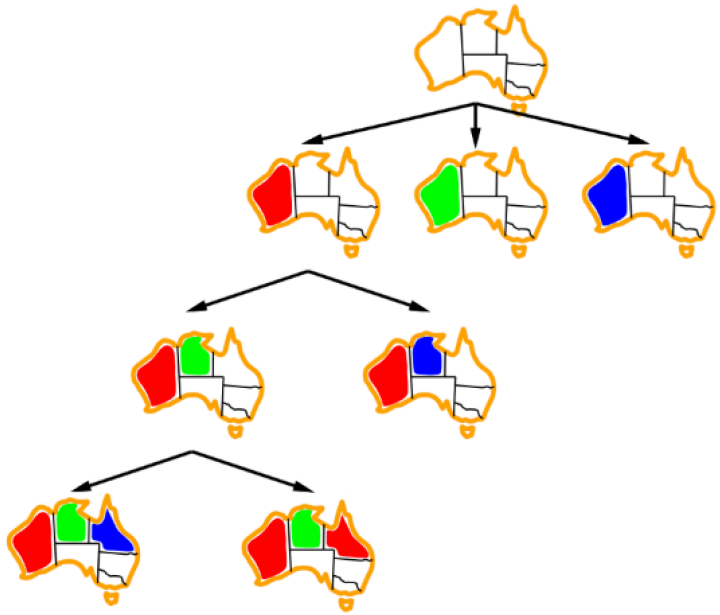
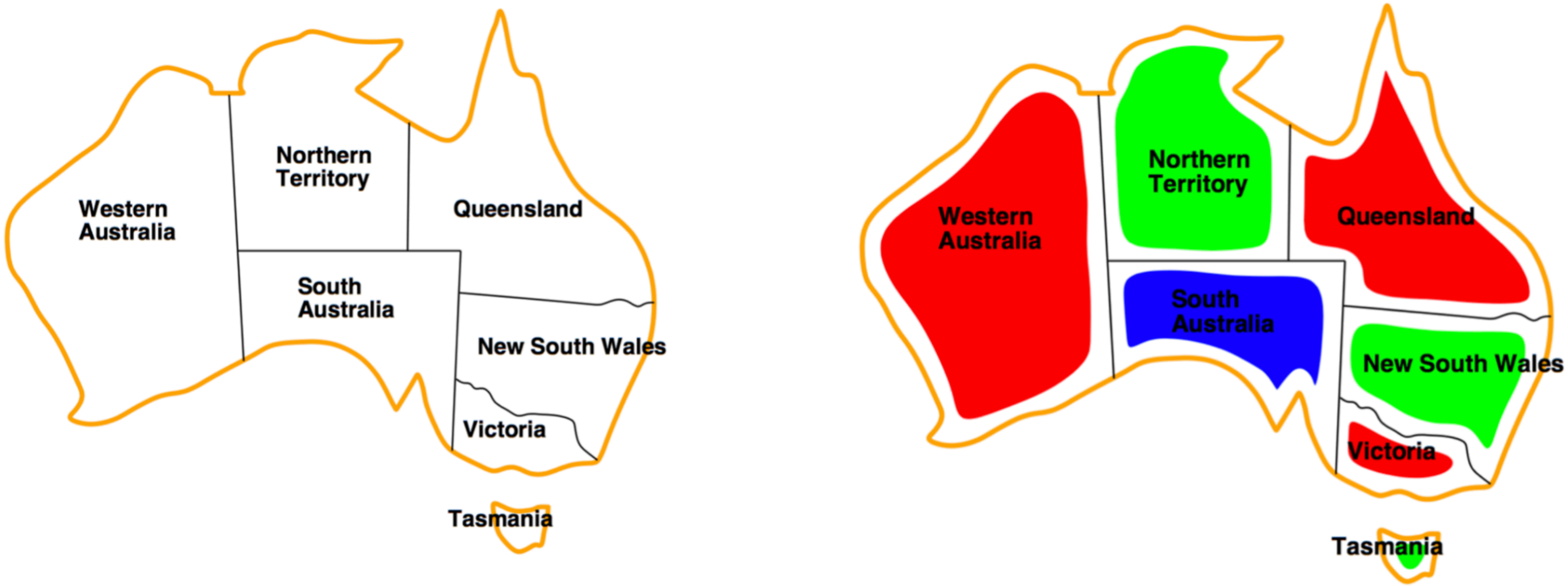
CS143: Artificial Intelligence



Backtracking Search

Drake UNIVERSITY

REVIEW: MAP COLORING IS A CONSTRAINT SATISFACTION PROBLEM (CSP)



- Color each state so that no two adjacent states get the same color. You can use Red, Green, and Blue to color each state (left). One of its solution (right).

