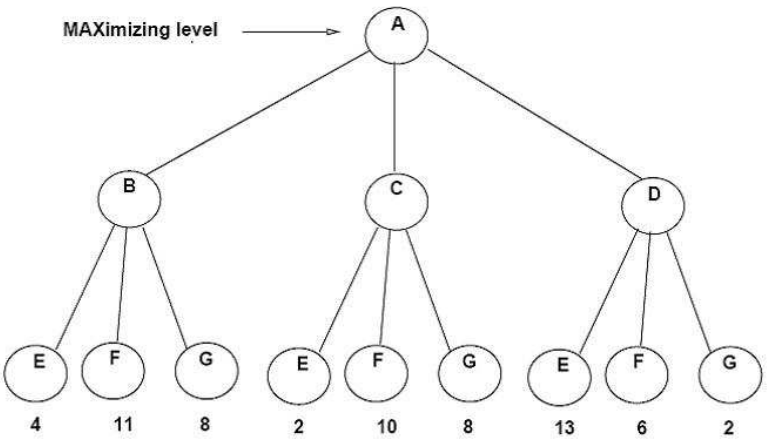

Figure 1: A minimax game tree.

(a) **(2 pts)** What should be the next move for the maximizer?

(b) **(6 pts)** Identify all the nodes that can be pruned using alpha-beta pruning. Assume a left-to-right evaluation of the nodes.

(c) **(6 pts)** Identify all the nodes that can be pruned using alpha-beta pruning. Assume a right-to-left evaluation of the nodes.

Part II: Programming (100 points)

In this problem, you will implement a CSP solver that takes exactly three arguments from the command line:

1. A `.var` file that contains the variables in the CSP to be solved and their domains. Each line of the file contains a variable (represented by a single letter), followed by a colon and its possible values, each of which is an integer. For instance, the line "A: 1 2 3" indicates that the possible values for variable A are 1, 2, and 3. Note that there is a space separating the domain values.

2. A `.con` file that contains the constraints. Each line corresponds to exactly one constraint, which involves two variables and has the form `VAR1 OP VAR2`. `VAR1` and `VAR2` are the names of the two variables involved, and `OP` can be one of four binary operators: = (equality), ! (inequality), $>$ (greater than), and $<$ (less than).

3. The consistency-enforcing procedure, which can take one of two values: `none` and `fc`. If `none` is used, no consistency-enforcing procedure is applied, and the solver simply uses backtracking to solve the problem. `fc` indicates that the solver will use forward checking to enforce consistency.

Note:

- Whenever the solver needs to choose a *variable* during the search process, apply the most constrained variable heuristic, breaking ties using the most constraining variable heuristic. If more than one variable remains after applying these heuristics, break ties alphabetically.

- Whenever the solver needs to choose a *value* during the search process, apply the least constraining value heuristic. If more than one value remains after applying this heuristic, break ties by preferring smaller values.

Your program should allow exactly three arguments to be specified in the command line invocation of your program, in the order in which they are listed above. Sample input files will be available in the assignment page of the course website. Your program should write to `stdout` the branches visited in the search tree (or stop when the solution is reached). By branches we mean an assignment that violates a constraint or that leads to one or more unassigned variables having empty domains (when using forward checking). Write them in text form with each branch on one line. For example, suppose we had variables X and Y with domains $\{0, 1, 2\}$, variable Z with domain $\{1, 2\}$, and constraint Y=Z. The branches visited in the search tree without forward checking should be written as:

1. Z=1, X=0, Y=0 failure
2. Z=1, X=0, Y=1 solution

However, if forward checking is applied, the output should be:

1. Z=1, Y=1, X=0 solution

Important: Note that the printed branches should keep a consistent variable ordering. Variables should be printed in the order they are assigned values. Do not output anything else to `stdout`. Any program that does not conform to the above specification will receive no credit. To implement the program, you may use C/C++, Java, or Python (3). You **may not** use external libraries. Your code will be run on a linux distribution, so you should make sure that it runs/compiles on the UTD Linux boxes without installing extra software. If you have any questions, post them on piazza.

Submission

Directly submit all your source files as a zip file to eLearning. Note that **you must provide an entry point** main.py, Main.java, main.cpp or main.c for your code to run (these are the only files that we will directly run when grading). If you worked in a group, make sure to add your partner when you submit!

Example Files

We will provide the corresponding .con and .var files for 3 test cases (ex1, ex2, and ex3), along with the correct output using both forward checking and no consistency enforcing procedure (fc, none). You should use these test cases to verify the basic functionality of your code and output formatting; however, remember that **we will test your code on hidden test cases**; so correctly solving ex1, ex2, and ex3 does not guarantee you will get all points.

Important note: We will be using MOSS for plagiarism detection on all programming assignments (you can look it up, it's quite powerful). Any students caught cheating will receive a 0 on the corresponding assignment and will be reported accordingly.