Lecture 10

Backtracking Algorithm I

Lusi Li

Department of Computer Science

ODU

Reading for This Class:

Chapter 6, Russell and Norvig

Review

- **Last Class**
  - Constraint Satisfaction Problems
  - Definitions, toy and real-world examples

- **This Class**
  - Backtracking Algorithm I
  - Basic algorithms to solve CSPs

- **Next Class**
  - Backtracking Algorithm II
  - Pruning space through propagating information

Review: Constraint Satisfaction Problem

- A CSP is defined as a triple <*V, D, C*>
  - finite set of variables $V = \{V_1, V_2, \ldots, V_n\}$
  - non-empty domains of possible values for each variable

    $D = \{D_{V1}, D_{V2}, \ldots, D_{Vn}\}$
  - finite set of constraints $C = \{C_1, C_2, \ldots, C_m\}$ that specify allowable combinations of values
    - each constraint consists of a pair <scope, relation>

- A state is an assignment of values to some or all variables.
- A solution to a CSP is a complete and consistent assignment.

CSP as a standard search problem

- Let's start with a straightforward approach, then fix it

- States: a partial assignment to n variables, made so far
- Initial state: an empty assignment, { }
- Action:
  - assign a value to an unassigned variable that does not conflict with current assignment
  - fail if no legal assignments (not fixable)
- Goal test: assignment consistent (no violations) and complete (all variables assigned)
- Step cost: constant

Why not using BFS?

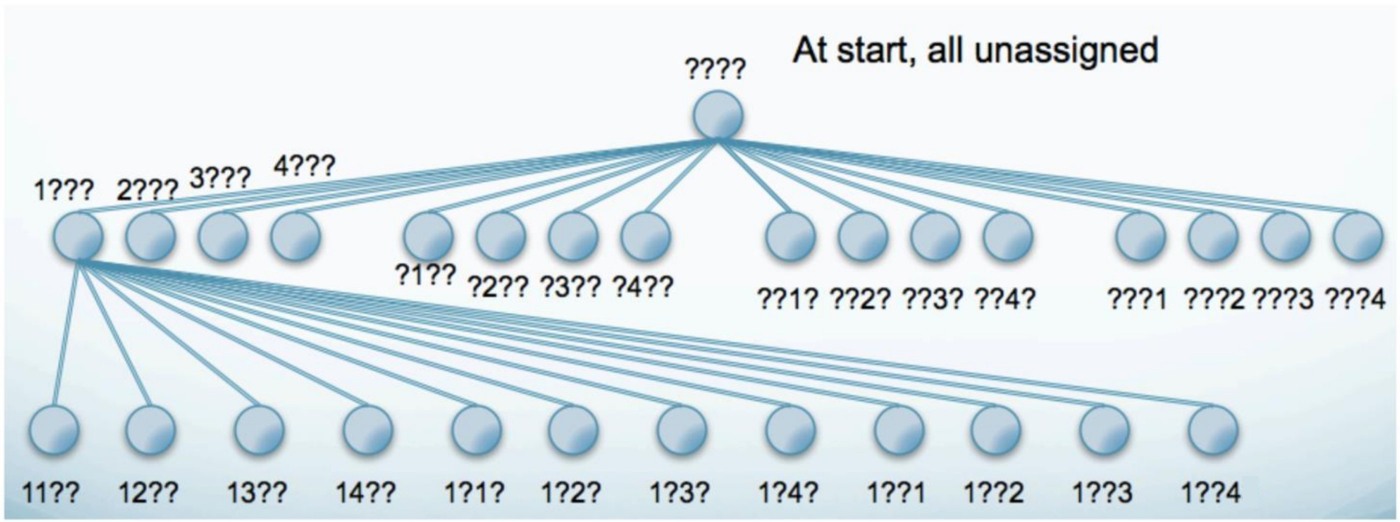
- This is the same for all CSPs!
- Solution is found at depth n, using depth-limited DFS
- Size of the search tree?

