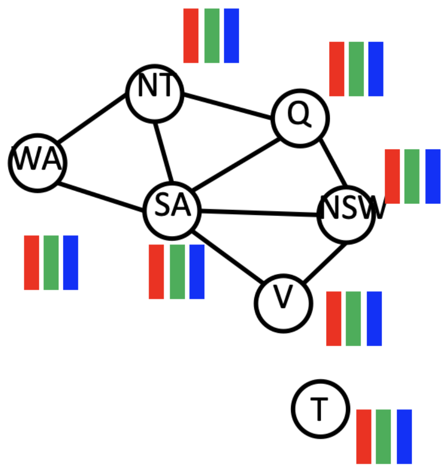
CAS CS 440
Lec 10

CSP II

1. Review
    a. A CSP or "Constrained Satisfaction Problem":
        i. Representing a problem as constraints and variables (generalizing)
        ii. Items/entities that can take on values
        iii. Items/entities have constraints that relate them to each other
    b. Variables $X = \{X_1, X_2, \ldots, X_n\}$
        i. Each variable $X_i$ has its own domain $Di = \{v_1, v_2, \ldots, v_k\}$
            1. Possible values that can be assigned to
        ii. Each variable must be assigned a value
    c. Constraints $C = \{C_1, C_2, \ldots, C_m\}$
        i. Each constraint is Boolean: relates variables to each other
        ii. In map coloring:
            1. Adjacent variables must have different colors
                a. $C_j \leftarrow X_l \mathrel{!=} X_p$
    d. All constraints are unary/binary constraints (unary constraint has self-edges)
        i. Any (n > 2)-ary constraint can be reduced to a bunch of binary constraints
            1. Trick: invent new variables!
            2. Reduce constraints using new variables as "intermediary steps"
    e. Turn CSP into a graph
        i. Vertex = variable
        ii. Edge = constraint
    f. How do we know if a CSP is solvable?
        i. We don't beforehand!
        ii. Can check though (in polynomial time)
    g. Vocab terms needed:
        i. A vertex is node consistent iff no value in it's domain breaks a unary constraint (domain is good)
        ii. A vertex is arc consistent iff no value in it's domain breaks a binary constraint
            1. Arc consistent with respect to another vertex!
    h. Is NT arc consistent with respect to Q? ✓
        i. Consider NT=R $\exists y \in D_Q \;\; s.t. \;\; NT \neq Q?$ ✓
            1. If I assign value of Red to NT, does it work with Q/will that break constraint with Q? (yes)

ii. Consider NT=G $\exists y \in D_Q \ \ s.t. \ \ NT \neq Q?$ ✓

iii. Consider NT=B $\exists y \in D_Q \ \ s.t. \ \ NT \neq Q?$ ✓

i. 


j. How do we check?
   i. AC-3 Algorithm
   ii. Preprocessing step!

k. Bonus:
   i. $$\left(\forall\ j \ \ \left|D_j\right| = 1\right) \rightarrow solution$$

l. **function** AC-3( $csp$) **returns** false if an inconsistency is found and true otherwise
   **inputs**: $csp$, a binary CSP with components $(X, D, C)$
   **local variables**: $queue$, a queue of arcs, initially all the arcs in $csp$

   Doesn't need to be a queue
   An unordered collection

   **while** $queue$ is not empty **do**
     $(X_i, X_j) \leftarrow$ REMOVE-FIRST($queue$)
     **if** REVISE($csp$, $X_i$, $X_j$) **then**
       **if** size of $D_i = 0$ **then return** $false$
       **for each** $X_k$ **in** $X_i$.NEIGHBORS - $\{X_j\}$ **do**
         add $(X_k, X_i)$ to $queue$
   **return** $true$

   Make $X_i$ arc consistent wrsp to $X_j$

   If we change $D_i$
   - Need to recheck arc consist. with all other neighbors
   - Another neighbor $X_k$ may have relied on a value we removed to be arc consist. with $X_i$!