Review: CSP Abstraction

- Given a domain
  - possible values for each state in a problem
  - $D = \{D_1, D_2, D_3 \ldots D_n\}$
- Define a "State" by a set of variables X_i
  - $X = \{X_1, X_2, X_3 \ldots X_n\}$
  - Each variable X_i can be one of the values from the Domain
  - Each Domain D_i consists of a set of allowable values
- Define a set of constraints that must be true
  - Goal test is a set of *constraints* specifying *allowable* combinations of values
    - An "Assignment"

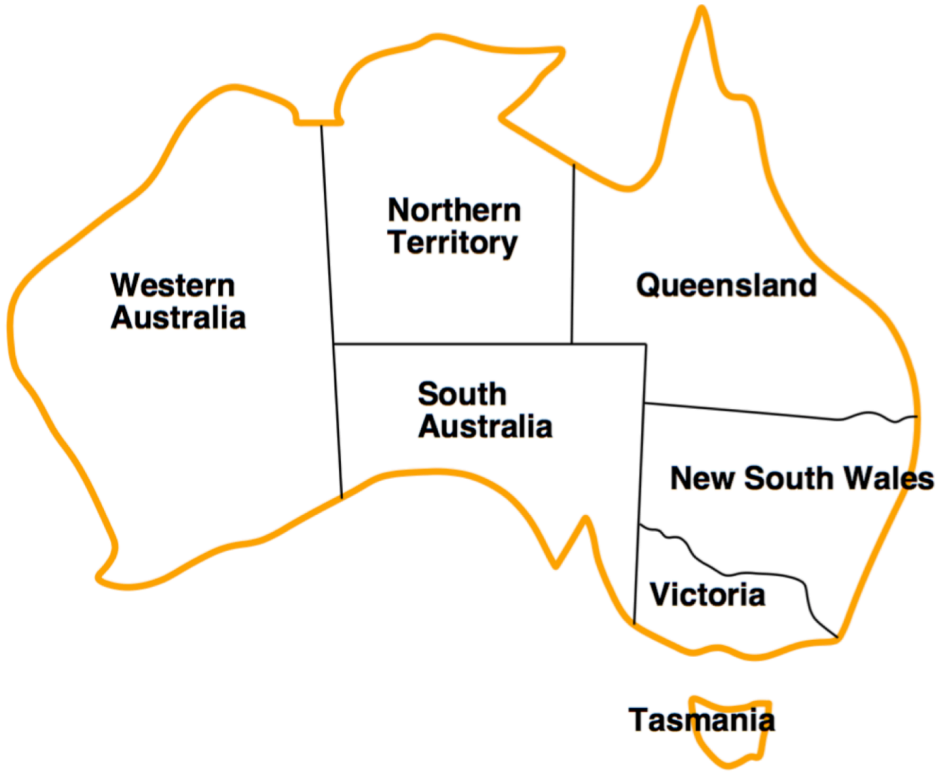
Review: CSP Abstraction

- If we put a problem into the format of having
  - X: a set of variables
  - D: a set of domains
  - C: a set of constraints that specify allowable combinations of values
- Allows useful *general-purpose* algorithms with more power than standard search algorithms

