Assignment Two: CSPs
Due: 1:00pm, Wednesday 21 October 2015.

Some of the following problems may be easier to do if you use the AIspace.org applets and/or the Python code at [http://artint.info/code/python/aifca_322_as2.zip](http://artint.info/code/python/aifca_322_as2.zip).

This can be done in groups of size 1, 2 or 3. Working alone is not recommended. A group of size $n$ can choose any $n + 2$ questions out of questions 1 to 6. Every group should do question 7. All members of the group need to be able to explain the group’s answer. Please look at all of the questions, as the exam will assume that you have thought about all of the questions. Please post questions to the Piazza web site.

Submit your answers in class or in box #26 in room X235. You must submit a written response to the questions; it is this written response that will be marked. Use proper sentences in your answer. Code and/or AIspace representations should be submitted via handin using the directory cs322/assign2 in case the markers want to look at it.

Ask questions on Piazza. Feel free to answer them too.

Question One

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 their own 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.

- There is a cemetery in cell 0,0.
- There is a lake in cell 1,2.
- 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.

Represent this problem as a CSP. Be as precise as you can in specifying the constraints. Also there may be some basic constraints that are inherent in allocating objects in space that may not be not listed above. You need to give a human-level description of the variables, domains and constraints as well as either an AIspace CSP representation or a Python representation and demonstrate that it works. If there is a solution, give a solution (in a way that a human can understand) and tell us how many solutions there are. If there are no solutions, explain why (in a way that someone who has not taken an AI course can understand).
Question Two

Consider a scheduling problem, where there are eight variables $A$, $B$, $C$, $D$, $E$, $F$, $G$, $H$ each with domain \{1, 2, 3, 4\}. Suppose the constraints are: $A > G$, $A \leq H$, $|F - B| = 1$, $G < H$, $|G - C| = 1$, $H - C$ is even, $H \neq D$, $D > G$, $D \neq C$, $E \neq C$, $E < D - 1$, $E \neq H - 2$, $G \neq F$, $H \neq F$, $C \neq F$, $D \neq F$, $|E - F|$ is odd.

An AlSpace representation for this graph is at: [link](http://www.cs.ubc.ca/~poole/cs322/2015/as2/as2csp.xml) (copy this url into “File” —> “Load from URL” in the applets.) A Python implementation that works with the supplied Python code is available in [as2csp.py](http://www.cs.ubc.ca/~poole/cs322/2015/as2/as2csp.py) in the Python distribution referenced above.

(a) 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, write a program to write out all answers and count the number of failing consistency checks.

You can use whatever programming language you like. We don’t want a general program; just one for this example. Python code that produces the answer for variables $X$, $Y$ and $Z$ each with domains \{t, f\}, and constraints $X \neq Y$, $Y \neq Z$ is at [link](http://www.cs.ubc.ca/~poole/cs322/2015/as2/streeCount.py)

(b) Is there a smaller tree? Give a simple node selection heuristic that results in as small a tree as you can find. Show the tree, and say how many failing consistency checks there are. Explain why you expect the tree resulting from this heuristic to be good. (A good explanation as to why your ordering is expected to be good is more important than the perfect ordering.)

Question Three

Show how arc consistency can be used to solve the scheduling problem of question two. To do this you need to

(a) For the first 5 instances of arc consistency, explain which elements of a domain are deleted at each step, and which arc is responsible for removing the element. You need to explain it at a level for one of your peers to understand if this is the first time they have seen arc consistency.

(b) Show explicitly the constraint graph after arc consistency has stopped.

(c) Show how splitting domains can be used to solve this problem. Draw the tree of splits and show the solutions.

(d) Based on this experience, discuss how much arc consistency saves over the backtracking for this problem (even for the best tree that you found).

Question Four

Show how stochastic local search can be used for the scheduling problem of question two. Make sure that it tries 2000 steps before termination. You can use the AlSpace applet or the Python code.

(a) For one particular run, where you select a variable that is involved in the most conflicts, and select a value that results in the minimum number of conflicts, explain which element is changed at each step and what was the resulting number of unsatisfied arcs. (You only need to do this for 5 steps). Again, explain this at a level for one of your peers to understand if this is the first time they have seen local search.

(b) 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 which is involved in unsatisfied constraints, and the best value
iii) a probabilistic mix of the (i) and (ii), such as with probability 0.4 select a variable involved in
the most conflicts, otherwise select variable in any conflict, and select its best value. Try a few
probabilities and report on the best one found.

You must show and explain the runtime distributions and describe what you observe (in particular, tell
us which curves correspond to which settings, when one algorithm better than another, and how often
each setting does not find solutions; be specific). You should use multiple runs for each setting in the
comparisons.

(c) How important is to choose the value that results in the fewest unsatisfied arcs as opposed to choosing
a value at random? Explain.

(d) Based on this experience suggest good settings for the parameters that control which algorithms is
run (e.g., which method is chosen, and the probabilities that different choices are made). Justify your
choice.

**Question Five**

Sam and Pat were having a disagreement about solving Sodoku. Sam suggested implementing an *alldifferent*
constraint. Pat suggested implementing lots of pairwise constraints. We managed to get an implementation
of Sam’s suggestion in *sodoku.py* in the code distribution.

(a) Which would you expect to work better? Explain why. (Please try this before implementing it.)
   Explain why someone may think the other method would be better.

(b) Implement Pat’s suggestion. Which one is more efficient in terms of runtime? You can use either arc
   consistency with domain splitting or stochastic local search. (Give the evidence for your claim.)

(c) Given the evidence, explain why the method that is more efficient is more efficient. Is there a general
   principle you could suggest about which one is better?
   [Note: a good way to learn something is to make a prediction before making observations. There are
   no marks for getting the correct prediction in part (a) — don’t try to change your prediction based on the
   evidence (do that in part (c)).]

**Question Six**

The Python implementation of domain splitting with arc consistency in the provided code starts each arc
consistency with all of the arcs in *to do*; (see *make_arc_consistent* called in *Search_with_AC_from_CSP*).
The CSP applet does something smarter in only including the arcs that could have been make inconsistent
by the assignment. Implement arc consistency with domain splitting that only includes the arcs in *to do*
that could have changed. Compare it to the original using a sensible measure of search costs.

**Question Seven**

For each question, specify how long you spend on it, and what you learned. How was the work in the team
allocated? Was the question reasonable? (This questions is worth marks, so please do it!)