CSP as a standard tree search problem

- **For example, *n* = 4 variables each taking *d* = 4 values**

$b = nd$

$b=(n-1)d$



- **Use Depth-limited Search to solve CSP**
  - **branching factor *b = (n-m+1)d* at depth m (m>=1)**
  - **generate a search tree of $n!d^n$ leaves in the worst case even though there are only d^n possible complete assignments!**
- **How to eliminate duplicate assignments?**

Commutativity of CSP

- **The order of assigning the variables has no effect on the final outcome**
- **CSPs are commutative:**
  - **Assigning values to variables in different orders will arrive at the same state**

- **Variable assignments are commutative, i.e.,**
  - **[WA = red then NT = green] same as [NT = green then WA = red]**

    They are the same assignment but in different paths in a search tree

- **Don't care about path!**

- **Only need to consider assignments to a single variable at each node**
  - **branching factor $b = (n-m+1)d$ at depth m is changed into $b = d$ at any depth**
  - **there are d^n leaves**

Backtracking Search

- Idea 1: Only consider a single variable at each node
  - Don't care about path
- Idea 2: Only consider values which do not conflict with assignment made so far
- Depth-first search for CSPs with these two improvements is called **backtracking search**
- Backtracking search is a basic uninformed algorithm for CSPs

Backtracking Search Algorithm

function BACKTRACKING-SEARCH(*csp*) return a solution or failure

- **return BACKTRACK (*{} , csp*)**

function BACKTRACK (*assignment, csp*) return a solution or failure

- **if *assignment* is complete then return *assignment***
- ***var* ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)**
- **for each *value* in ORDER-DOMAIN-VALUES(*var, assignment, csp*) do**
  - **if *value* is consistent with *assignment* according to CONSTRAINTS[*csp*] then**

    add *{var=value}* to assignment

    ***result* ← BACKTRACK(*assignment, csp*)**

    if *result* ≠ *failure* then return *result*

    ***else* remove {*var=value*} from assignment**
  - **end if**
- **end for**
- **return *failure***

Review: Class Scheduling Example

- **CSP Example:**
  - **4 required classes to graduate: A, B, C, D**
  - **A must be taken same semester as D**
  - **C is a prereq for D and B so must take C earlier than D & B**
  - **A & B are always offered at the same time, so they cannot be taken the same semester**
  - **3 semesters (semester 1,2,3) when can take classes**

- **Formulation:**
- **VARIABLES: A,B,C,D**
- **DOMAIN: {1,2,3}**
- **CONSTRAINTS: $A \neq B$, $A=D$, $C < B$, $C < D$**

Class Scheduling Example

- VARIABLES: A,B,C,D
- DOMAIN: {1,2,3}
- CONSTRAINTS: $A \neq B$, $A=D$, $C < B$, $C < D$
- Variable order: ALPHABETICAL
- Value order: DESCENDING

- () // initial assignment

Class Scheduling Example

- VARIABLES: A,B,C,D
- DOMAIN: {1,2,3}
- CONSTRAINTS: $A \neq B$, $A=D$, $C < B$, $C < D$
- Variable order: ALPHABETICAL
- Value order: DESCENDING

- (A=3)
- (A=3, B=3) inconsistent with $A \neq B$
- (A=3, B=2)
- (A=3, B=2, C=3) inconsistent with $C < B$
- (A=3, B=2, C=2) inconsistent with $C < B$
- (A=3, B=2, C=1)
- (A=3, B=2, C=1,D=3) VALID

- The number of trials of assigning values to variables is 7

Class Scheduling Example

- **VARIABLES: A,B,C,D**
- **DOMAIN: {1,2,3}**
- **CONSTRAINTS: A ≠ B, A=D, C < B, C < D**
- **Variable order: ALPHABETICAL**
- **Value order: ASCENDING**

- **()**

Class Scheduling Example

- VARIABLES: A,B,C,D
- DOMAIN: {1,2,3}
- CONSTRAINTS: $A \neq B$, $A=D$, $C < B$, $C < D$
- Variable order: ALPHABETICAL
- Value order: ASCENDING

