CS 188
Spring 2019

Section Handout 6 Solutions

Structure

- **Tree-structured CSP**: a CSP that has no loops in its constraint graph. In other words, there is only one path between each pair of nodes. Whereas for a general CSP the runtime is $O(d^N)$, for a tree-structured CSP we can reduce the runtime to $O(nd^2)$, linear in the number of variables. The tree-structured CSP algorithm operates as follows:

  1. Pick an arbitrary node in the constraint graph for the CSP to serve as the root of the tree.
  2. Convert all undirected edges in the tree to directed edges that point *away* from the root. Then **linearize** (or **topologically sort**) the resulting directed acyclic graph.
  3. Perform a **backwards pass** of arc consistency. Iterating from $i = n$ down to $i = 2$, enforce arc consistency for all arcs $Parent(X_i) \longrightarrow X_i$. This means that for each element in the child's domain, prune values in the parent's domain with are inconsistent with that element. By the time the backwards pass is finished, the domains of all the parents are consistent with those of their children.
  4. Perform a **forward assignment**. Starting from X_1 and going to X_n, assign each X_i a value consistent with that of its parent. Because we've enforced arc consistency on all of these arcs, no matter what value we select for any node, we know that its children will each all have at least one consistent value.

- **Cutset Conditioning**: Cutset conditioning involves first finding the smallest subset of variables in a constraint graph such that their removal results in a tree (such a subset is known as a **cutset** for the graph). For all possible assignments for the variables in the cutset, run the tree-structured CSP algorithm in the remaining constraint graph. Since removal of the cutset leaves us with a tree-structured CSP with $(n - c)$ variables, we know this can be solved (or determined that no solution exists) in $O((n - c)d^2)$. Hence, the runtime of cutset conditioning on a general CSP is $O(d^c(n - c)d^2)$, very good for small c.

Local Search

Local search works by iterative improvement - start with some random assignment to values then iteratively select a random conflicted variable and reassign its value to the one that violates the fewest constraints until no more constraint violations exist (a policy known as the **min-conflicts heuristic**).

Propositional Logic

A **knowledge base** is a set of **sentences** which are statements that represent some assertion about the world. A **model** is a possible world that fixes the truth or falsehood of every sentence. If a sentence α is true in model m, we say that m **satisfies** α or m **is a model of** α.

In propositional logic, the **syntax** of propositional logic defines the allowable sentences. A **propositional symbol** stands for a proposition that can be true or false. **complex sentences** are constructed from simpler sentences, using parentheses and **logical connectives.** There are five common connectives: $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\Rightarrow$ (implies), $\Leftrightarrow$ (if and only if).

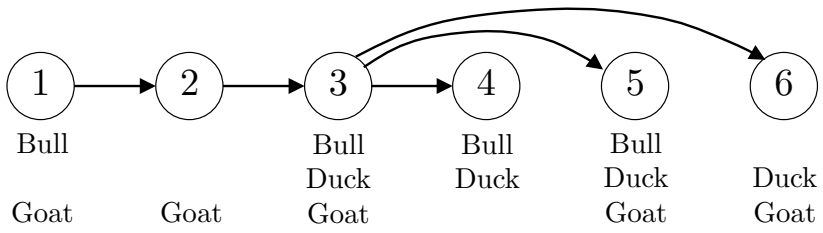
The truth table for $A \Rightarrow B$ is:

A	B	$A \Rightarrow B$
T	T	T
T	F	F
F	T	T
F	F	T

Q1. Farmland CSP

The animals in Farmland aren't getting along and the farmers have to assign them to different pens. To avoid fighting, animals of the same type cannot be in connected pens. Fortunately, the Farmland pens are connected in a tree structure.

(a) Consider the following constraint diagram that shows six pens with lines indicating connected pens. The remaining domains for each pen are listed below each node.



After assigning a bull to pen 5, enforce arc consistency on this CSP considering only the *directed* arcs shown in the figure. What are the remaining values for each pen?

