

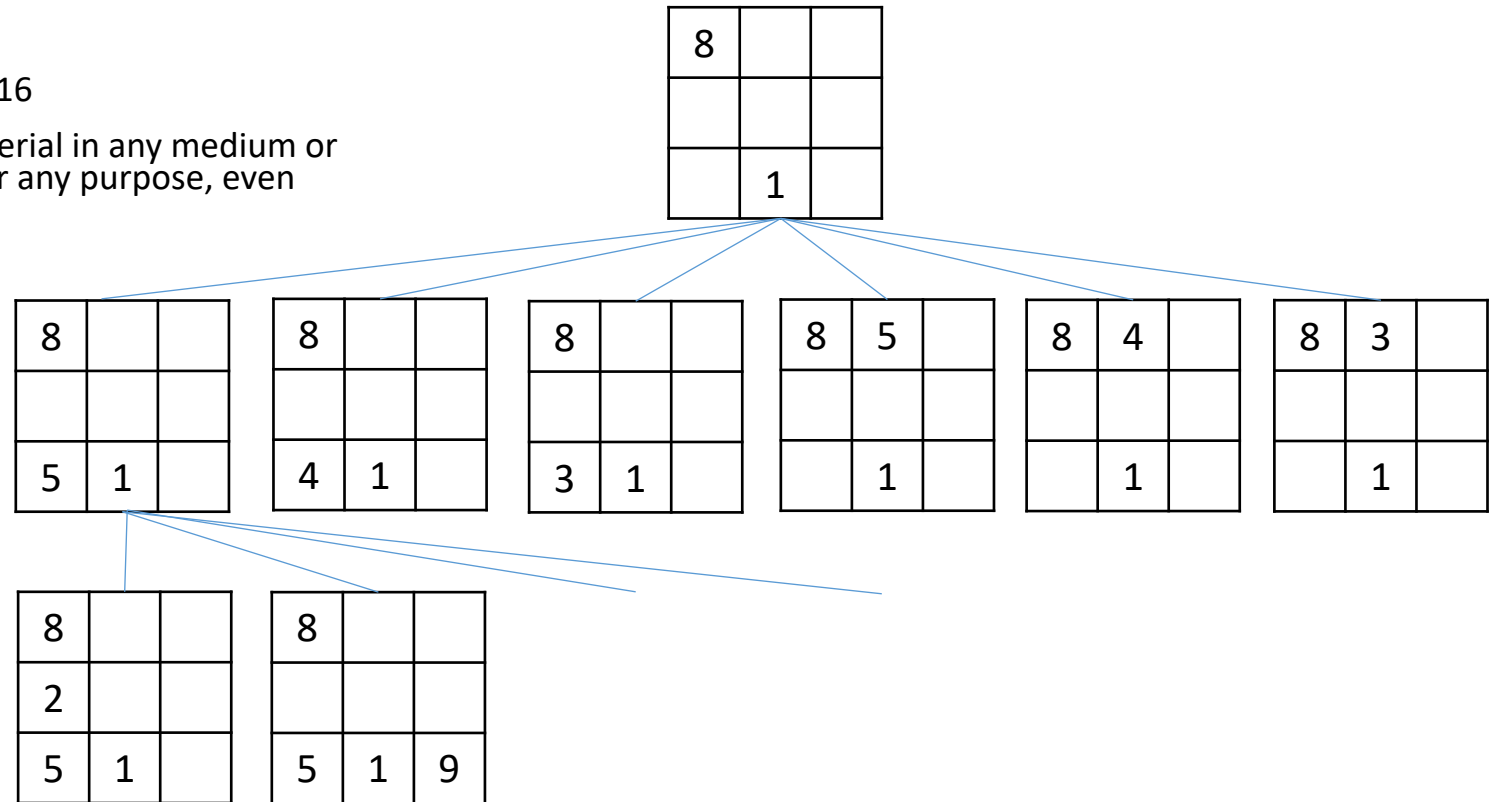
CS440/ECE 448, Lecture 6: Constraint Satisfaction Problems

Slides by Mark Hasegawa-Johnson, 1/2020

Including some slides written by Svetlana Lazebnik, 9/2016

[CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/): You are free to: copy and redistribute the material in any medium or format, remix, transform, and build upon the material for any purpose, even commercially, if you give appropriate credit.

8			4	6			7
					4		
	1				6	5	
5		9		3		7	8
				7			
	4	8		2		1	3
	5	2					9
		1					
3			9	2			5



Content

- What is a CSP? Why is it search? Why is it special?
- Backtracking Search
- $O\{1\}$ heuristics to improve backtracking search
- $O\{N\}$ and $O\{N^2\}$ heuristics: early detection of failure

What is search for?