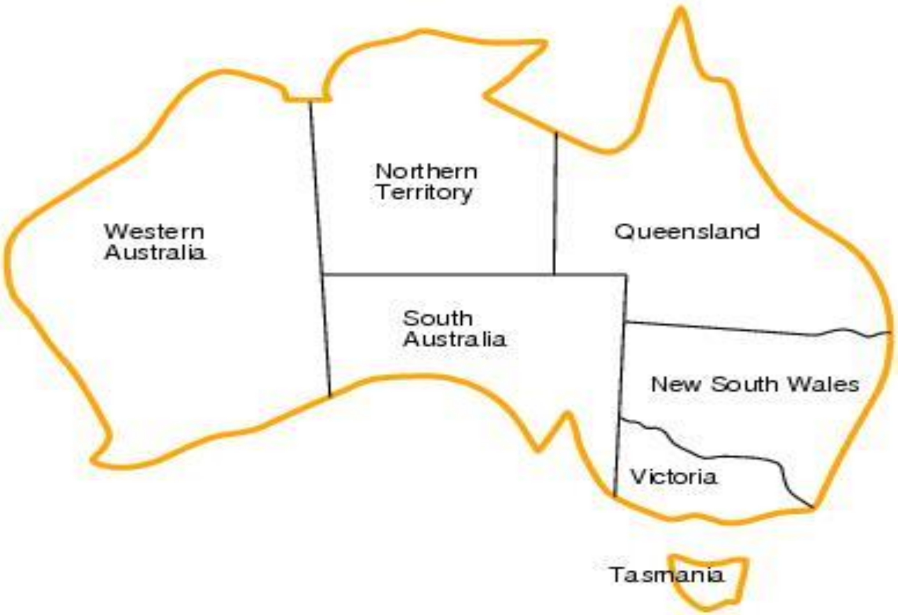
- (A=1)
- (A=1,B=1) inconsistent with $A \neq B$
- (A=1,B=2)
- (A=1,B=2,C=1)
- (A=1,B=2,C=1,D=1) inconsistent with $C < D$
- (A=1,B=2,C=1,D=2) inconsistent with $A=D$
- (A=1,B=2,C=1,D=3) inconsistent with $A=D$
- ……

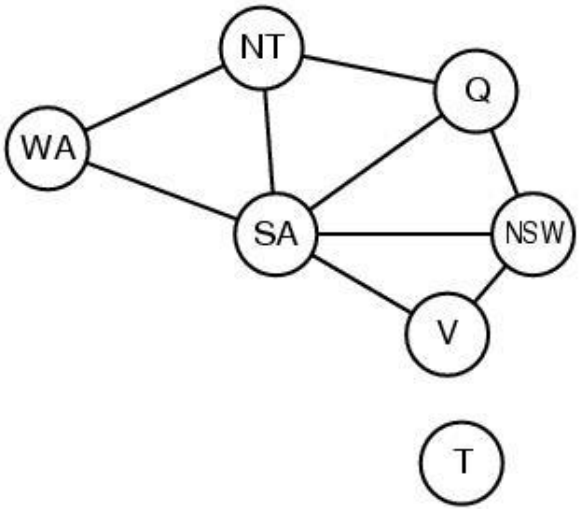
VARIABLES: A,B,C,D DOMAIN: {1,2,3} CONSTRAINTS: A ≠ B, A=D, C < B, C < D
Variable order: ALPHABETICAL Value order: ASCENDING

- (A=1)
- (A=1,B=1) inconsistent with A≠ B
- (A=1,B=2)
- (A=1,B=2,C=1)
- (A=1,B=2,C=1,D=1) inconsistent with C < D
- (A=1,B=2,C=1,D=2) inconsistent with A=D
- (A=1,B=2,C=1,D=3) inconsistent with A=D
- No valid assignment for D, return result = fail
  - Backtrack to (A=1,B=2,C=)
- (A=1,B=2,C=2) but inconsistent with C < B
- (A=1,B=2,C=3) but inconsistent with C < B
- No valid assignments for C, return result = fail
  - Backtrack to (A=1,B=)
- (A=1,B=3)
- (A=1,B=3,C=1)
- (A=1,B=3,C=1,D=1) inconsistent with C < D
- (A=1,B=3,C=1,D=2) inconsistent with A = D
- (A=1,B=3,C=1,D=3) inconsistent with A = D
- No valid assignments for D, return result = fail
  - Backtrack to (A=1,B=3,C=)
- (A=1,B=3,C=2) inconsistent with C < B
- (A=1,B=3,C=3) inconsistent with C < B
- No valid assignments for C, return fail
  - Backtrack to (A=1,B=)
- No valid assignments for B, return fail
  - Backtrack to A
- (A=2)
- (A=2,B=1)
- (A=2,B=1,C=1) inconsistent with C < B
- (A=2,B=1,C=2) inconsistent with C < B
- (A=2,B=1,C=3) inconsistent with C < B
- No valid assignments for C, return fail
  - Backtrack to (A=2,B=?)
- (A=2,B=2) inconsistent with A≠ B
- (A=2,B=3)
- (A=2,B=3,C=1)
- (A=2,B=3,C=1,D=1) inconsistent with C < D
- (A=2,B=3,C=1,D=2) ALL VALID