Pen	Values
1	Bull
2	Goat
3	Duck, Goat
4	Bull, Duck
5	Bull
6	Duck, Goat

(b) What is the computational complexity of solving general tree structured CSPs with n nodes and d values in the domain? Give an answer of the form $O(\cdot)$.

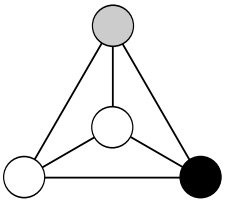
$O(nd^2)$

(c) True/False: If root to leaf arcs are consistent on a general tree structured CSP, assigning values to nodes from root to leaves will not back-track if a solution exists.

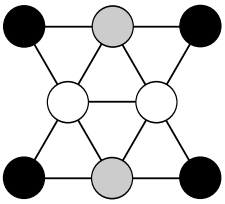
True. Because the arcs are consistent, there is a valid value not matter which parent value was assigned.

Q2. Local Search

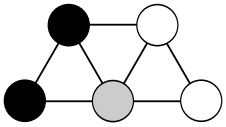
In this question we are considering CSPs for map coloring. Each region on the map is a variable, and their values are chosen from {black, gray, white}. Adjacent regions cannot have the same color. The figures below show the constraint graphs for three CSPs and an assignment for each one. None of the assignments are solutions as each has a pair of adjacent variables that are white. For both parts of this question, let the score of an assignment be the number of satisfied constraints (so a higher score is better).



(1)



(2)



(3)

(a) Consider applying Local Search starting from each of the assignments in the figure above. For each successor function, indicate whether each configuration is a local optimum and whether it is a global optimum (note that the CSPs may not have satisfying assignments).

Successor Function	CSP	Local optimum?		Global Optimum?	
Change a single variable	(1)	**Yes**	No	**Yes**	No
	(2)	**Yes**	No	Yes	**No**
	(3)	Yes	**No**	Yes	**No**
Change two or three variables at a time	(1)	**Yes**	No	**Yes**	No
	(2)	Yes	**No**	Yes	**No**
	(3)	Yes	**No**	Yes	**No**

In the first row ("Change a single variable"), (1) is a local optimum as well as a global optimum because no configuration can be strictly better. This is because the center node has three neighbors, making it impossible violate less than one edge constraint. (2) is a local optimum because changing the assignment of one node will not reduce the number of constraints violated, but it is not a global optimum because it is possible to find a set of assignments that violate zero constraints. (3) is not a local optimum because changing the assignment of the bottom right node to black will reduce the number of violated constraints by 1 (the constraint between the two currently white nodes).

In the second row ("Change two or three variables at a time"), (1) is still a local and global optimum because no configuration can be strictly better. (2) is not a local optimum because it is possible to assign the top right node to white, the middle right node to black, and the bottom right node to white and violate no constraints which is strictly better, which also makes the current configuration of (2) not a global optimum. (3) is a not a local optimum because it is possible to change the top-left node to white and the bottom-right node to black and violate strictly less constraints.

Q3. Propositional Logic

(a) Pacman has lost the meanings for the symbols in his knowledge base! Luckily he still has the list of sentences in the KB and the English description he used to create his KB.

For each English sentence on the left, there is a corresponding logical sentence in the knowledge base on the right (**not necessarily the one across from it**). Your task is to recover this matching. Once you have, please fill in the blanks with the English sentence that matches each symbol.

English	**Knowledge Base**
There is a ghost at (0, 1).	$(C \vee B) \wedge (\neg C \vee \neg B)$
If Pacman is at (0, 1) and there is a ghost at (0, 1), then Pacman is not alive.	$C \wedge \neg D$
Pacman is at (0, 0) and there is no ghost at (0, 1).	$\neg A \vee \neg(B \wedge D)$
Pacman is at (0, 0) or (0, 1), but not both.	D

$A =$ Pacman is alive

$B =$ Pacman is at (0, 1)

$C =$ Pacman is at (0, 0)

$D =$ There is a ghost at (0, 1)

(b) Consider a generic propositional model with 4 symbols: A, B, C, D. For the following sentences, mark how many models containing these 4 symbols will satisfy it.