Review: Constraint Satisfaction Problem - Step 1

- Come up with a list of variables and possible values (domains) they can have

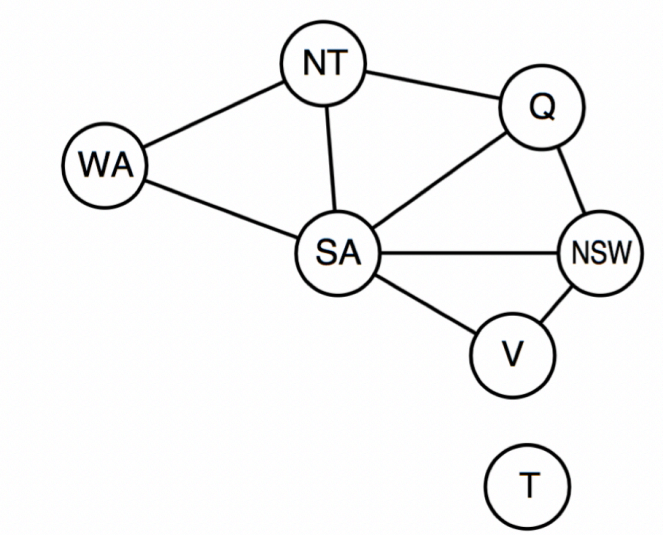
- WA: red, green, blue
- NT: red, green, blue
- SA: red, green, blue
- Q: red, green, blue
- NSW: red, green, blue
- V: red, green, blue
- T: red, green, blue



Review: Constraint Satisfaction Problem - Step 2

- Come up with a list of constraints on the values (variables)

- WA ≠ NT
- WA ≠ SA
- NT ≠ SA
- NT ≠ Q
- SA ≠ Q
- SA ≠ NSW
- SA ≠ V
- Q ≠ NSW
- NSW ≠ V

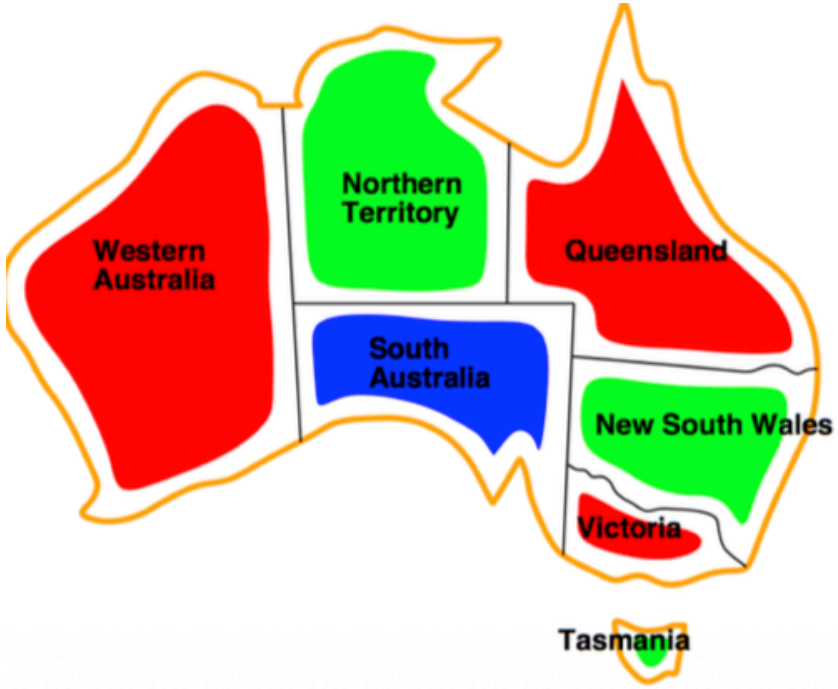


Constraint graph: nodes are variables, arcs show constraints

Review: Constraint Satisfaction Problem - Step 3

- Input variables and constraints into a CSP solving algorithm
- There may be multiple solutions
- A solution is an assignment of values to variables

- WA = red
- NT = green
- SA = blue
- Q = red
- NSW = green
- V = red
- T = green



THE EXERCISE

- You are in charge of scheduling for computer science classes. There are 5 classes and 3 professors who will be teaching these classes. You are constrained by the fact that each professor can only teach one class at a time.

- The classes are:
  - **Class 1** - Intro to Programming: meets from **8:00-9:00am**
  - **Class 2** - Intro to Artificial Intelligence: meets from **8:30-9:30am**
  - **Class 3** - Natural Language Processing: meets from **9:00-10:00am**
  - **Class 4** - Computer Vision: meets from **9:00-10:00am**
  - **Class 5** - Machine Learning: meets from **9:30-10:30am**

- The professors are:
  - **Professor A**, who is available to teach Classes 3 and 4.
  - **Professor B**, who is available to teach Classes 2, 3, 4, and 5.
  - **Professor C**, who is available to teach Classes 1, 2, 3, 4, 5.