- The number of trials of assigning values to variables is 26

Backtracking example

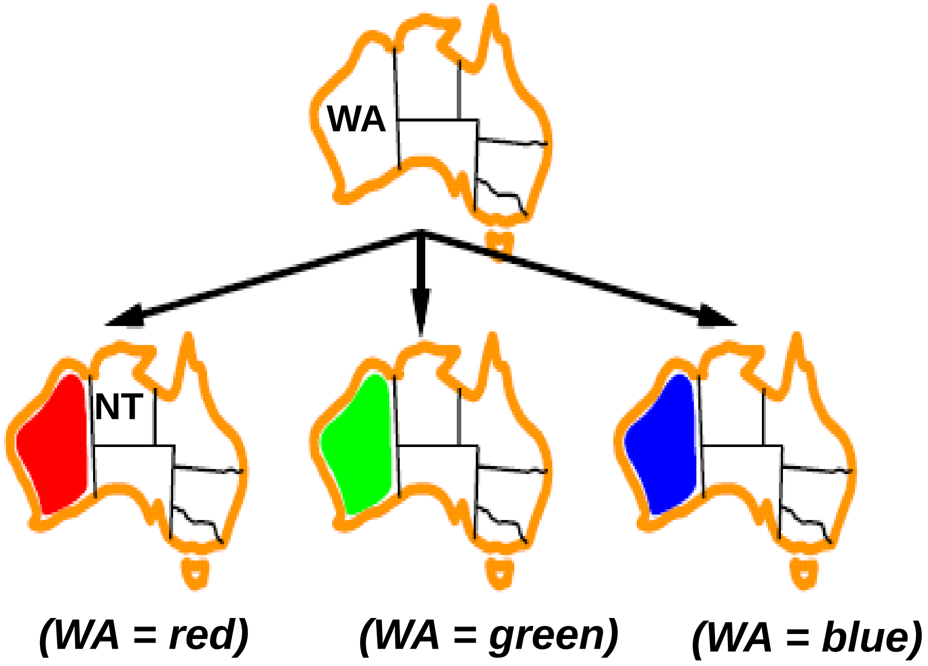




- **Variables: V = {*WA, NT, Q, NSW, V, SA, T*}**
- **Domains: D_i=*{red, green, blue}***
- **Constraints: adjacent regions must have different colors.**
  - C = {WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q, SA≠NSW, SA≠V, Q≠NSW, NSW≠V}