- Assumptions: single agent, deterministic, fully observable, discrete environment
- **Search for *planning***
 - The path to the goal is the important thing
 - Paths have various costs, depths
- **Search for *assignment***
 - Assign values to variables while respecting certain constraints
 - The goal (complete, consistent assignment) is the important thing



8			4	6		7	
	1				4		
					6	5	
5		9		3		7	8
				7			
	4	8		2		1	3
	5	2					9
		1					
3			9	2			5

Why are Constraint satisfaction problems (CSPs) just a special case of generic search problems?

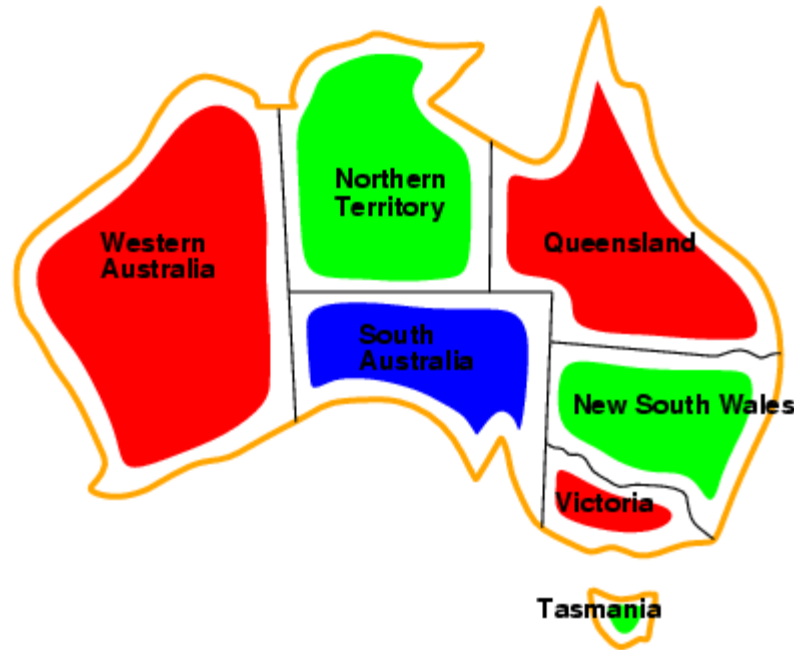
- **State** is defined by **N variables**, each takes one of **D possible values**
- **Goal test** is a set of **constraints** specifying allowable combinations of values for subsets of variables.
- **Solution** is a **complete, consistent** assignment

Example: Map Coloring



- **Variables:** WA, NT, Q, NSW, V, SA, T
- **Domains:** {red, green, blue}
- **Constraints:** adjacent regions must have different colors
 - Logical representation: $WA \neq NT$
 - Set representation: $(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}$

Example: Map Coloring



- **Solutions** are *complete* and *consistent* assignments, e.g.,
WA = red, NT = green, Q = red, NSW = green,
V = red, SA = blue, T = green

Why are Constraint satisfaction problems (CSPs) different from generic search problems?

- Because every path has N steps! So the computational cost of DFS, is the SAME as the cost of BFS, $O\{b^m\} = O\{b^d\} = O\{D^N\}$
 - Path length is N , because there are N variables to assign
 - Branching factor is D , because there are D possible values.
- Meanwhile, space is still a problem. DFS allows us to delete the part of the tree corresponding to an unsuccessful path. So DFS is more useful than BFS.
- Topic of today: how do we use heuristics with DFS?
- Hint: it's not as elegant as A^* . There is no provable optimality. In fact...

Computational complexity of CSPs

- The satisfiability (SAT) problem:

- Given a Boolean formula, is there an assignment of the variables that makes it evaluate to true?

$$(X_1 \vee \bar{X}_7 \vee X_{13}) \wedge (\bar{X}_2 \vee X_{12} \vee X_{25}) \wedge \dots$$

- SAT and CSP are NP-complete

- **NP**: a class of decision problems for which
 - the “yes” answer can be verified in polynomial time
 - no known algorithm can find a “yes” answer, from scratch, in polynomial time
- An **NP-complete** problem is in NP and every other problem in NP can be efficiently reduced to it (Cook, 1971)
- Other NP-complete problems: graph coloring, n-puzzle, generalized sudoku
- It is not known whether P = NP, i.e., no efficient algorithms for solving SAT in general are known

Content

- What is a CSP? Why is it search? Why is it special?
- Backtracking Search
- $O\{1\}$ heuristics to improve backtracking search
- $O\{N\}$ and $O\{N^2\}$ heuristics: early detection of failure

