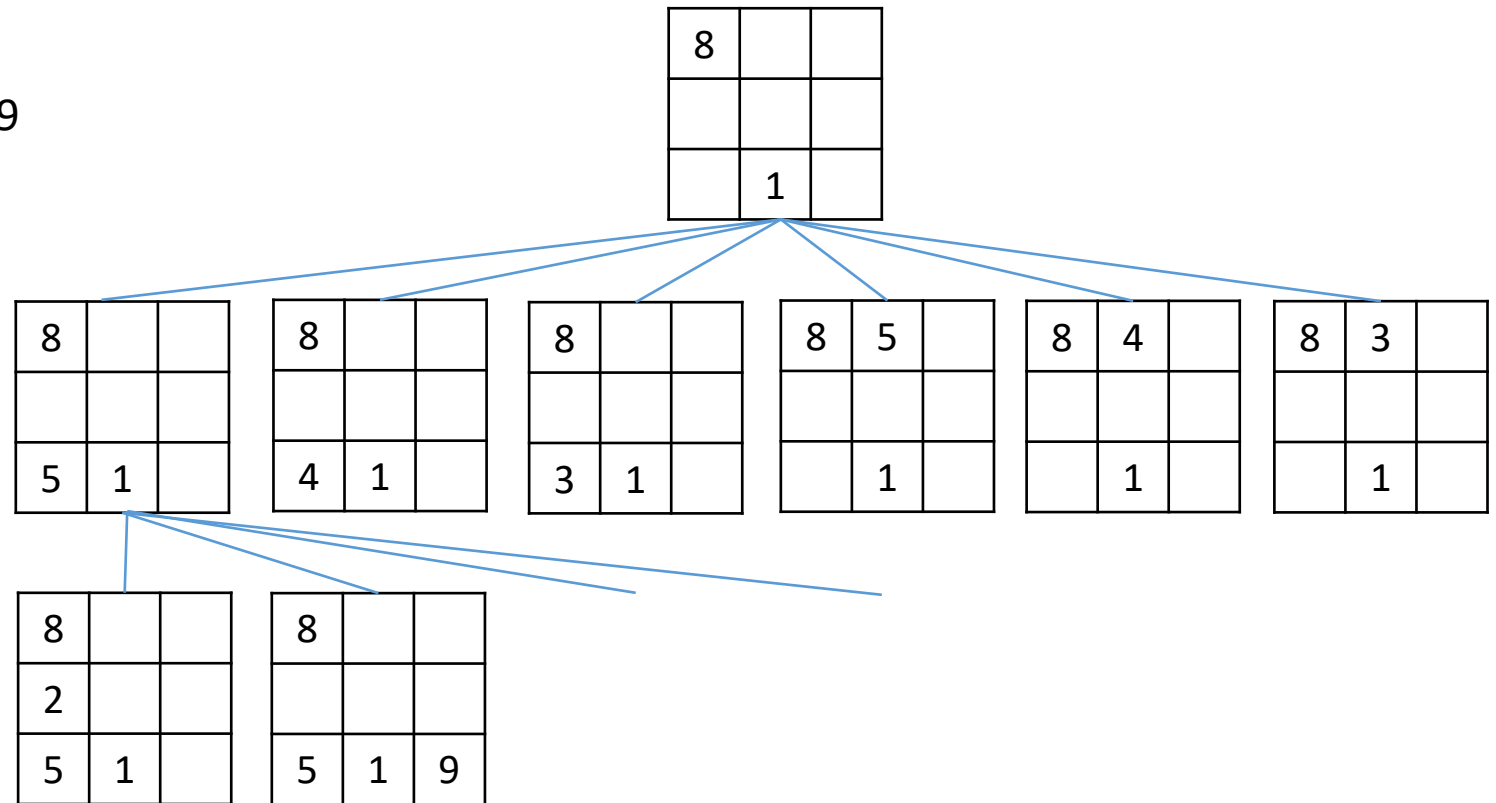


CS440/ECE 448, Lecture 6: Constraint Satisfaction Problems

Slides by Svetlana Lazebnik, 9/2016

Modified by Mark Hasegawa-Johnson, 1/2019

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?
- Examples: Map Coloring, N-Queens, Cryptarithmic, Classroom Assignment
- Formulation as a standard search
- Backtracking Search
- Heuristics to improve backtracking search
- Tree-structured CSPs
- NP-completeness of CSP in general; the SAT problem
- Local search, e.g., hill-climbing