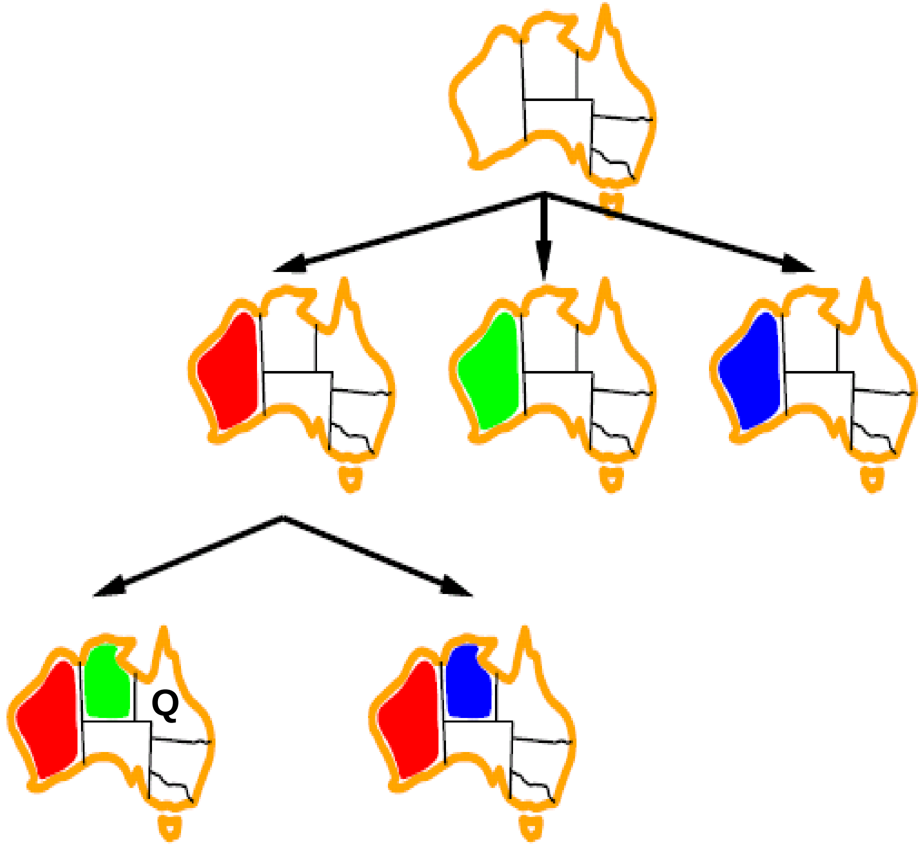
Backtracking example

- **Variables: V = {*WA, NT, Q, NSW, V, SA, T}***
- **Domains: *D_i={red, green, blue}***
- **Constraints: adjacent regions must have different colors.**
  - C = {WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q, SA≠NSW, SA≠V, Q≠NSW, NSW≠V}
- **Variable order: WA, NT, Q, NSW, V, SA, T**
- **Value order: red, green, blue**



Backtracking example

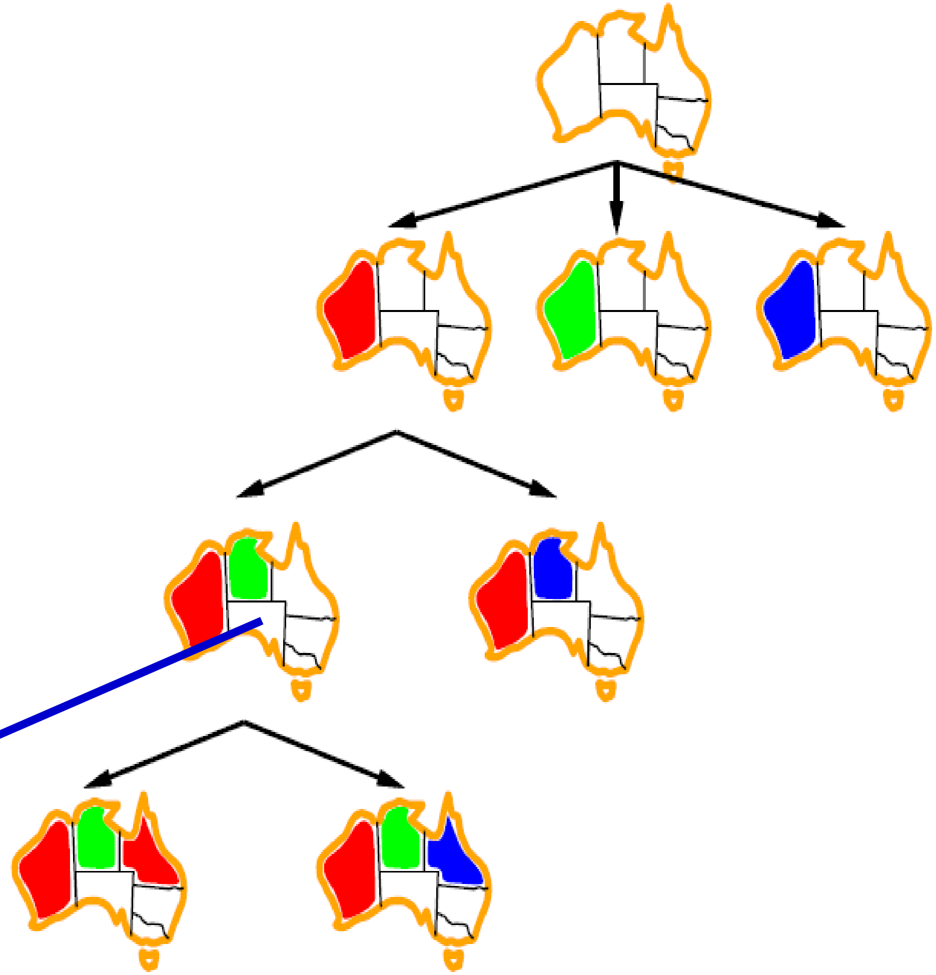
- **Variables: V = {*WA, NT, Q, NSW, V, SA, T*}**
- **Domains: *D$_i$={red, green, blue}***
- **Constraints: adjacent regions must have different colors.**
  - C = {WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q, SA≠NSW, SA≠V, Q≠NSW, NSW≠V}
- **Variable order: WA, NT, Q, NSW, V, SA, T**
- **Value order: red, green, blue**



