CPSC 322 2022W1 Assignment 2

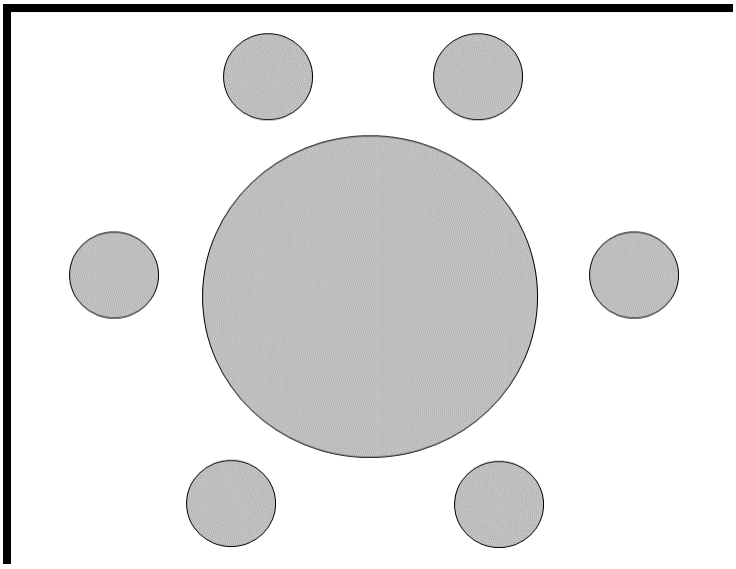
Make sure you follow all instructions in the course syllabus and in this assignment. Failure to do so may result in heavy penalties.

Make sure that in your answers you clearly indicate the exact section you are answering.

Please start each question on a new page (eg. don't put your answer to Q3 on the same page as your answer to Q2).

Question 1 [20 points] A Wedding Reception

Suppose you are given the task of assigning seating arrangements for a friend's wedding reception. You have assigned everyone to various tables, but you still need to assign each guest to a specific seat.



The last table is in the far corner, as shown in the figure above, and the guests to be seated there are:

- Uncle Joe, who is attempting to set a world record for time spent without bathing
- Aunt Marge, who constantly binge-watches her favorite shows on her phone at full volume (without headphones)
- Cousins Cara and Claude, the "Cosplay Twins", who treat every special event like Comic-Con and dress accordingly
- "Captain Sweatpants", the friend and comic book store owner who always wears sweatpants and superhero t-shirts, even on special occasions

There are six seats available at the table (note that one seat will thus be left empty). In deciding who will take which seat, you should consider some additional information:

- The other guests at the table won't want to sit next to Uncle Joe or Aunt Marge (though, as far as you know, Joe and Marge would be indifferent about sitting next to each other).
- Guests at other tables would not want to be close to Joe or Marge either.
- Cara and Claude would prefer to sit next to each other, since they're planning their costumes together (they've narrowed it down to Team Rocket or Nier: Automata).
- Captain Sweatpants would likely want to sit next to one (or both) of the Cosplay Twins.

a) **[10 points]** Represent this problem as a CSP; list the constraints and the initial domains of the variables (i.e. the domains of the variables before taking any constraints into account).

- Do not forget some common-sense constraints that are not listed above. Rather than representing your constraints as tables of satisfying variable assignments, you must represent them using mathematical and/or logical formulas (you may also define functions using pseudocode if you find that easier).
- You are not limited to unary and/or binary constraints; you may use any k-ary constraints as you see fit (in fact, it may make some things much easier if you do this).
- Rather than writing out the same functions/formulas multiple times, you may find it useful to define one or more generic functions/formulas and then refer to those functions as needed (just as you would when writing code).
- Make sure your constraints are precise and readable enough that someone could implement them without asking you for clarification (eg. what does "beside" or "next to" mean in this scenario in mathematical/logic terms?)

b) **[10 points]** Create a constraint network for this problem. If a constraint/domain is too long to fit easily in the graph, use a label in the graph instead, and indicate which constraint/domain the label refers to.

Note: while it is possible to "preprocess" at constraint network by reducing variable domains in order to reduce the number of constraints needed, you should not do this for either part (a) or (b) above; make sure to include all relevant constraints.