- **Exercise:** What are the variables and their domains? What are the constraints?

Backtracking Search

- Backtracking Search is an algorithm that can be used for solving CSPs

- The term *backtracking-search* is used for a depth-first search that
  - chooses values for one variable at a time
  - backtracks when a variable has no legal values left to assign

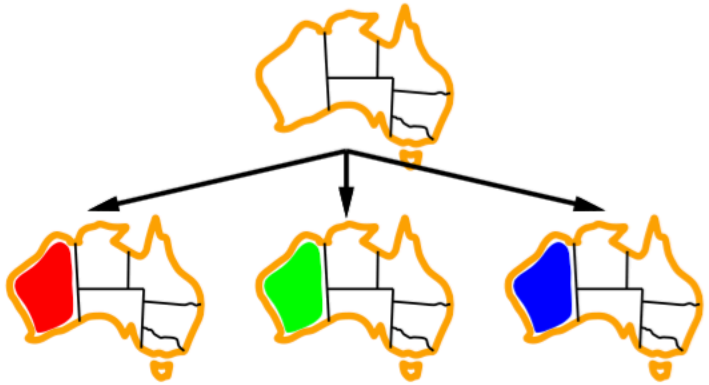
Backtracking Search

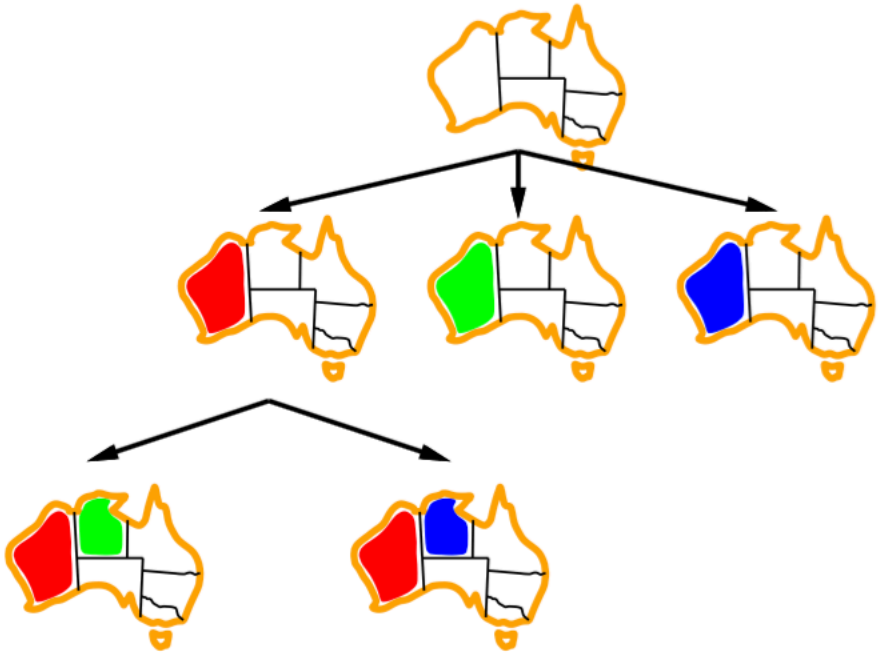
- Backtracking Search is an algorithm that can be used for solving CSPs
  - Chooses an unassigned variable and tries all values in the domain trying to find a solution
  - If an inconsistency is detected, returns failure causing the previous call to try another value