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

Constraint satisfaction problems (CSPs)

- **Definition:**

- **State** is defined by **N variables** X_i with **values** from **domain** D_i
- **Goal test** is a set of **constraints** specifying allowable combinations of values for subsets of variables.
- **Solution** is a **complete, consistent** assignment
- True path costs are all N or ∞ .
Any path that works is exactly as good as any other.

- **How does this compare to the “generic” tree search formulation?**

- Far more states than usual.
BFS and A* are almost never computationally feasible.
- (Hopefully) many different paths to the same solution, therefore DFS might work.
- Structured state space allows us to use greedy search with really good heuristics.

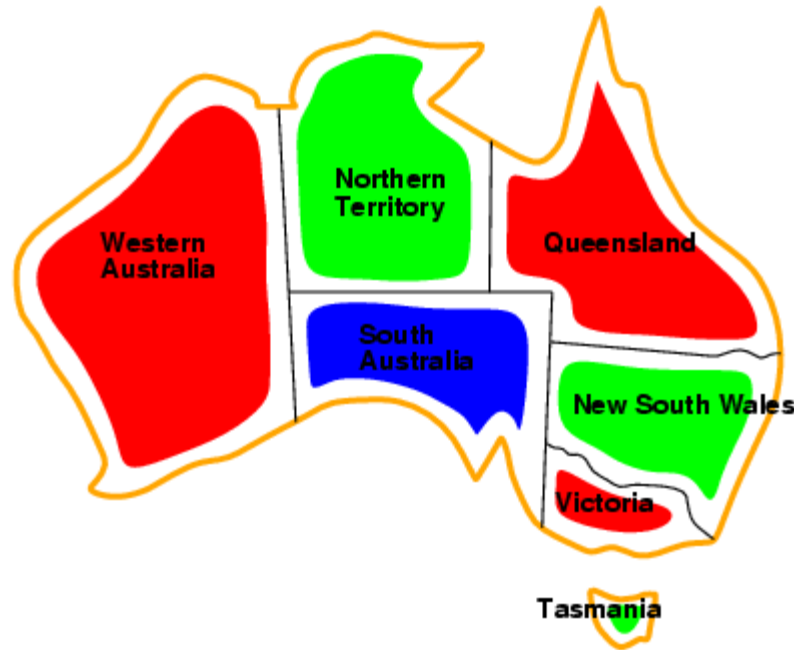
Examples

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: $(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}$