(WA = red, NT = green) ***(WA = red, NT = blue)***

Backtracking example

- **Variables: V = {*WA, NT, Q, NSW, V, SA, T*}**
- **Domains: *D_i={red, green, blue}***
- **Constraints: adjacent regions must have different colors.**
  - C = {WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q, SA≠NSW, SA≠V, Q≠NSW, NSW≠V}
- **Variable order: WA, NT, Q, NSW, V, SA, T**
- **Value order: red, green, blue**



If SA is selected, SA = blue

Variable ordering matters!

(WA = red, NT = green, Q=red) ***(WA = red, NT = green, Q=blue)***

Improving Backtracking Efficiency

- **Previous improvements on uninformed search**
  → introduce heuristics

- **For CSPs, general-purpose heuristic methods can give large gains in speed, e.g.,**
  - **Ordering:**
    - **Which variable should be assigned next?**
    - **In what order should its values be tried?**

Focus!

  - **Inference (constraint propagation):**
    - **Can we detect inevitable failure early?**
  - **Structure:**
    - **Can we take advantage of problem structure?**

  - **They can be used in combination!**

Variable Ordering

function BACKTRACKING-SEARCH(*csp*) return a solution or failure

- **return BACKTRACK (*{}* , *csp*)**

function BACKTRACK (*assignment, csp*) return a solution or failure

- **if *assignment* is complete then return *assignment***
- ***var* ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)**
- **for each *value* in ORDER-DOMAIN-VALUES(*var, assignment, csp*) do**
  - **if *value* is consistent with *assignment* according to CONSTRAINTS[*csp*] then**

    add *{var=value}* to assignment

    ***result* ← BACKTRACK(*assignment, csp*)**

    if *result* ≠ *failure* then return *result*

    ***else* remove {*var=value*} from assignment**
  - **end if**
- **end for**
- **return *failure***

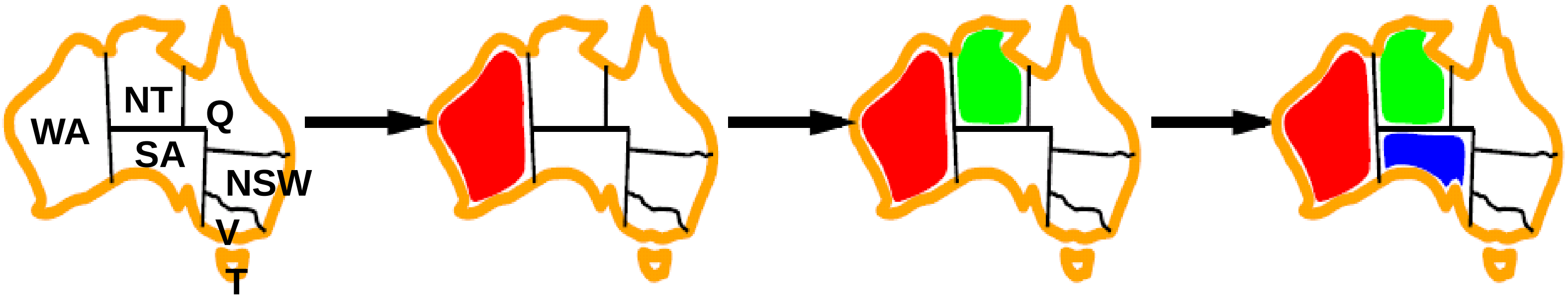
Variable Ordering

- **Static ordering**
  - the order of the variables is specified before the search begins, and it is not changed thereafter

- **Dynamic ordering**
  - the choice of next variable to be considered at any point depends on the current state of the search
  - needs extra information (heuristics) to look ahead