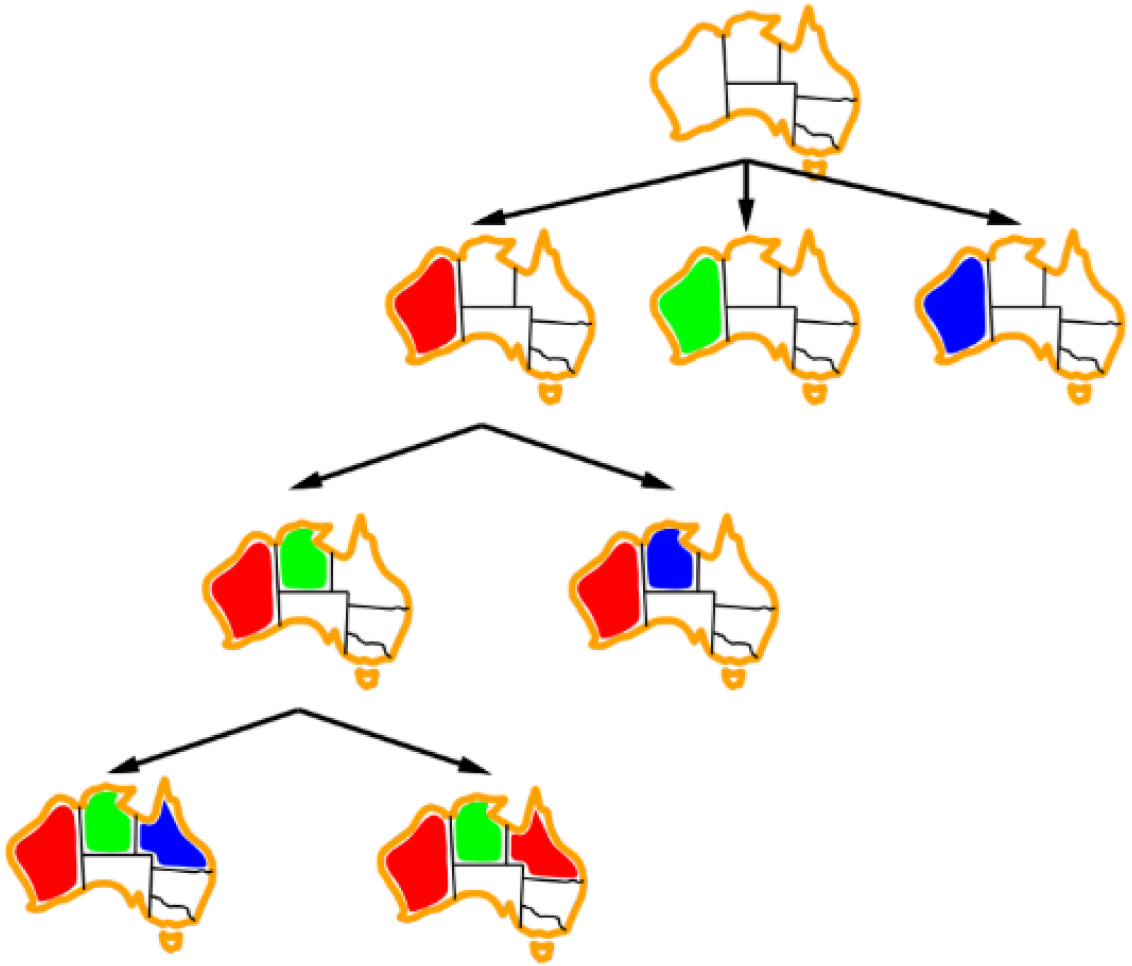
Backtracking

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/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 given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```







Backtracking Search

- How could we make backtracking search faster (or more effective)?
  - It's kind of like an uninformed search
  - Which of the possible successors should be expanded first
    - (akin to greedy best-first search or A* search)

Backtracking

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/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 given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

HOW ????

HOW ????