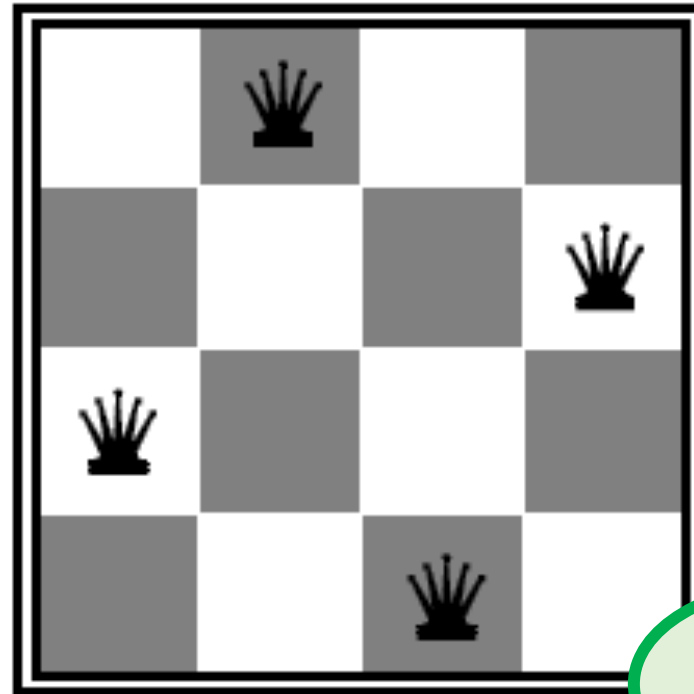
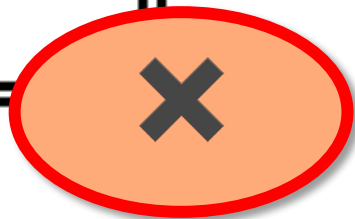
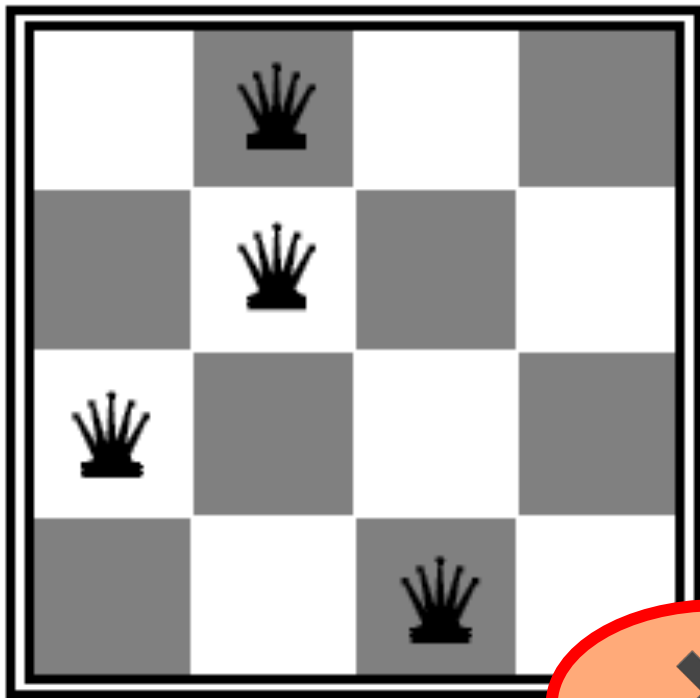
Example: Map Coloring



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

Example: n -queens problem

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal



Example: N-Queens

Variables: X_{ij} Domains: $\{0, 1\}$

Constraints:

N queens on board: $\sum_{i,j} X_{ij} = N$

Only 1 queen/row: $X_{ij} \wedge X_{ik} = 0$

Only 1 queen/column: $X_{ij} \wedge X_{kj} = 0$

Only 1 queen/diagonal: $X_{ij} \wedge X_{i+k,j+k} = 0$

Only 1 queen/diagonal: $X_{ij} \wedge X_{i+k,j-k} = 0$

Set

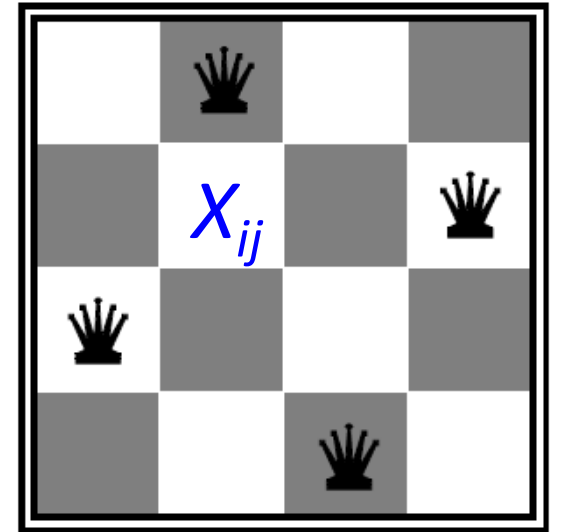
(??)

