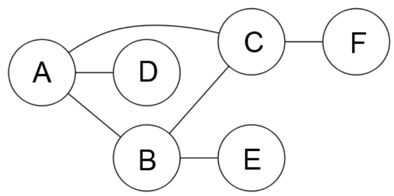
CS188: Exam Practice Session 2 Solutions

Q1. CSPs

(a) The graph below is a constraint graph for a CSP that has only binary constraints. Initially, no variables have been assigned.

For each of the following scenarios, mark all variables for which the specified filtering might result in their domain being changed.



(i) A value is assigned to A. Which domains might be changed as a result of running forward checking for A?

○ A ● B ● C ● D ○ E ○ F

Forward checking for A only considers arcs where A is the head. This includes $B \to A$, $C \to A$, $D \to A$. Enforcing these arcs can change the domains of the tails.

(ii) A value is assigned to A, and then forward checking is run for A. Then a value is assigned to B. Which domains might be changed as a result of running forward checking for B?

○ A ○ B ● C ○ D ● E ○ F

Similar to the previous part, forward checking for B enforces the arcs $A \to B$, $C \to B$, and $E \to B$. However, because A has been assigned, and a value is assigned to B, which is consistent with A or else no value would have been assigned, the domain of A will not change.

(iii) A value is assigned to A. Which domains might be changed as a result of enforcing arc consistency after this assignment to A?

○ A ● B ● C ● D ● E ● F

Enforcing arc consistency can affect any unassigned variable in the graph that has a path to the assigned variable. This is because a change to the domain of X results in enforcing all arcs where X is the head, so changes propagate through the graph. Note that the only time in which the domain for A changes is if any domain becomes empty, in which case the arc consistency algorithm usually returns immediately and backtracking is required, so it does not really make sense to consider new domains in this case.

(iv) A value is assigned to A, and then arc consistency is enforced. Then a value is assigned to B. Which domains might be changed as a result of enforcing arc consistency after the assignment to B?

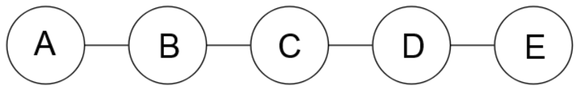
○ A ○ B ● C ○ D ● E ● F

After assigning a value to A, and enforcing arc consistency, future assignments and enforcing arc consistency will not result in a change to As domain. This means that Ds domain won't change because the only arc that might cause a change, $D \to A$ will never be enforced.

(b) You decide to try a new approach to using arc consistency in which you initially enforce arc consistency, and then enforce arc consistency every time you have assigned an even number of variables.

You have to backtrack if, after a value has been assigned to a variable, X, the recursion returns at X without a solution. Concretely, this means that for a single variable with d values remaining, it is possible to backtrack up to d times. For each of the following constraint graphs, if each variable has a domain of size d, how many times would you have to backtrack in the worst case for each of the specified orderings?

(i)



A-B-C-D-E: 0

A-E-B-D-C: $2(d-1)$

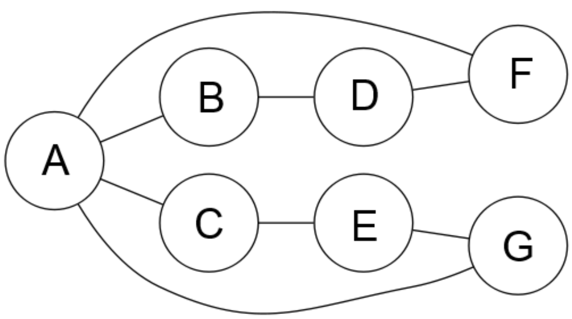
C-B-D-E-A: 0

If no solution containing the current assignment exists on a tree structured CSP, then enforcing arc consistency will always result in an empty domain. This means that running arc consistency on a tree structured CSP will immediately tell you whether or not the current assignment is part of a valid solution, so you can immediately start backtracking without further assignments.

$A-B-C-D-E$ and $C-B-D-E-A$ are both linear orderings of the variables in the tree, which is essentially the same as running the two pass algorithm, which will solve a tree structured CSP with no backtracking.

$A-E-B-D-C$ is not a linear ordering, so while the odd assignments are guaranteed to be part of a valid solution, the even assignments are not (because arc consistency was not enforced after assigning the odd variables). This means that you may have to backtrack on every even assignment, specifically E and D. Note that because you know whether or not the assignment to E is valid immediately after assigning it, the backtracking behavior is not nested (meaning you backtrack on E up to $d-1$ times without assigning further variables). The same is true for D, so the result is backtracking $2(d-1)$ times.