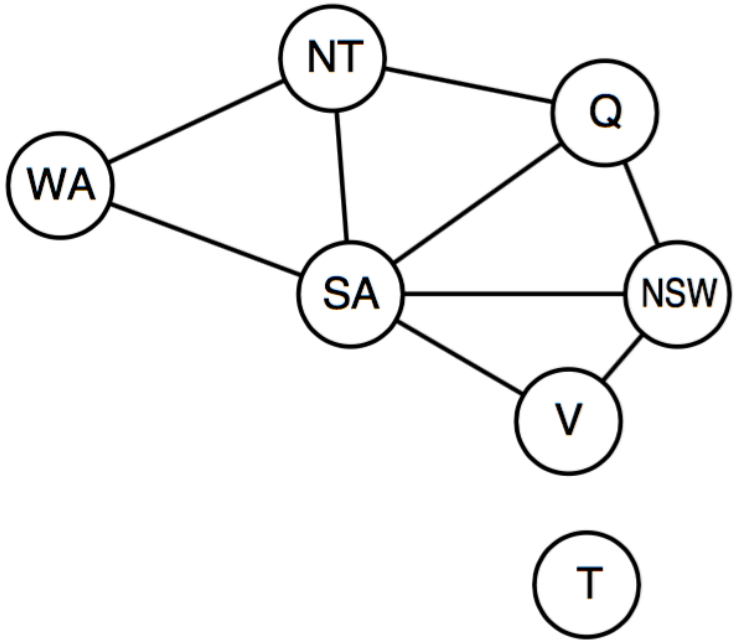
Backtracking Search Questions

1. Which variable should be assigned next?
   - ????

2. In what order should its values be tried?
   - ????

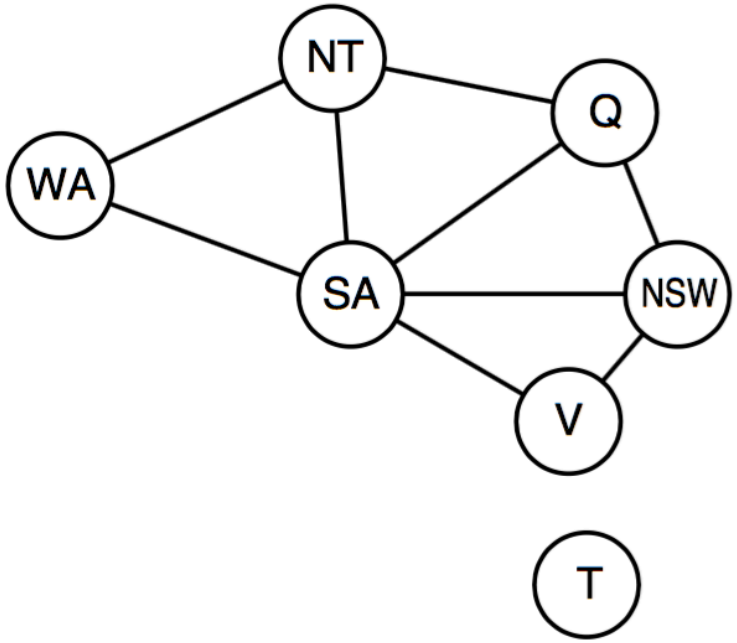
3. Can we detect inevitable failure early?

1. Which variable should be assigned next?



1. Which variable should be assigned next?

- Which one you should pick first: SA, WA, NT, Q, NSW, or V?



- Notice: after SA (and WA) are assigned, the values for NT, Q, NSW, and V are all forced.

Minimum remaining values (MRV)

Fail-first Strategy

- Minimum-remaining-values (MRV) heuristic
  - Most constrained variable
  - Fail-first there by minimize the number of nodes in the search tree

- Idea: choose the most constrained choice that will lead to a possible failure *faster*
  - Thereby eliminating the need to expand unnecessary nodes

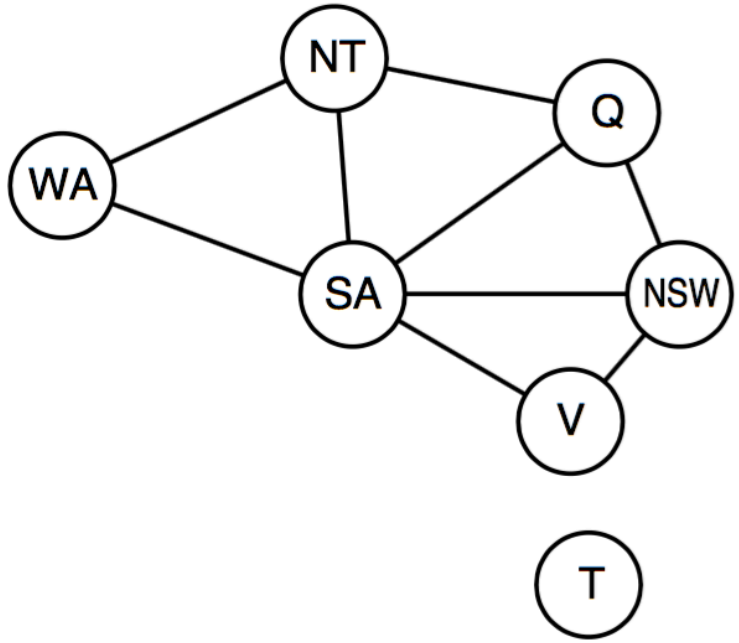
Degree Heuristic

- Minimum-remaining-values (MRV) heuristic doesn't help make the first choice in the Australia map problem, since every region initially has three legal colors.