$(X_{ij}, X_{ik}) \in \{(0, 0), (0, 1), (1, 0)\}$

$(X_{ij}, X_{kj}) \in \{(0, 0), (0, 1), (1, 0)\}$

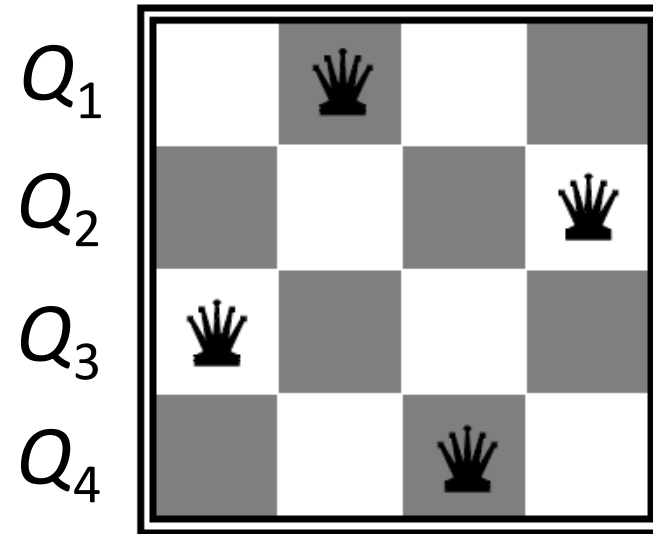
$(X_{ij}, X_{i+k,j+k}) \in \{(0, 0), (0, 1), (1, 0)\}$

$(X_{ij}, X_{i+k,j-k}) \in \{(0, 0), (0, 1), (1, 0)\}$



N-Queens: Alternative formulation

- **Variables:** Q_i
- **Domains:** $\{1, \dots, N\}$
- **Constraints:**
 $\forall i, j$ non-threatening (Q_i, Q_j)



Example: Crossword Puzzle

- **Variables:** 193 squares
- **Domains:** {a,b,...,z}

- **Constraints:**

Each row-segment is a word from the dictionary.

Each column-segment is a word from the dictionary.

1	2	3	4		5	6	7	8	9		10	11	12	13
14					15						16			
17				18						19				
20				21				22						
23			24				25							
		26			27				28		29	30	31	
32	33				34				35					
36				37							38			
39			40								41			
42						43				44				
			45		46				47				48	49
50	51	52						53				54		
55							56				57			
58					59						60			
61					62						63			

Example: Cryptarithmic

- **Variables:** T, W, O, F, U, R, X, Y
- **Domains:** {0, 1, 2, ..., 9}
- **Constraints:**
 - $O + O = R + 10 * Y$
 - $W + W + Y = U + 10 * X$
 - $T + T + X = 10 * F$
 - $\text{Alldiff}(T, W, O, F, U, R, X, Y)$
 - $T \neq 0, F \neq 0, X \neq 0$

$$\begin{array}{r} X Y \\ T W O \\ + T W O \\ \hline F O U R \end{array}$$

Real-world CSPs

- Assignment problems
 - e.g., who teaches what class
- Timetable problems
 - e.g., which class is offered when and where?
- Transportation scheduling
- Factory scheduling
- More examples of CSPs: <http://www.csplib.org/>

Formulation as a standard
search

Standard search formulation (incremental)

- **States:** Variables and values assigned so far
- **Initial state:** The empty assignment
- **Action:** Choose any unassigned variable and assign to it a value that does not violate any constraints
Fail if no legal assignments
- **Goal test:** The current assignment is *complete* and *satisfies all constraints*

Standard search formulation (incremental)

What is the depth of any solution (assuming N variables)?