(i) $\alpha_1 = A$: 8

There are 16 total models; A is true in half of them.

(ii) $\alpha_2 = (C \wedge D) \vee (A \wedge B)$: 7

There are 4 possible worlds where $C \wedge D$ is true, 4 where $A \wedge B$ is true, and one where they are both true, so the total is 7.

(iii) $\alpha_3 = (A \vee B) \Rightarrow C$: 10

There are 4 possible worlds where $A \vee B$ is false so the implication is satisfied for all values of C. In the remaining 12 possible worlds, C is true is 6 of them. So there are 10 possible worlds

(c) The DPLL satisfiability algorithm is a backtracking search algorithm with 3 improvements: PURE-SYMBOLS, UNIT-CLAUSES, and EARLY TERMINATION. In this question, we'll ask you to relate these improvements (if possible) to more general CSP techniques used with backtracking search.

(i) The PURE-SYMBOL technique finds a propositional symbol that only occurs with the same "sign" throughout the expression and assigns it the corresponding value. Such symbols are called pure. In the following CNF expression, which symbols are pure and what value does this process assign to those symbols?

$$(C \vee D) \wedge (C \vee \neg A) \wedge (\neg D \vee A) \wedge (\neg B \vee A)$$

pure symbols(s): B, C

value(s) assigned: B = False, C = True

(ii) Which of the following CSP techniques is equivalent to the PURE-SYMBOLS technique when applied to SAT for CNF sentences:

- ○ MINIMUM-REMAINING-VALUES
- ○ FORWARD-CHECKING
- ● LEAST-CONSTRAINING-VALUE
- ○ BACKTRACKING
- ○ No equivalent CSP technique

Assigning a value to a pure literal can only cause clauses to become true, so it will impose 0 additional constraints on the remaining literals. Notice that a pure symbol (except in a unit clause) can still take on either value, so this is not an MRV choice.

(iii) The UNIT-CLAUSE technique finds all clauses that contain a single literal and assigns values to those literals. Which of the following CSP techniques is equivalent to the UNIT-CLAUSE technique when applied to SAT for CNF sentences:

- ● MINIMUM-REMAINING-VALUES
- ○ FORWARD-CHECKING
- ○ LEAST-CONSTRAINING-VALUE
- ○ BACKTRACKING
- ○ No equivalent CSP technique

A variable in a unit clause can only be a assigned to a single value in a satisfying assignment, otherwise that clause would be false. Thus, there is only 1 remaining value for such literals.

(iv) DPLL performs early termination in two steps: SUCCESS-DETECTION and FAILURE-DETECTION. First, SUCCESS-DETECTION checks to see if *all* clauses evaluate to true. If so, the sentence is satisfiable and any unassigned propositional symbols can be assigned arbitrarily. Next, FAILURE-DETECTION checks if *any* clause evaluates to false. In this case the sentence is unsatisfiable and this branch of the search can be pruned. How does this strategy relate to the early termination strategy used by the general BACKTRACKING algorithm?

- ○ BACKTRACKING does both SUCCESS-DETECTION and FAILURE-DETECTION
- ○ BACKTRACKING does only SUCCESS-DETECTION
- ● BACKTRACKING does only FAILURE-DETECTION
- ○ BACKTRACKING does neither

In the backtracking CSP algorithm from the book, early termination is triggered whenever the domain for a variable is empty. This is exactly the case when a clause has been reduced to false, so FAILURE-DETECTION is an application of this technique to SAT. Early detection of success is possible in SAT

because if *any* literal in a clause is true, the clause is satisfied, and no further assignments to the other variables can unsatisfy it. This is *not* true in general CSPs: a partial assignment may not violate any constraints, but it is never (in general) certain to become a solution because further assignments may violate some constraints. Hence early success detection is not possible in general CSPs.