- Possible tie-breaker:
  - Degree heuristic:
    - Attempts to reduce the branching factor on future choices by selecting the variable that is involved in the largest number of constraints on unassigned variables

Degree Heuristic

- Color SA first, as it will minimize the need to consider it in future branches



Backtracking

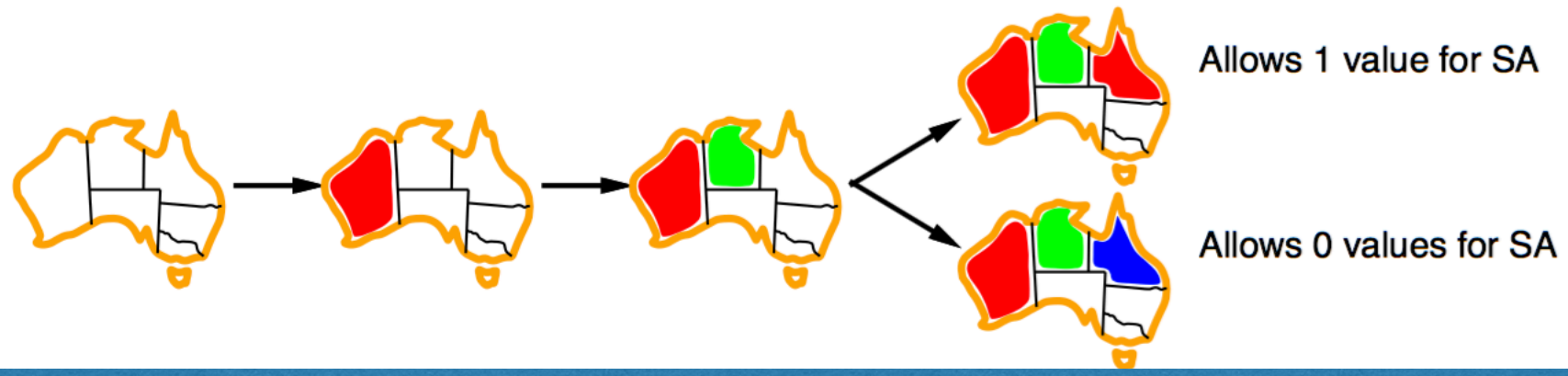
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/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 given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Minimum Remaining Value (MRV)

2. In what order should values be tried?

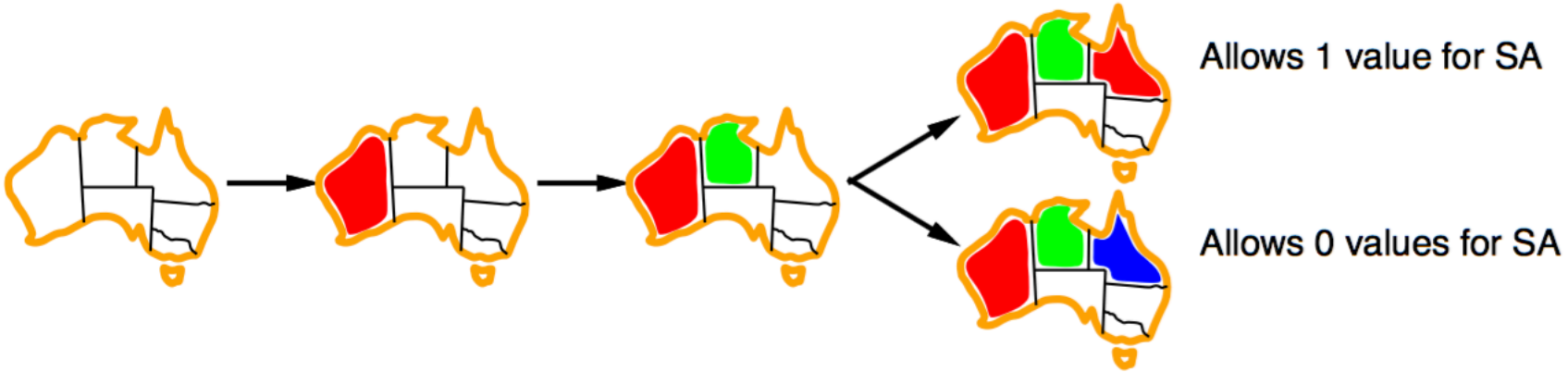
- Once a variable has been selected, what order to examine the values?
- Least-constraining value (LCV):
  - Value that allow for the most possible choices or most flexibility for the neighboring variables in the constraint graph
  - We need just one solution (out of many)