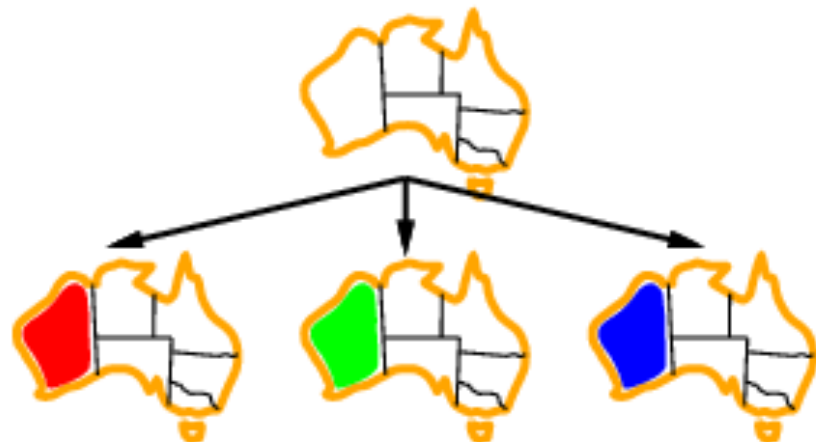
Backtracking search

- In CSP's, variable assignments are **commutative**
 - For example, *[WA = red then NT = green]* is the same as *[NT = green then WA = red]*
- We only need to consider assignments to a single variable at each level (i.e., we fix the order of assignments)
 - There are $N!$ different orderings of the variables. If we choose a particular ordering, and then never change it, we reduce computational complexity by a factor of $N!$
 - With a fixed order, there are still D^N possible paths.
 - At each level, choose one of the D possible assignments, and explore to see if it gives you a solution. If not, **backtrack**: delete the whole sub-tree, and try something different.
- Depth-first search for CSPs with single-variable assignments is called **backtracking search**

