Assignment Four: CSPs

Solution

Question One

(a) This can be done with the applet or the Python. You were to write it out just to make sure you understood what was going on.

(b) The arc consistent domains are

```
'A': {2, 3, 4},
'B': {1, 2, 3, 4},
'C': {1, 2, 3, 4},
'D': {3, 4},
'E': {1, 2},
'F': {1, 2, 3, 4},
'G': {1, 2, 3},
'H': {2, 3, 4}
```

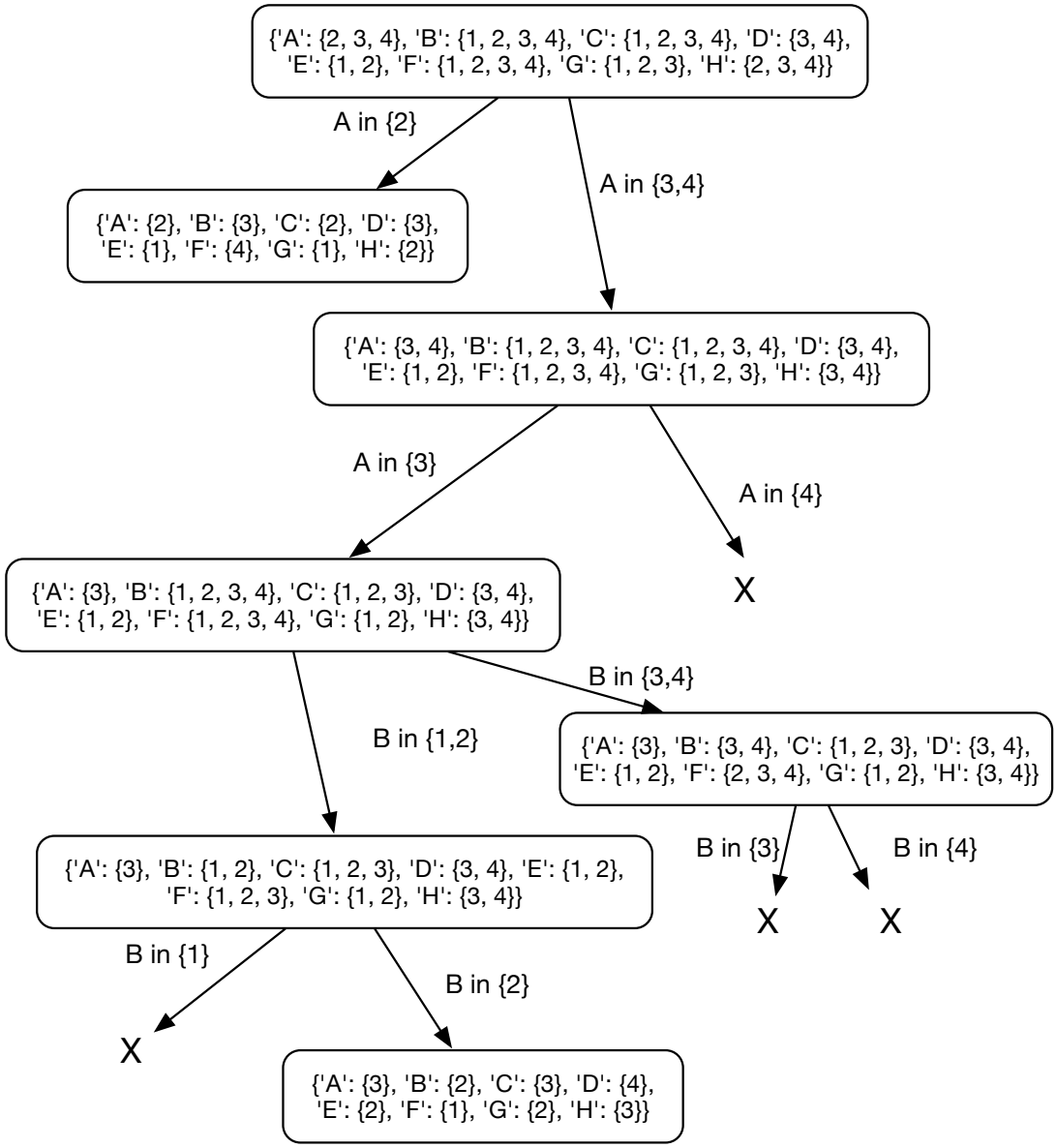
Note that there is a unique correct answer.

(c) Here is one domain splitting history. This is not particularly good ordering (this is the one given by the AIPython code).

This can be extracted from puzzling out AIPython when you load as3csp.py and do:

```
>>> from cspConsistency import Search_with_AC_from_CSP
>>> from searchGeneric import Searcher
>>> Search_with_AC_from_CSP.max_display_level = 3
>>> Searcher.max_display_level = 2
>>> schr = Searcher(Search_with_AC_from_CSP(as2csp))
>>> print(schr.search())
```

The same information can be obtained from the CSP applet. (But the applet lets you play with more interesting variable orderings).



(d) This tree is obviously smaller than the trees we got in assignment 3, but that isn't a fair comparison as the CSP does much more work to expand a node. One fair comparison may be the number of consistency tests. This is easy to get in the CSP applet; just count how many clicks you needed to find all solutions. The AIPython code processes 335 arcs (you may have to add a counter to the code to get this), using this poor ordering, which is much worse than the ordering for the search trees (counting the number of evaluations of constraints).

You could have done what you wanted to get marks here as long as you tried to get a fair comparison.

Note that arc consistency does not save a lot for this example. Sometimes the improvement is dramatic; but sometimes brute-force search is not so bad (with a good variable ordering).

Question Two

(a) This is what the show trace in the applet gives you. Just make sure you understand it. Or from using a more detailed trace in AIPython.

Here is one fro AIPython where I just included the variable that changes. Note that it chooses another value at random.

A	B	C	D	E	F	G	H	# Conflicts
3	2	1	2	3	1	2	4	5
					4			5
							2	7
							3	3
			1					4
			4					3
			2					3
			3					3
			4					3
					3			3
				4				2

Note this this eventually found a solution after 17 steps.

(b) You need to refer to the runtime distribution to get full marks. (i) works better on average than (ii) for the first 60 steps, but can only solve about 40% of the runs. (ii) can solve all of the runs. (Set "terminate after" to be about 2000 or maybe a bit more to see this). (iii) can solve all of the runs, but is dominated by (ii), where *dominated* means that for every number of steps, it can solve a larger proportion of the problems in that many (or fewer) steps.

(c) In terms of steps, choosing the best node and value, with a random restart after about 6 steps works quite well. These may not so good in term of time though.

(d) Choosing the best variable say 70% of the time and a random "red" node 30% of the time, with the best value works well. Perhaps you did better with some other settings. We just wanted you to play.

Question Three

It should not have taken more than a few hours. Most of this should have been in understanding the material, and playing, not in doing busy work. It should make more sense now. I hope it was reasonable, and you learned something.