(ii)



A-B-C-D-E-F-G: d^2-1

F-D-B-A-C-G-E: $d^4-1+d-1$

C-A-F-E-B-G-D: d^2-1

$A-B-C-D-E-F-G$: The initial assignment of A, B might require backtracking on both variables, because there is no guarantee that the initial assignment to A is a valid solution. Because A is a cutset for this graph, the resulting graph consists of two trees, so enforcing arc consistency immediately returns whether the assignments to A and B are part of a solution, and you can begin backtracking without further assignments.

$F-D-B-A-C-G-E$: Until A is assigned, there is no guarantee that any of the previous values assigned are part of a valid solution. This means that you may be required to backtrack on all of them, resulting in d^4-1 times. Furthermore, the remaining tree is not assigned in a linear order, so further backtracking may be required on G (similar to the second ordering above) resulting in a total of $(d^4-1)+(d-1)$.

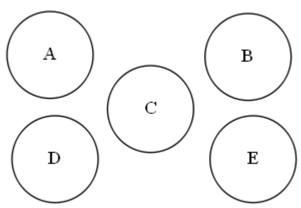
$C-A-F-E-B-G-D$: This ordering is similar to the first one. except that the resulting trees are not being assigned in linear order. However, because each tree only has a single value assigned in between each run of arc consistency, no backtracking will be required (you can think of each variable as being the root of the tree, and the assignment creating a new tree or two where arc consistency has been enforced), resulting in a total of d^2-1 times.

(c) Consider a modified CSP in which we wish to find *every possible satisfying assignment*, rather than just one such assignment as in normal CSPs. In order to solve this new problem, consider a new algorithm which is the same as the normal backtracking search algorithm, except that when it sees a solution, instead of returning it, the solution gets added to a list, and the algorithm backtracks. Once there are no variables remaining to backtrack on, the algorithm returns the list of solutions it has found.

For each graph below, select whether or not using the MRV and/or LCV heuristics could affect the number of nodes expanded in the search tree in this new situation.

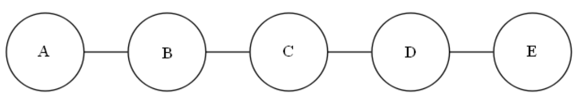
The remaining parts all have a similar reasoning. Since every value has to be checked regardless of the outcome of previous assignments, the order in which the values are checked does not matter, so LCV has no effect.
In the general case, in which there are constraints between variables, the size of each domain can vary based on the order in which variables are assigned, so MRV can still have an effect on the number of nodes expanded for the new "find all solutions" task.
The one time that MRV is guaranteed to not have any effect is when the constraint graph is completely disconnected, as is the case for part i. In this case, the domains of each variable do not depend on any other variable's assignment. Thus, the ordering of variables does not matter, and MRV cannot have any effect on the number of nodes expanded.

(i)



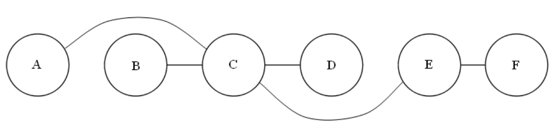
● Neither MRV nor LCV can have an effect.
○ Only MRV can have an effect.
○ Only LCV can have an effect .
○ Both MRV and LCV can have an effect.

(ii)



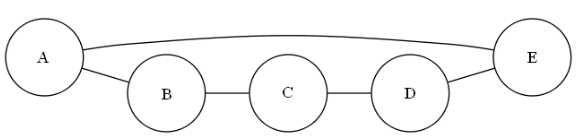
○ Neither MRV nor LCV can have an effect.
● Only MRV can have an effect.
○ Only LCV can have an effect .
○ Both MRV and LCV can have an effect.

(iii)

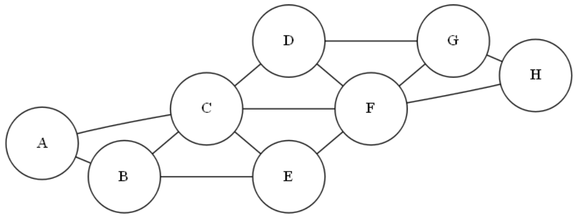


○ Neither MRV nor LCV can have an effect.
● Only MRV can have an effect.
○ Only LCV can have an effect .
○ Both MRV and LCV can have an effect.

(iv)



○ Neither MRV nor LCV can have an effect.
● Only MRV can have an effect.
○ Only LCV can have an effect .
○ Both MRV and LCV can have an effect.

(v) 

- ◯ Neither MRV nor LCV can have an effect.
- ● Only MRV can have an effect.
- ◯ Only LCV can have an effect .
- ◯ Both MRV and LCV can have an effect.