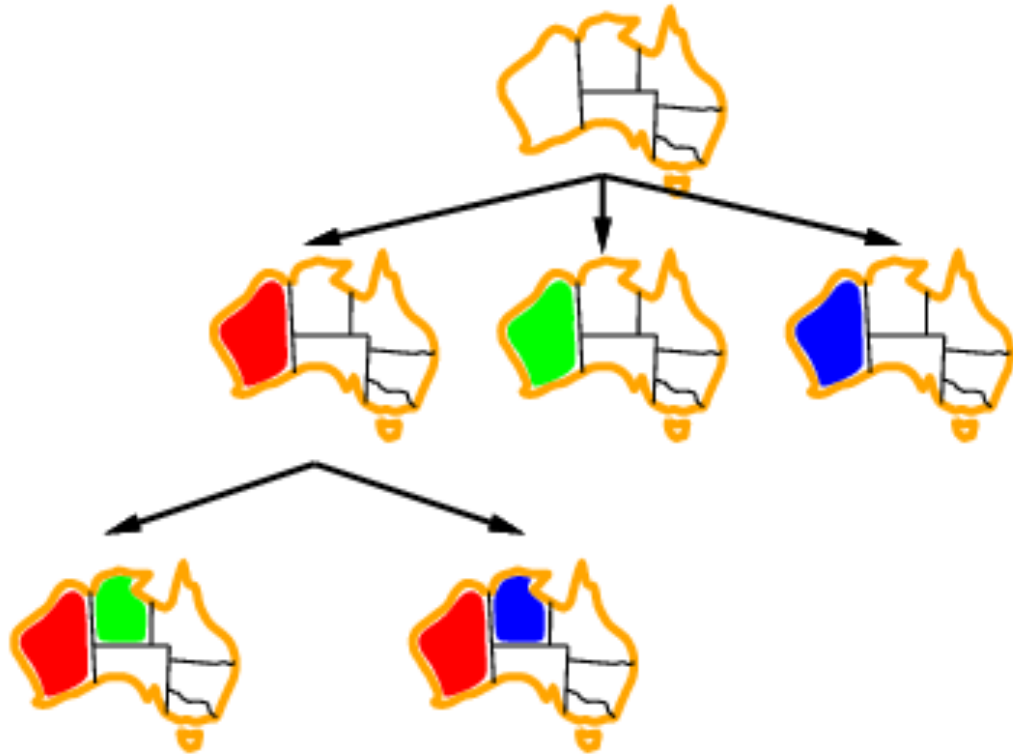
Example



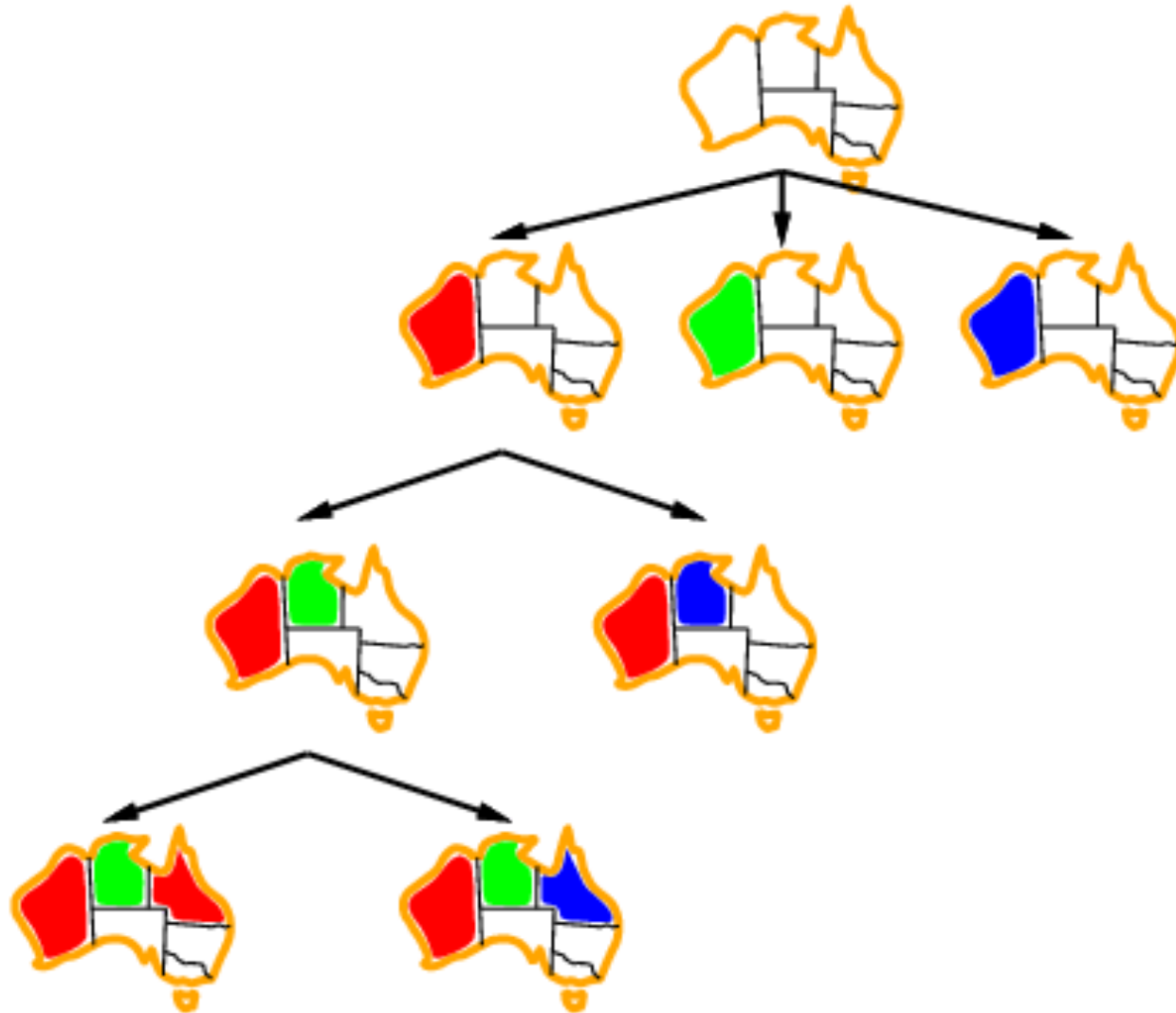
Example



Example



Example



Backtracking search algorithm

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

- Making backtracking search efficient:
 - Which variable should be assigned next?
 - In what order should its values be tried?
 - Can we detect inevitable failure early?

Content


- What is a CSP? Why is it search? Why is it special?
- Backtracking Search
- $O\{1\}$ heuristics to improve backtracking search
- $O\{N\}$ and $O\{N^2\}$ heuristics: early detection of failure

Content

- What is a CSP? Why is it search? Why is it special?
- Backtracking Search
- $O\{1\}$ heuristics to improve backtracking search
 1. Given a particular variable, which value should you assign?
 2. Which variable should you consider next?
- $O\{N\}$ and $O\{N^2\}$ heuristics: early detection of failure

Given a variable, in which order should its values be tried?

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



- Making backtracking search efficient:
 - Which variable should be assigned next?
 - In what order should its values be tried?
 - Can we detect inevitable failure early?

