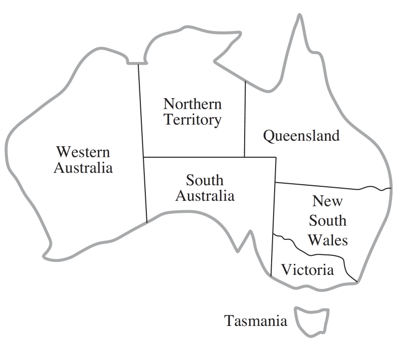
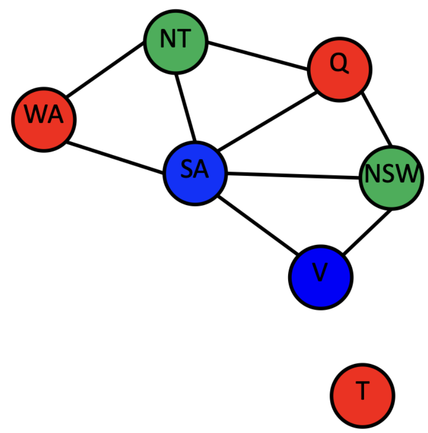
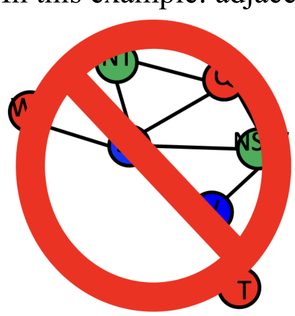
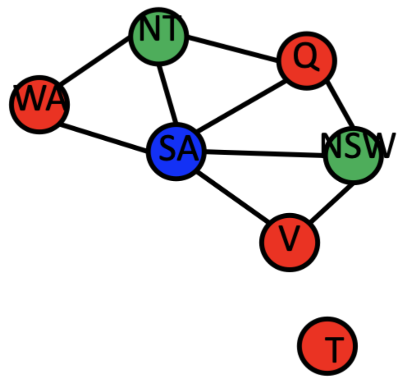
CAS CS 440
Lec 9

CSP I

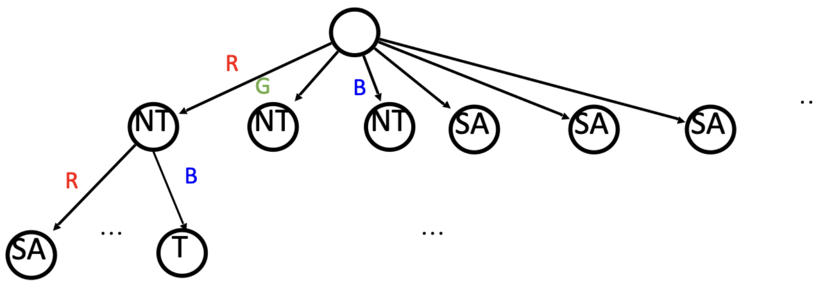
1. Tree Search
    a. Expand tree to a new level
        i. Consider new moves to apply!
        ii. Leaf nodes in tree should be terminal states
            1. Problem when leaf nodes are nonterminals! (pretend our utility values are heuristic values)
    b. Can apply this to single player games
2. Example
    a. Want to assign a color (RGB) to each region of Australia
        i. Model each region as a graph
        ii. Add edge connecting adjacent regions
        iii. 

→ 

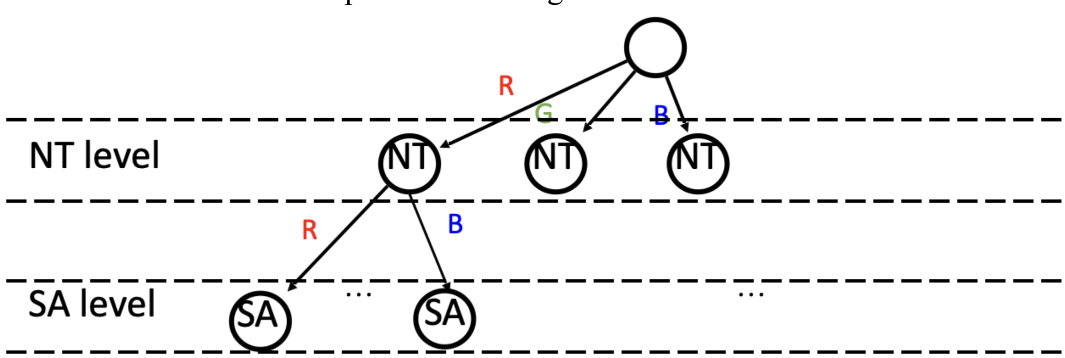
3. Trouble with Tree Search
    a. What if we have constraints?
        i. In this example: adjacent regions can't have the same color
        ii. 

iii. 