Answer: N (this is good)

Given that there are D possible values for any variable, how many paths are there in the search tree?

Answer: $N! D^N$ (this is bad)

All paths have the same depth, so DFS and BFS have the same complexity ($O(N! D^N)$)

Other reasons to use DFS:

- There are usually many paths to the solution (at least $N!$)
- Often, if a path fails, we can detect this early
- Today's goal: develop heuristics to reduce the branching factor

Backtracking search

Backtracking search

In CSP's, variable assignments are **commutative**

For example, $[WA = \text{red then } NT = \text{green}]$
is the same as $[NT = \text{green then } WA = \text{red}]$

We only need to consider **assignments to a single variable at each level** (i.e., we fix the order of assignments)

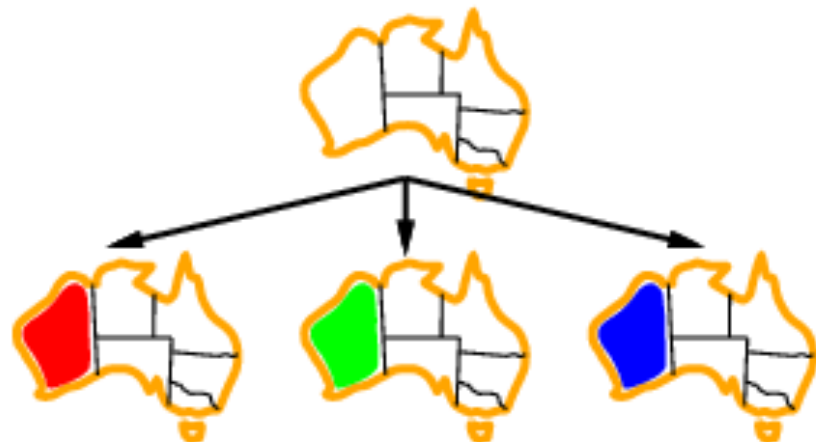
Then there are only D^N paths. We have eliminated the $N!$ redundancy by arbitrarily choosing an order in which to assign variables.

- Depth-first search for CSPs with single-variable assignments is called **backtracking search**