Given a variable, in which order should its values be tried?

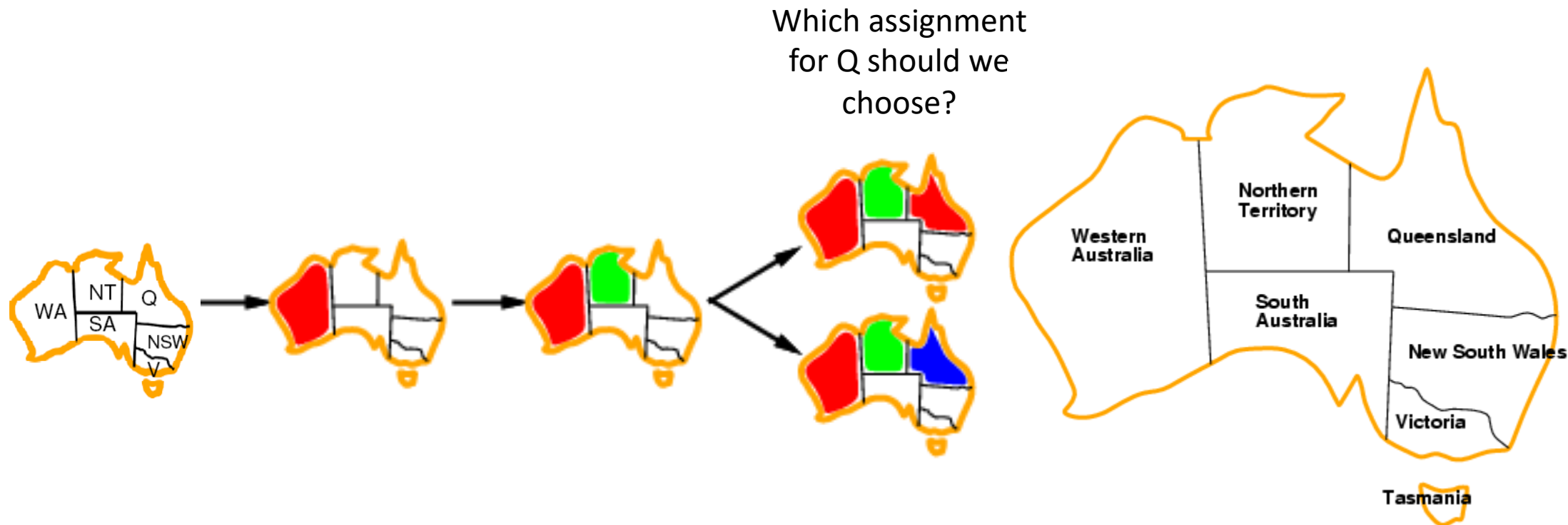
- **Least Constraining Value (LCV) Heuristic:**
 - Try the following assignment first: to the variable you're studying, the value that rules out the fewest values in the remaining variables

- Key intuition: maximize the probability of success.

Given a variable, in which order should its values be tried?

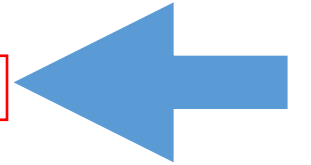
- **Least Constraining Value (LCV) Heuristic:**

- Try the following assignment first: to the variable you're studying, the value that rules out the fewest values in the remaining variables



Which variable should be assigned next?

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



- Making backtracking search efficient:
 - Which variable should be assigned next?
 - In what order should its values be tried?
 - Can we detect inevitable failure early?

Which variable should be assigned next?

- **Key intuitions:**

- If there is a solution possible, it will still be possible, regardless of the order in which you study the variables.
- So choosing a VARIABLE is easier than choosing a VALUE. Just minimize the branching factor.

