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. 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 AlSpace 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.

Solution   Python code that implements the coding questions can be found at: http://artint.info/code/python/aifca_322_as2_sol.zip

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 a 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 AlSpace 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).
Solution  We have four vars $HC, BH, GD, RA$. The domains are the locations.

It is possible to have the domains being all of the locations, and have domain constraints, but it is probably simpler to have it domain consistent initially. Thus the domains of $HC$ and $BH$ is \{02, 11, 20, 21, 22\}. The domain of $RA$ is \{02, 11, 22\}. The domain of $GD$ is \{01, 02, 10, 11, 20, 21, 22\}.

There are pairwise constraints saying each variable is different. For the other constraints see the AIspace representation at:

http://www.cs.ubc.ca/~poole/cs322/2015/as2/322as2q1.xml

load this link into the AIspace CSP applet.

There is no solution. There is no place to put RA so that it is close to the lake and has two locations that are not adjacent to the cemetery; every location close to the lake has at most one adjacent location that is not close to the cemetery.

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, $H \neq D, D > G, D \neq C, E \neq C, E < D - 1, E \neq H - 2, G \neq F, H \neq F, C \neq D, D \neq F, |E - F|$ is odd.

An AIspace representation for this graph is at:

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 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

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.)

Solution  There are two solutions and 1525 failing branches.


To find a smaller tree, a good heuristic is to always choose the most constrained variable. This is the variable that is in the most constraints with other variables. Another good heuristic is to prefer the one that is in the most constraints with previous variables in the ordering, and to break ties by choosing the variable that is in the most constraints.

You get a bonus mark for having a non-uniform tree. That is, where which variable chosen depends on the context. You could possibly do better than the 74 failing branches. (Just as Figure 4.1 in the textbook is smaller than any static 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).

Solution Parts (a)-(c) can be done with the applet. You were to write it out just to make sure you understood what was going on. Note that the “Domain-splitting history” at the bottom of the applet is a representation of the tree of splits.

For part (d) you should compare them on the number of consistency checks (or another fair measure). The correct answer depends on the ordering chosen.

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 AI space 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.
Solution

(a) This is what the show trace in the applet gives you. Just make sure you understand it.

(b) You need to refer to the runtime distribution to get full marks. (i) works better than (ii) for the first 60 steps, but can only solve about 70% of the runs. (ii) can solve all of the runs. (Set “terminate after” to be about 2000 to see this). (iii) can solve all of the runs, but is dominated by (ii).

(c) For each of these, choosing a random value works much worse than choosing the best value.

(d) In terms of steps, choosing the best node and value, with a random restart after about 6 steps works really well. Choosing the best node-value pair works well (which for some reason doesn’t seem get stuck in a local minima in this example). These may not so good in term of time though. In terms of run time choosing the best variable say 60% of the time and a random “red” node 40% of the time, with the best value works well. Perhaps you did better with some other settings. We just wanted you to play.

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)).]

Solution I had thought that Sam’s suggestion may be better, because it has many fewer constraints, and each is more powerful. But it is much worse, mainly because it requires finding a solution for all of the other variables in the constraint whenever one value is considered.

Here is a procedural specification of Pat’s idea:

```python
def pairwise_ne_constraints(vars):
    """make each pair of variables in vars into a not-equal (ne) constraint. returns the list of constraints ""
    return [Constraint([vars[i],vars[j]],ne) for i in range(len(vars)) for j in range(i)]

def sodoku_CSP(eg):
    """returns a CSP given the partially filled Sodoku table in eg.""
```
eg[y][x] gives the value in column y, row x, or 0 if it is unknown

```python
cons = []
for i in range(9):
    cons += pairwise_ne_constraints([(i, j) for j in range(9)]) # column constraint
    cons += pairwise_ne_constraints([(j, i) for j in range(9)]) # row constraint
for xo in range(3): # x-offset
    for yo in range(3): # y-offset
        cons += pairwise_ne_constraints(
            [(3*xo+i, 3*yo+j) for x in range(3) for y in range(3)])
return CSP(
    {(x, y): (digits if eg[y][x]==0 else [eg[y][x]])
     for y in range(9) for x in range(9)},
    cons)
```

Or for those who prefer a more declarative representation (but this doesn’t take into account the symmetry of inequality for the blocks):

```python
def sodoku_CSP_alt(eg):
    """returns a CSP given the partially filled Sodoku table in eg.
    eg[y][x] gives the value in column y, row x, or 0 if it is unknown.

    This implementation is for those who prefer a more declarative representation
    """
    return CSP(
        {(x, y): (digits if eg[y][x]==0 else [eg[y][x]])
         for y in range(9) for x in range(9)},
        [Constraint([(i,j),(i,k)],ne) for i in range(9)
         for j in range(9) for k in range(j)] + # column constraint
        [Constraint([(j,i),(k,i)],ne) for i in range(9)
         for j in range(9) for k in range(j)] + # row constraint
        [Constraint([(3*xo+i,3*yo+j),(3*xo+k,3*yo+l)],ne)
         for xo in range(3) for yo in range(3)
         for i in range(3) for j in range(3)
         for k in range(3) for l in range(3) if (i,j)!=(k,l)])
```

This is so much faster even thought there are around 1000 constraints rather than the 27 constraints. Each check is very efficient. See sodoku_alt.py in the

\[ http://artint.info/code/python/aifca_322_as2_sol.zip \]

**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.
**Solution**  Python code that implements the solutions can be found at:

http://artint.info/code/python/aifca_322_as2_sol.zip

The important part to look for is the call to `make_arc_consistent` can now optionally take in a `to_do` set.

**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!)

**Solution**  It should not have taken more than a few hours. Most of this should have been in understanding the material, not in doing busy work. I hope it was reasonable, and you learned something. Questions 5 and 6 were designed to be more challenging, but we expected these to be done by teams.