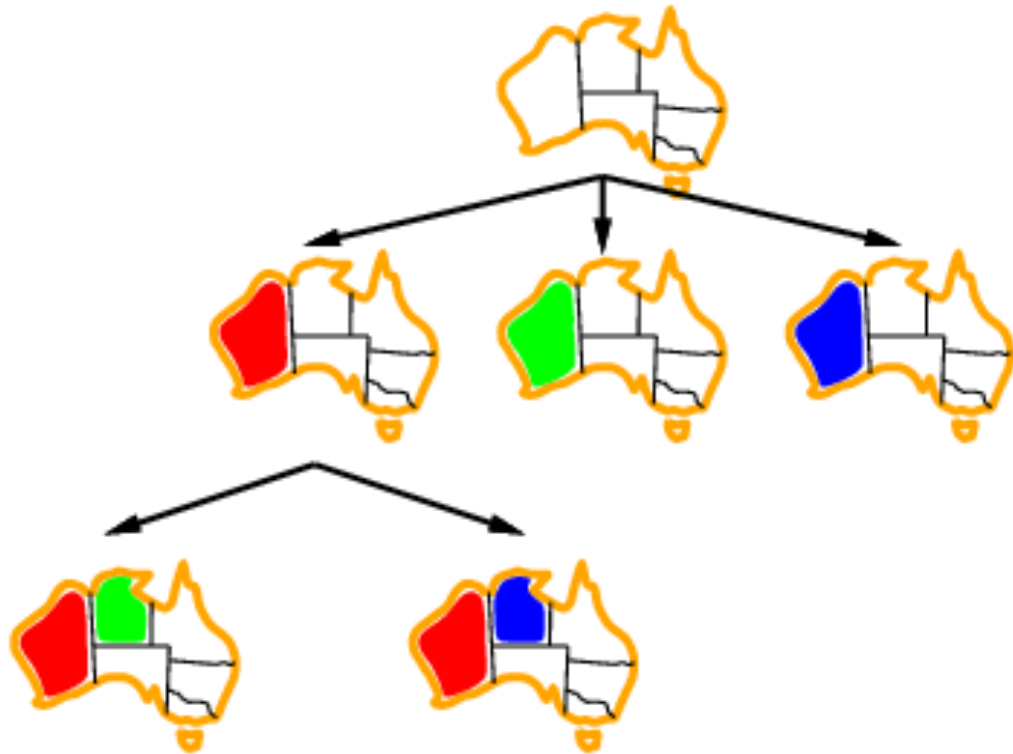
Example



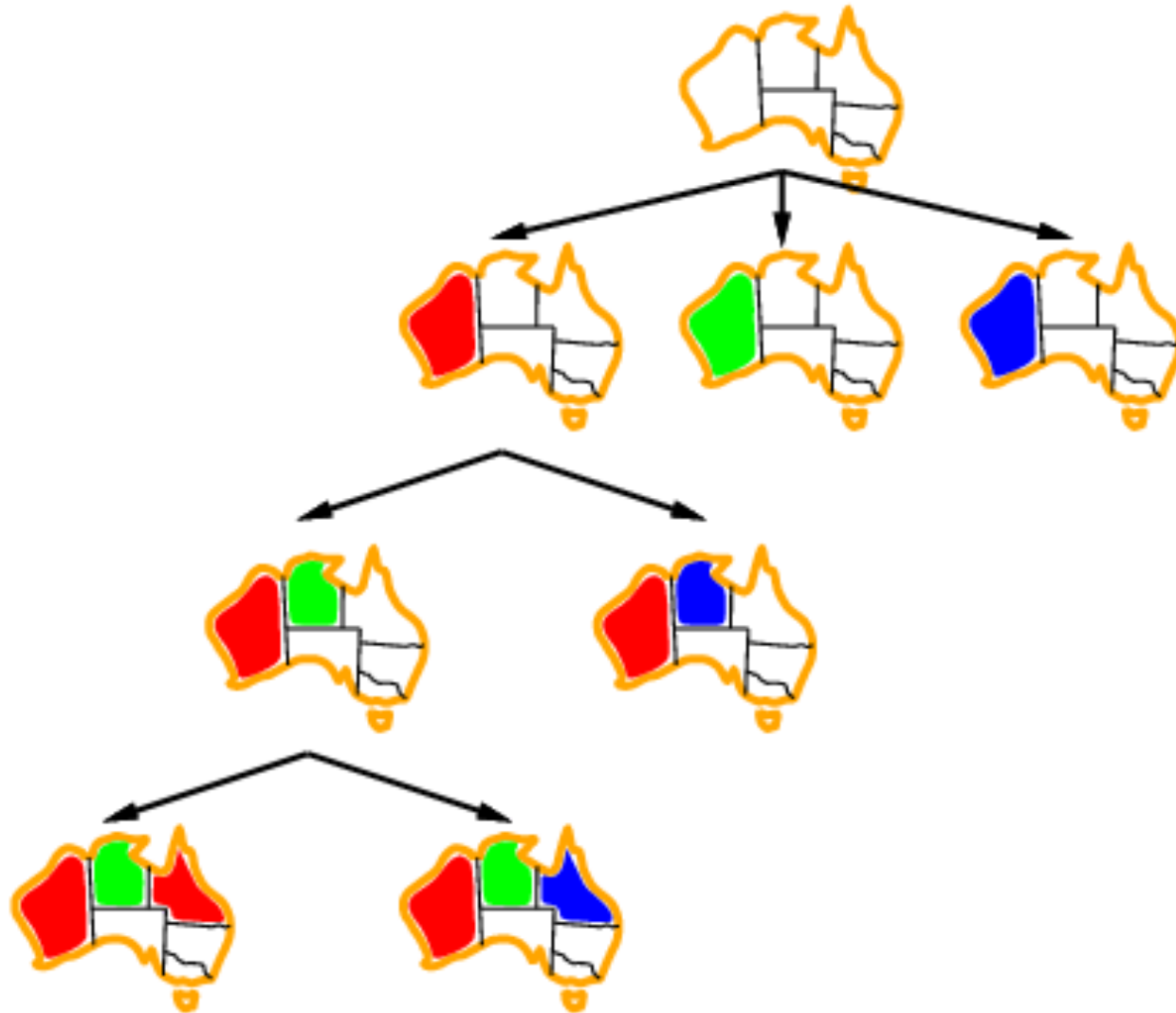
Example



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?

Heuristics for making
backtracking search
more efficient

Heuristics for making backtracking search more efficient

Still DFS, but we use heuristics to decide which child to expand first. You could call it GDFS...