- **Least Remaining Values (LRV) Heuristic:**

- Choose the variable with the fewest legal values

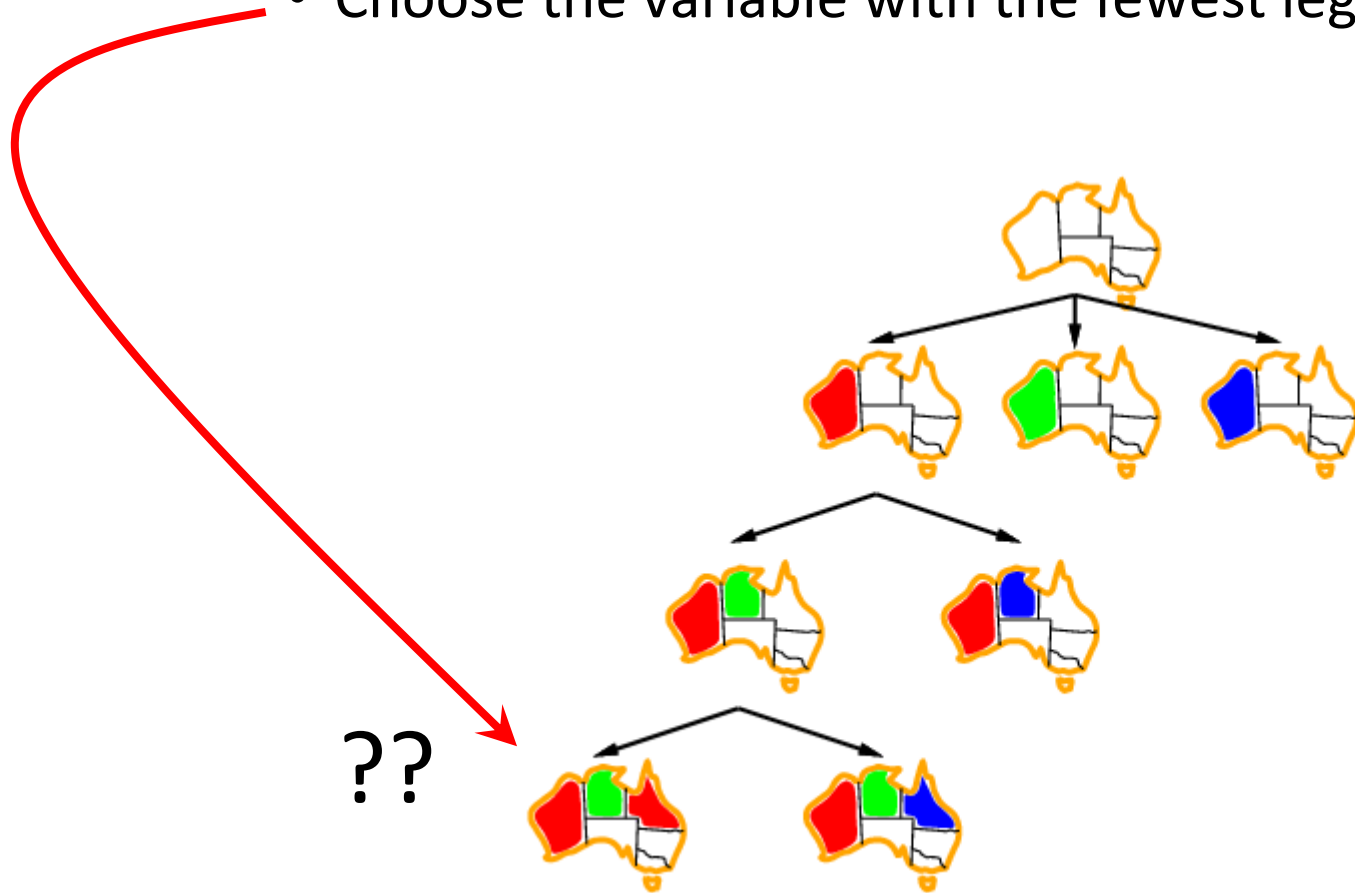
- **Most Constraining Variable (MCV) Heuristic:**

- Choose the variable that imposes the most constraints on the remaining variables

Which variable should be assigned next?

- **Least Remaining Values (LRV) Heuristic:**

- Choose the variable with the fewest legal values



Which variable should be assigned next?

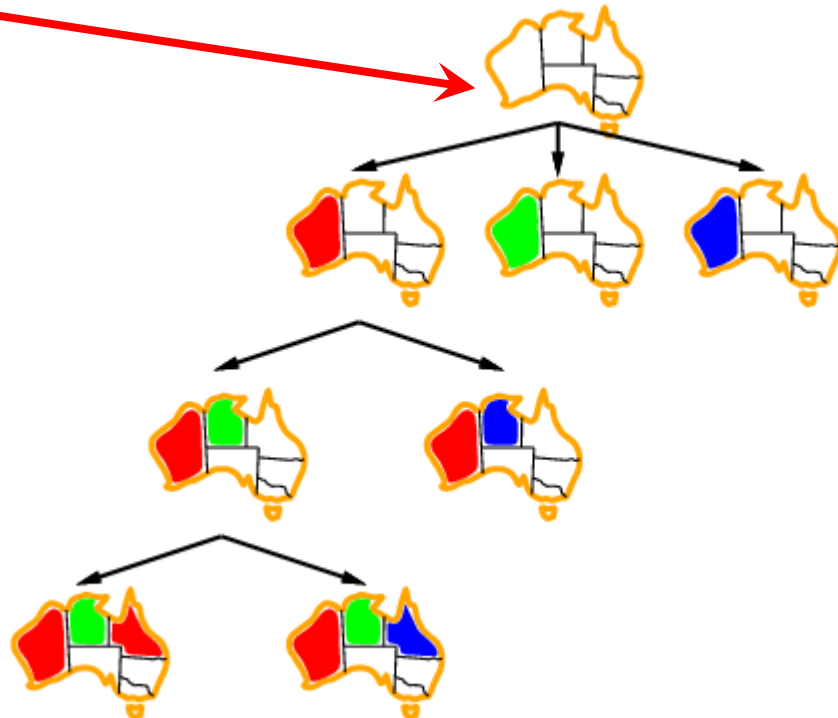
- **Most Constraining Variable (MCV) Heuristic:**
 - Choose the variable that imposes the most constraints on the remaining variables
 - Tie-breaker among variables that have equal numbers of LRV