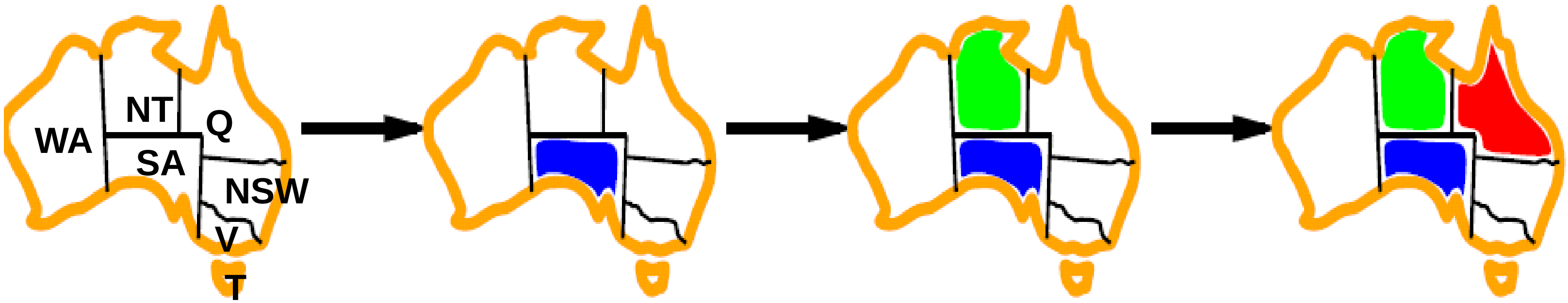
  - two heuristics:
    - minimum remaining values (MRV) heuristic
    - degree heuristic
  - When choosing a variable, apply the MRV heuristic first.
  - Whenever there is a constraint, use the degree heuristic to break it.

Variable Ordering: minimum remaining values (MRV) (only)



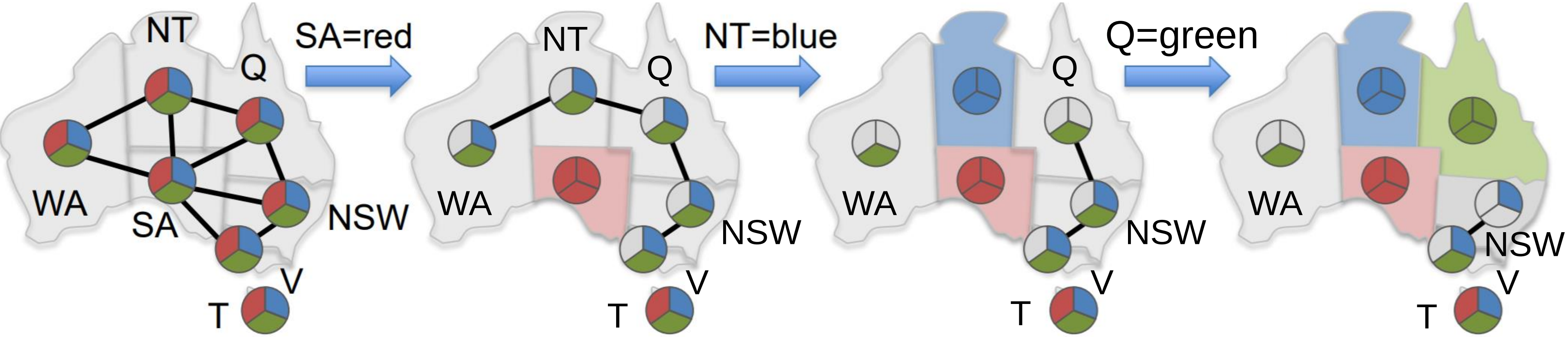
- *Heuristic Rule*: choose variable with the fewest legal moves
- will immediately detect failure if X has no legal values
- a.k.a. most constrained variable
- “Fail first”
- Implementation: keep track of remaining legal values for unassigned variables

Variable Ordering: degree heuristic (only)



- *Heuristic Rule*: choose variable that is involved in the largest number of constraints on other unassigned variables.
- Often applied when all variables have the same number of values
- Try to cut off search ASAP
- Degree heuristic can be useful as a tie-breaker.
- Implementation: keep track of the updated degrees for unassigned variables

Detailed Example: MRV + Degree



- Initially, all variables have 3 values; tie-breaker degree => SA, e.g., assign red
  - No neighbor can be red; we remove the edges to assist in counting degree
- Now, WA, NT, Q, NSW, V have 2 values each
  - WA,V have degree 1; NT,Q,NSW all have degree 2
  - Select one at random, e.g. NT; assign it a value, e.g., blue
- Now, WA and Q have only one possible value; degree(Q)=1 > degree(WA)=0
  - We will solve the remaining problem with no search
- Now, NSW=blue; WA=green; V=green; T={red, green, blue}
- Idea: reduce branching in the future
  - The variable with the largest # of constraints will likely knock out the most values from other variables, reducing the branching factor in the future