- Heuristics that choose **the next variable** to assign:
 - Least Remaining Values (LRV)
 - Most Constraining Variable (MCV)
- Heuristic that chooses **a value** for that variable:
 - Least Constraining Assignment (LCA)
- **Early detection of failure:**
 - Forward Checking
 - Arc Consistency

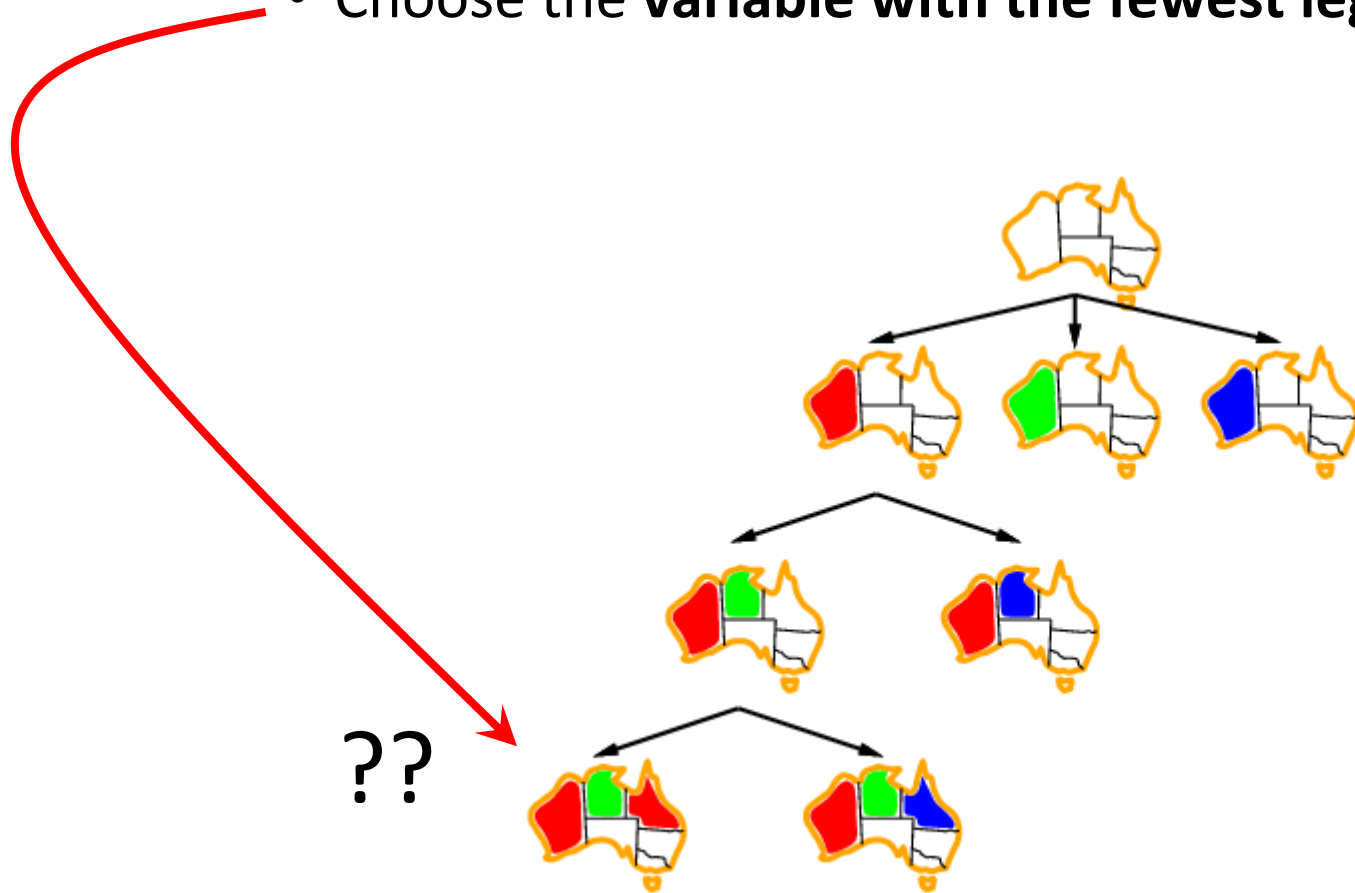
Which variable should be assigned next?