Which variable should be assigned next?

- **Most Constraining Variable (MCV) Heuristic:**

- Choose the variable that imposes the most constraints on the remaining variables
- Tie-breaker among variables that have equal numbers of MRV

??



Content

- What is a CSP? Why is it search? Why is it special?
- Backtracking Search
- $O\{1\}$ heuristics to improve backtracking search
- $O\{N\}$ and $O\{N^2\}$ heuristics: early detection of failure

Early detection of failure: $O\{N\}$ checking

- **Forward Checking:**

- Check to make sure that every variable still has at least one possible assignment

Early detection of failure: $O\{N\}$ checking

Forward checking

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



Early detection of failure: $O\{N\}$ checking

Forward checking

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



Early detection of failure: $O\{N\}$ checking

Forward checking

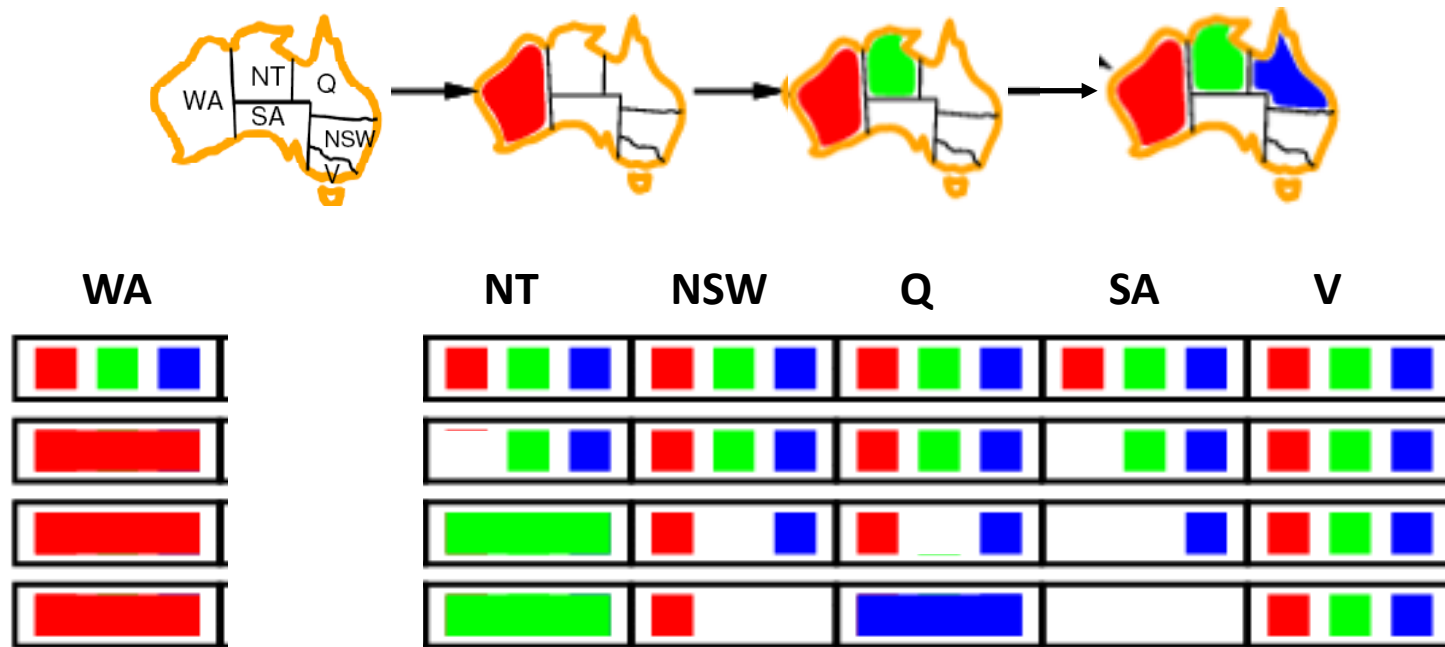
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values