   **function** REVISE( $csp$, $X_i$, $X_j$) **returns** true iff we revise the domain of $X_i$
     $revised \leftarrow false$
     **for each** $x$ **in** $D_i$ **do**
       **if** no value $y$ in $D_j$ allows $(x,y)$ to satisfy the constraint between $X_i$ and $X_j$ **then**
         delete $x$ from $D_i$
         $revised \leftarrow true$
     **return** $revised$

   Pseudocode:
   - Remove any values from $D_i$ that make it not arc consist. w/ $X_j$

   i. Grab an edge first and get the two nodes that have that edge
   ii. REVISE returns true when one or more domains are removed
   iii. Since I lost domain, I have to check whether the node that connected with me is arc consistent again (recheck and make all neighbors arc consistent for me)

2. Today: Searching for Solution
    a. Remember:
        i. We can tree search this
        ii. Need to make the tree small enough to be worth it
        iii. $O\left(|C|\left(\max_{1\le i\le n}|D_i|\right)^3\right)$
        iv. Run AC-3 as a preprocessing step!
            1. If AC-3 returns false, not solvable so don't even try to solve!
            2. If AC-3 returns true:
                a. Might be lucky and AC-3 has solved it for you (O(n) time to check)
                b. Domains of Variables produced by AC-3 are minimal
                    i. Every vertex is node + arc consistent
                    ii. No unnecessary searching
    b. How to actually solve given minimal CSP?
3. DFS Tree Search
    a. Like we've talked about before:
        i. Fix order of variables (prune the tree)
        ii. Expand the tree using DFS
            1. State = partial assignment (assign some partial variables)
            2. Action = assigning (Var = value) to the partial assignment
            3. Stop expanding a branch when a constraint is violated
            4. Leaf node in tree = legal (does not break constraints) & complete (went to all nodes) assignment
4. DFS Tree Search & Backtrack Algorithm
    a.

```
                    DFS-Search interface function
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return BACKTRACK({ }, csp)
                    DFS helper function (to do the actual DFS)
function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment    Base case: legal & compete assignment → solution!
    var ← SELECT-UNASSIGNED-VARIABLE(csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do    Actions! Expand the tree
        if value is consistent with assignment then
            add {var = value} to assignment                assignment is a reference! Not a deepcopy!
            inferences ← INFERENCE(csp, var, value)        TODO: talk about this
            if inferences ≠ failure then
                add inferences to assignment
                result ← BACKTRACK(assignment, csp)         Recursive DFS call
                if result ≠ failure then
                    return result
        remove {var = value} and inferences from assignment    assignment is a reference! Not a deepcopy!
    return failure
```

    b. How to configure Backtrack?
        i. Select-Unassigned-variable
        ii. Order-Domain-Values
        iii. Inference
5. CSP Heuristics
    a. Heuristics in past problems:
        i. Domain-specific knowledge
        ii. Task Engineering
    b. Heuristics in CSPs:
        i. More abstract
        ii. Apply to all CSPs
    c. Variable ordering
        i. SELECT-UNASSIGNED-VARIABLE
        ii. Goal: Prune the tree
            1. Rather than rely on a fixed ordering of variables
            2. Pick new variable based on what other's have already been chosen!
            3. Pick variable with smallest domain remaining!
                a. Minimum Remaining Values (MRV) heuristic
                b. Fail-first heuristic
            4. Degree heuristic:
                a. Pick variable involved in the most constraints!
            5. Can combine multiple heuristics!
                a. Use MRV and settle ties with degree heuristic!
    d. Value ordering
        i. ORDER-DOMAIN-VALUES
            1. Choose value that is the most "flexible"
            2. Least Constraining Value (LCV) heuristic:
                a. Prefers domain values that affect neighbor domains the least
                b. Fail-last heuristic
6. 