CPSC 322 2018W2 Assignment 2

Make sure you follow all assignment instructions on the course website and in the assignment description on Canvas. 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 [21 points] Allocating Developments Problem

CSP techniques are useful in solving complex configuration and allocation problems. You are given the task of allocating four developments in a new site in Whistler. You have to place a housing complex, a big hotel, a recreational area and a garbage dump. The area for development can be represented as 3x3 grid (three rows 0,1,2 and three columns 0,1,2) and you need to place each development in one cell of the grid. Unfortunately there are some practical constraints on the problem that you need to take into account. In the following, A is close to B if A is in a cell that shares an edge with B (but sharing a corner does **not** make two objects close).

- There is a cemetery in cell 0,0. (You do not need to treat the cemetery as a variable.)
- There is a lake in cell 1,2. (You do not need to treat the lake as a variable.)
- The housing complex and the big hotel should not be close to the cemetery.
- The recreational area should be close to the lake.
- The housing complex and the big hotel should be close to the recreational area.
- The housing complex and the big hotel should not be close to the garbage dump.

a) **[16 points]** Represent this problem as a CSP. Do not forget some basic constraints that are inherent in allocating objects in space but are not listed above. Rather than representing your constraints as tables of satisfying variable assignments, you should represent them using mathematical and/or logical formulas. You may also define functions with pseudocode and refer to them for simplicity (for example, you may wish to define a "close" function and then use that function in your answer rather than writing it out explicitly multiple times).

We want your constraints to be:

- correct
- precise and readable enough that someone could implement them without asking you for clarification (eg. what does "close" mean in mathematical terms?)
- reasonably formatted

b) **[5 points]** Draw a constraint graph 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. Include unary constraints in the graph rather than reducing variable domains.

Question 2 [40 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, A < H, |F – B| = 1, G < H, |G – C| = 1, |H – C| is even, H != D, D ≥ G, D != C, E != C, E < D – 1, E != H – 2, G != F, H != F, C != F, D != F, |E – F| is odd.

a) [25 points] Show how search can be used to solve this problem, using the variable ordering A, B, C, D, E, F, G, H. To do this you should write a program to

- draw the search tree generated (see below)
- report all solutions (models) found
- report the number of failing consistency checks (i.e. failing branches) in the tree

You can use whatever programming language you like.

To draw the search tree, 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.

Submit your search code as an appendix at the end of your PDF submission, but indicate the solution assignments found and the number of failures in your answer to the question. One way to do this is to simply copy and paste your code at the end of the submission document; a fixed-width font like Courier New will help the code remain readable. **You do not need to include the generated search tree in your submission.**

b) [10 points] Is it possible to generate a smaller tree? Come up with a simple variable selection heuristic that results in as small a tree as you can find, 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 the 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 good.

Question 3 (16 points) CSP – Arc Consistency

a) [6 points] Show how arc consistency can be used to solve the scheduling problem in Question 2. You can use AISpace and the as2csp.xml file provided. You need to:

- Show the initial constraint graph. (You may use screenshots.)
- **For the first 4 steps of arc consistency** show 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, screenshots are fine.)

b) [5 points] Use domain splitting to solve this problem. Draw your tree of splits and show the solutions. (It is sufficient to only label the arcs in your tree of splits)

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 tradeoff between the two methods in terms of time/space complexity?

Question 4 (33 points) CSP – Stochastic Local Search

Show how stochastic local search can be used for the scheduling problem in Question 2. Use the AISpace "Stochastic Local Search Based CSP solver" applet.

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.

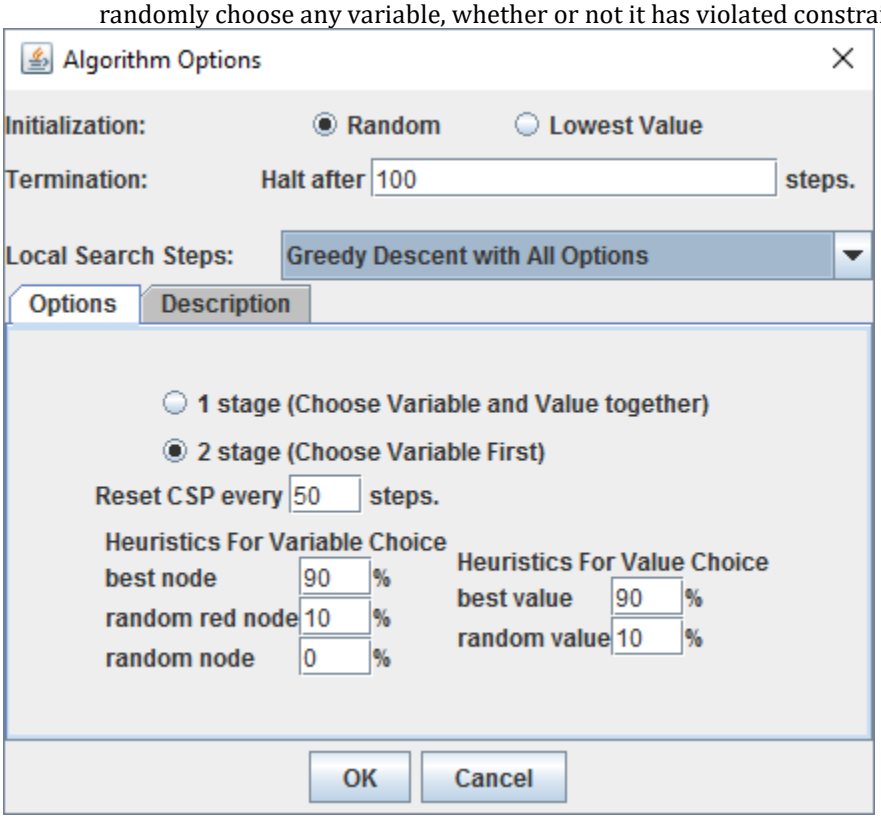

Figure 1. Algorithm options for the Hill applet in AISpace. The settings in the figure will result in an SLS that chooses a variable involved in the maximum number of unsatisfied constraints 90% of the time and any variable involved in unsatisfied constraints 10% of the time; and it will choose the best 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 arcs. (You only need to do this for 5 steps).

b. **[16 points]** Using batch runs, compare and explain the result of the following settings:
   i. **[4 points]** Select a variable involved in the maximum number of unsatisfied constraints, and the best value.
   ii. **[4 points]** Select any variable that is involved in unsatisfied constraints, and the best value.
   iii. **[4 points]** Select a variable at random, and the best value.
   iv. **[4 points]** 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 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), 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.