Value Ordering

function BACKTRACKING-SEARCH(*csp*) return a solution or failure

- **return BACKTRACK (*{}* , *csp*)**

function BACKTRACK (*assignment, csp*) return a solution or failure

- **if *assignment* is complete then return *assignment***
- ***var* ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)**
- **for each *value* in ORDER-DOMAIN-VALUES(*var, assignment, csp*) do**
  - **if *value* is consistent with *assignment* according to CONSTRAINTS[*csp*] then**

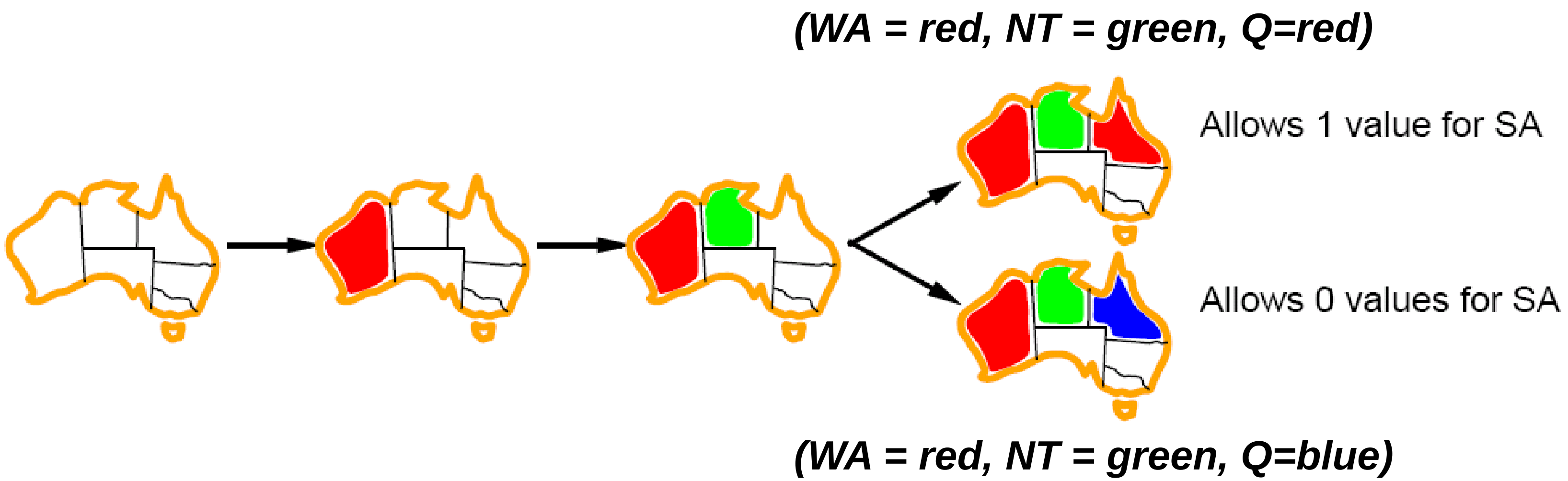
    add *{var=value}* to assignment

    ***result* ← BACKTRACK(*assignment, csp*)**

    if *result* ≠ *failure* then return *result*

    ***else* remove {*var=value*} from assignment**
  - **end if**
- **end for**
- **return *failure***

Least constraining value for value-ordering (only)



- **Heuristic for selecting what value to try next**
- **Heuristic Rule: given a variable, choose the least constraining value**
  - **the one that rules out the fewest values in the remaining variables**
  - **leaves the maximum flexibility for subsequent variable assignments**
- **"Fail last": selecting value least likely to cause future conflicts**
- **Implementation: keep track of remaining legal values for other unassigned variables (excluding the given variable)**

Improving Backtracking Efficiency

- **Why fail first when selecting variables?**
  - Prunes large portions of tree early on
- **Why fail last when selecting values?**
  - Only need one solution, so examine probable values first

Summary

- **Backtracking Algorithm**
  - **Minimum remaining values**
  - **Degree heuristics**
  - **Least constraining values**

What I want you to do

- Review Chapter 6
- Start working on Assignment 2