Early detection of failure: $O\{N\}$ checking

Forward checking

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

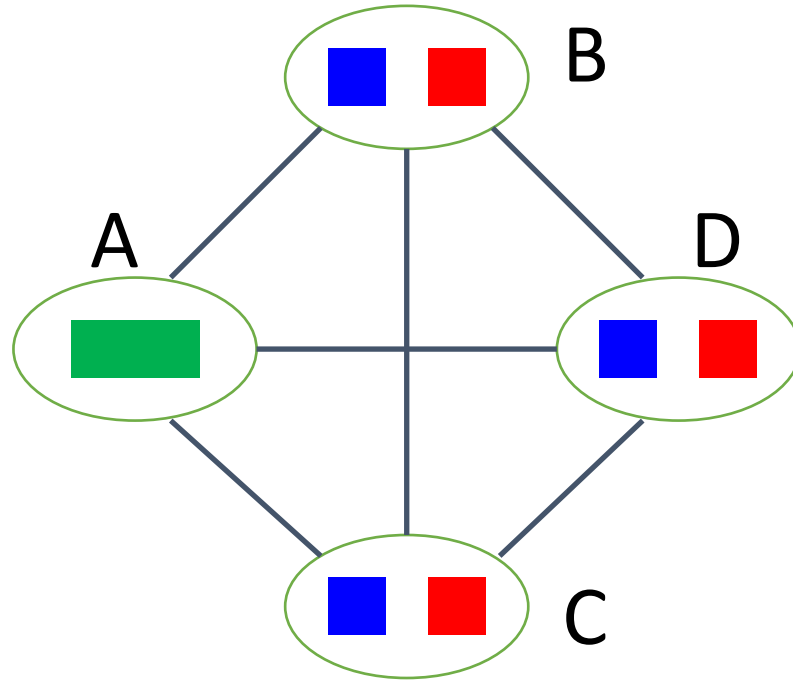
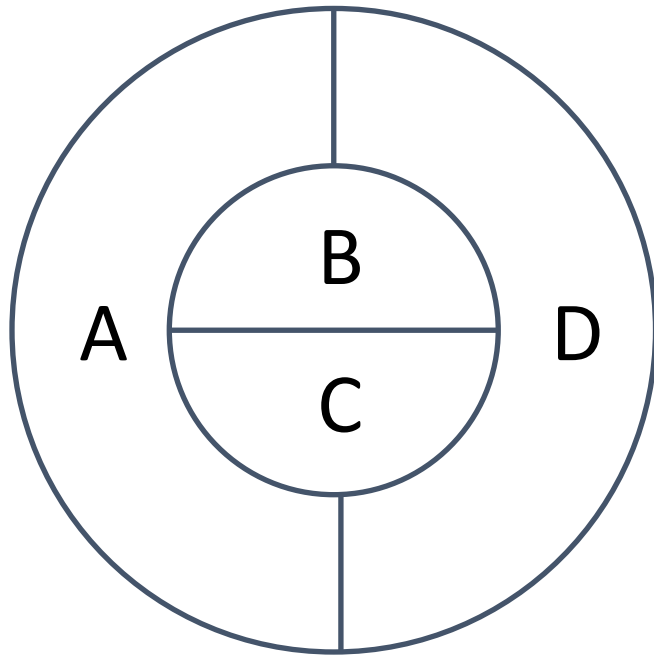


Early detection of failure: $O\{N^2\}$ checking

- **Arc consistency:**

- Check to make sure that every PAIR of variables (every “arc”) still has a pair-wise assignment that satisfies all constraints

Does arc consistency always detect the lack of a solution?



- There exist stronger notions of consistency (path consistency, k-consistency), but we won't worry about them

Summary

- CSPs are a special kind of search problem:
 - States defined by values of a fixed set of variables
 - Goal test defined by constraints on variable values
- **Backtracking** = depth-first search where successor states are generated by considering assignments to a single variable
 - **Variable ordering** (LRV, MCV) and **value selection** (LCV) heuristics can help significantly
 - **Forward checking** says: don't consider an assignment if it leaves any variable with no remaining possible values
 - **Arc consistency** says: don't consider an assignment if it leaves any **pair of variables** with no remaining mutually compatible pair of values
- Complexity of CSPs
 - NP-complete in general (exponential worst-case running time)
 - Typical run-time can be reduced substantially using polynomial-complexity forward-checking and arc-consistency heuristics