Least Constraining Value (LCV)

Leaving Most Possible Flexibility

- Don’t choose blue here



- Blue will lead to a dead end
- Trying to leave the maximum flexibility for subsequent variable assignments

Backtracking

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/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 given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Least Constraining Value (LCV)

- For value ordering, the trick is that we only need one solution; therefore it makes sense to look for the most likely values first.

Summary

- Variable selection via minimum-remaining value (MRV)
  - fail-first
- Value selection via least-constraining value (LCV)
  - fail-last

- It's a balance:
  - Variable ordering minimize the number of nodes in the search tree
  - Value ordering – searching for one solution

Summary: Answers

1. Which variable should be assigned next?
   - MRV heuristics: i) most constrained variable heuristic (fail-first), ii) degree heuristic

2. In what order should its values be tried?
   - Least-constraining value heuristic (leave most flexibility)

Backtracking

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/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 given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Minimum Remaining Value (MRV)

Least Constraining Value (LCV)

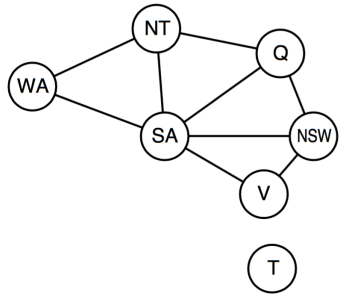
3. Can we detect inevitable failure early?

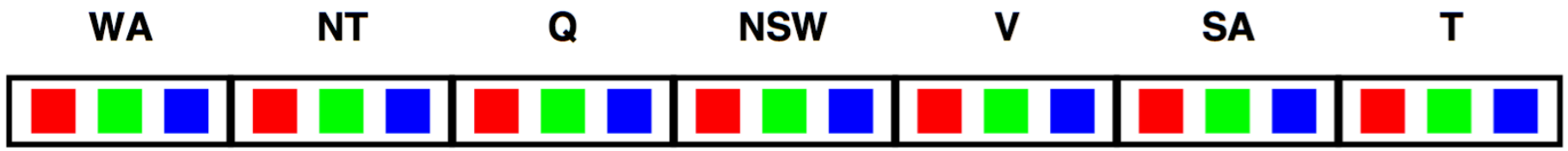
Forward Checking

- Whenever a variable is assigned, forward checking establishes arc consistency for it
  - A variable is **arc-consistent** if every value in it's domain satisfies the binary constraints
  - Delete from other's domain any value inconsistent with the one chosen

Forward Checking

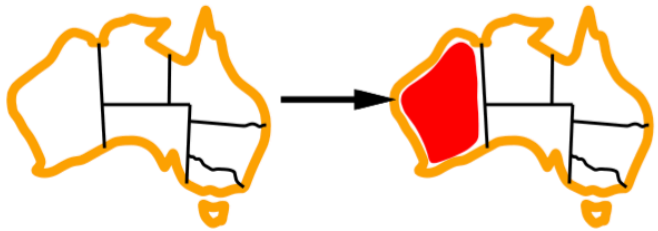
- Idea: Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values

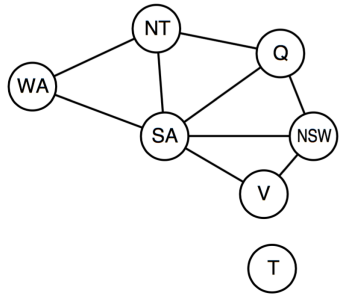




Forward Checking

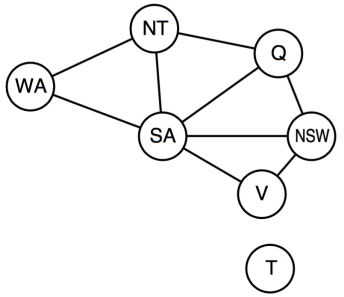
- Idea: Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values





Forward Checking

- Idea: Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values



Forward Checking

- Idea: Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values