Question 2 [35 points] CSP - Search

Consider a CSP, where there are eight variables A, B, C, D, E, F, G, H, each with domain {1, 2, 3, 4}. Suppose the constraints are:

- $A > G$
- $|G - C| = 1$
- $D \neq C$
- $G \neq F$
- $|E - F|$ is odd
- $A \leq H$
- $|H - C|$ is even
- $E \neq C$
- $H \neq F$
- $|F - B| = 1$
- $H \neq D$
- $E < D - 1$
- $C \neq F$
- $G < H$
- $D \geq G$
- $E \neq H - 2$
- $D \neq F - 1$

a) [25 points] Use DFS with pruning to solve this problem, using the variable ordering A, B, C, D, E, F, G, H. To do this you will write (and then run) a program that

- generates the search tree (see below)
- reports all solutions (models) found, if any
- reports the number of failing consistency checks (i.e. failing branches) in the tree

You can use whatever programming language you like. Make sure to make your code as readable as possible (good variable naming, plenty of comments, good indentation, etc.).

To draw the search tree, you may want to write it in text form with each branch on one line. For example, suppose we had variables X, Y and Z with domains {t, f} and constraints X != Y, Y != Z. The corresponding search tree, with the order X, Y, Z, can be written as:

```
X=t Y=t failure
    Y=f Z=t solution
        Z=f failure
X=f Y=t Z=t failure
        Z=f solution
    Y=f failure
```

Your tree output doesn't have to follow this exact format, but it should be readable enough to allow you to check your work. **Do not include the generated search tree in your submission.**

Include all of the following in your submission:

- The solutions found
- The number of failing branches
- Your code (do this by simply copying and pasting your code into your submission document; a fixed-width font like Courier New will help the code remain readable)

If you do not include all three of these items in your submission, you will get zero marks for this question.

BONUS [10 points]: Design your program such that you can search using **any variable ordering** without having to make changes to your code. The one allowed exception is if you initialize your ordering as a hard-coded list that you then input into a program function. For example, if you initialize your list as follows:

```
ordering = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
```

then you may edit that initialization when using different variable orderings.

b) [5 points] Come up with a simple variable selection heuristic (also called a "degree heuristic" in lecture) that results in a smaller tree than in part (a), and report the following:

- Your variable selection heuristic
- A variable ordering that you obtain using this heuristic
- How many failing consistency checks there are for a tree obtained from this variable ordering.

***Note:** you are **not** being asked to find the smallest possible tree.*

c) [5 points] Explain why you expect the heuristic in part (b) to be reasonable.

Question 3 (16 points) CSP – Arc Consistency

a) [6 points] Show how arc consistency can be used to solve the problem in Question 2. You can use the AISpace Arc Consistency applet at http://aispace.org/constraint/ (also available from the AIspace Downloads page) and the XML file provided. You need to:

- Show the initial constraint graph. (You may use a screenshot.)
- **For the first 4 steps of arc consistency** (i.e. the first four arcs that you check) indicate which elements (if any) of a domain are deleted at each step, and which arc is responsible for removing the element. If no elements are deleted at a particular step, report this and state which arc was checked in that step.
- Show the constraint graph after arc consistency has stopped. (Again, a screenshot is fine.)

b) [5 points] Use arc consistency with domain splitting to solve this problem by finding all possible solutions. Draw your tree of splits and show the solutions. (It is sufficient to only label the arcs in your tree of splits; you do not need to label the nodes.)

c) [5 points] Constraint satisfaction problems can become extremely large and complex. Given the choice between DFS with pruning and arc consistency with domain splitting, what (**qualitatively**) is the expected tradeoff between the two methods in terms of time/space complexity?

Question 4 (27 points) CSP – Stochastic Local Search

Show how stochastic local search can be used for the problem in Question 2. Use the AIspace "Stochastic Local Search Based CSP solver" applet at http://aispace.org/hill/ (also accessible from the AIspace Downloads page) and the XML file provided.