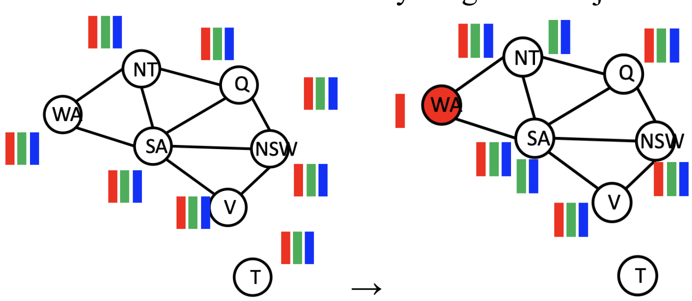
b. Tree Search crazy inefficient
    i. Will find correct answer (if one exists)
    ii. Tree will consider all possible orderings of vertices!

c. 

d. Tree is massive
e. Ordering of Vertices doesn't need to be permuted!
    i. Wasteful!
    ii. Solution doesn't depend on ordering of vertices!

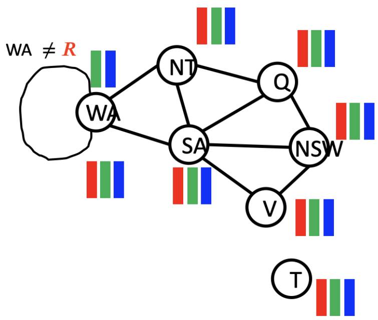
f. 

4. CSP
    a. A CSP or "Constrained Satisfaction Problem" meets this template
    b. Variables X = $\{X_1, X_2, \ldots, X_n\}$
        i. Each variable X has its own domain
            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. An assignment :
        i. Set of variables with their assignments
        ii. A partial assignment = not all variables have an assignment
        iii. A complete assignment = all variables have an assignment
        iv. A legal assignment = assignment satisfies contraints
    e. Search for a complete & legal assignment:
        i. Pick ordering of variables (reduces tree size)
        ii. Dfs the tree!
    f. It is possible to not be able to find a complete & legal assignment!
5. Tree Pruning
    a. Typically model CSPs as a constraint graph
        i. Every n-ary (n > 2) constraint can be converted to a bunch of binary constraints
        ii. Each variable becomes a vertex
        iii. (unary/binary) constraints becomes edges
    b. Prune the tree?
        i. Tree is still massive
        ii. Once we make partial assignment {WA = red}:
            1. Can we infer anything about adjacent vertices?
    c. 

6. Node & Arc Consistency
    a. Goal: prune domain D_i for variable X_i
        i. Pruning domain = pruning tree!
    b. How?
        i. Lets say variable X_j has some unary constraints:
            1. Reduce domain to all values that satisfy this constraint
            2. Node Consistency (1-consistency)
        ii. Let's say X_i, X_j participate in some binary constraint c
            1. When we have an assignment $X_i = v$
                a. Reduce X_j's domain to values that satisfy c knowing $D_i = \{v\}$
                b. Arc Consistency (2-consistency)

c. After pruning D_j if $D_j = 0$
   i. Cannot find legal assignment!
   ii. Stop expanding that branch!

d. 

7. AC-3 + REVISE: Forward Checking Neighbors
   a. queue ← $\{C(X_i, X_j)\}_{(i, j)}$ # All constraints (assume to be binary)

      while queue not empty:

          $C(X_i, X_j)$ ← queue.pop()

          D_i , D_j ←domains of X_i & X_j

          Revised ← False

          for each x_i in D_i:

              if no x_j in D_j satisfies $C(X_i, X_j)$:

              “““

              Possible Implementation:

              for xj in Dj:

                  if {Xi=xi , Xj=xj} satisfies C(Xi, Xj):

                      return False

              return True

              ”””

                  D_i.remove(x_i)

                  Revised ← True

          if revised is True:

          “““

          If Di becomes empty after revision, that means the constraint C(Xi, Xj) cannot be satisfied. Because there exists a constraint that cannot be satisfied, announce Failure.

          ”””

              if D_i empty.

                  return False

              for each X_k in X_i.neighbors.remove(X_j):

                  queue.append($C(X_k, X_i)$)

      return True
   b. If all constraints are satisfied, AC-3 returns True; otherwise, AC-3 returns False
   c. Sometimes AC-3 finds the solution too!