AIspace instructions:

- Open the **Algorithm Options** dialog (shown in Figure 1 below), and set the search method to **Greedy Descent with All Options**. All of the settings you will need, such as variable/value selection methods, can be set here.
- Use random initialization and 2-stage selection.
- Unless otherwise specified, make sure that SLS tries 2000 steps before terminating (halting).
- Unless otherwise specified, prevent random resets from occurring. This can be done by giving the "Reset CSP every ___ steps" field a sufficiently large value.
- There are options for how frequently to use different variable and value selection methods. "Best node" means choose the variable with the largest number of violated constraints. "Random red node" means randomly choose a variable with at least 1 violated constraint. "Random node" means randomly choose any variable, whether or not it has violated constraints. "Best value" means the value resulting in the fewest violated constraints, while "random value" means any value, no matter how many/few constraints that value violates.

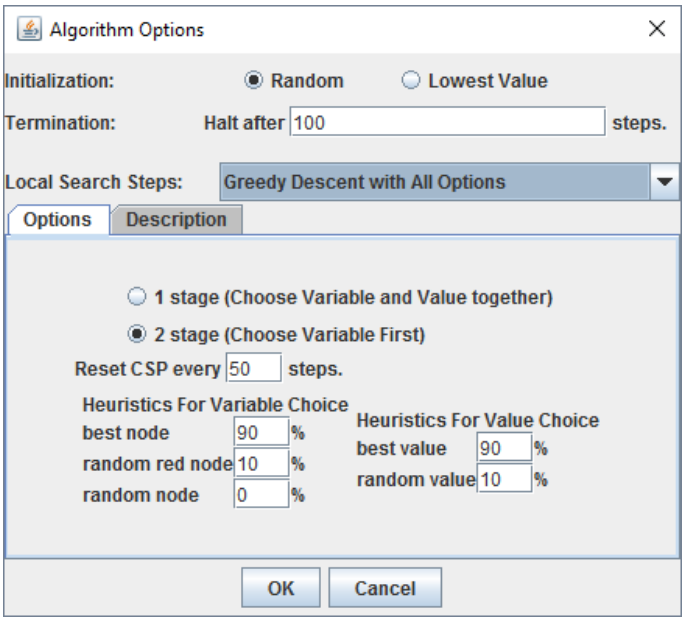


Figure 1. Algorithm options for the Hill applet in AISpace. The settings in the figure will result in an SLS that chooses a variable involved in the maximum number of unsatisfied constraints 90% of the time and any variable involved in unsatisfied constraints 10% of the time; and it will choose the best value 90% of the time and a random value 10% of the time. It will also do a random restart every 50 steps and halt after 100 steps.

a. **[5 points]** For one particular run, make SLS select any variable that is involved in an unsatisfied constraint, and select a value that results in the minimum number of unsatisfied constraints. Report the initialized values. At each step, report which variable is changed, its new value, and the resulting number of unsatisfied constraints. (You only need to do this for 5 steps). *The Show Trace function may be of use to you here.*

b. **[10 points]** Using batch runs, compare and explain the result of the following settings:
   i. Select a variable involved in the maximum number of unsatisfied constraints, and the best value.
   ii. Select any variable that is involved in unsatisfied constraints, and the best value.
   iii. Select a variable at random, and the best value.
   iv. A probabilistic mix of i. and ii. Try a few probabilities and report on the best one you find.

   Note that a plotted curve may seem to "stop", or a plot may not reach all the way to 2000 steps. This is due to the applet not continuing a plot if there are no further increases, and it is not something you need to "fix"; simply carry on with the question. To facilitate comparison, you should have (i)-(iv) plotted on the same graph, and you should indicate in your answer which plot color corresponds to which method. You may also find it useful to have a logarithmic scale on the x-axis.

c. **[4 points]** Pick one of the methods (ii)-(iv) from part (b) and run it repeatedly (5 batch runs should be enough, but you may do more runs if you wish). State which method you chose. Give a screenshot of the results; then think about, describe and explain what you observe. *(This question is intentionally vague; I want to see what insights and conclusions you draw. Clarifications will not be provided for this question.)*

d. **[4 points]** How important is it to choose the value that results in the fewest unsatisfied constraints as opposed to choosing a value at random? Justify your answer with evidence.

e. **[4 points]** For the best variable/best value method from part (b.i), allow random resets (for example, after 50 steps). Give a screenshot comparing the results to what happens if random resets are not allowed, and explain how this affects the performance of the algorithm, and why.