- **Least Remaining Values (LRV) Heuristic:**
 - Choose the **variable with the fewest legal values**

Which variable should be assigned next?

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

- Choose the **variable with the fewest legal values**



Which variable should be assigned next?

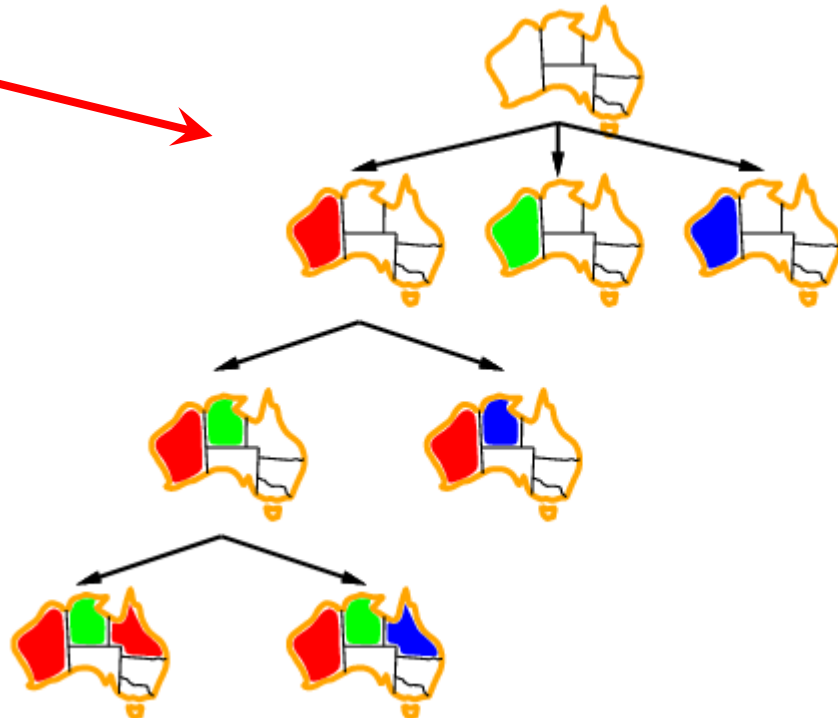
- **Most Constraining Variable (MCV) Heuristic:**
 - Choose the **variable that imposes the most constraints on the remaining variables**
 - Tie-breaker among variables that have equal numbers of LRV

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

??

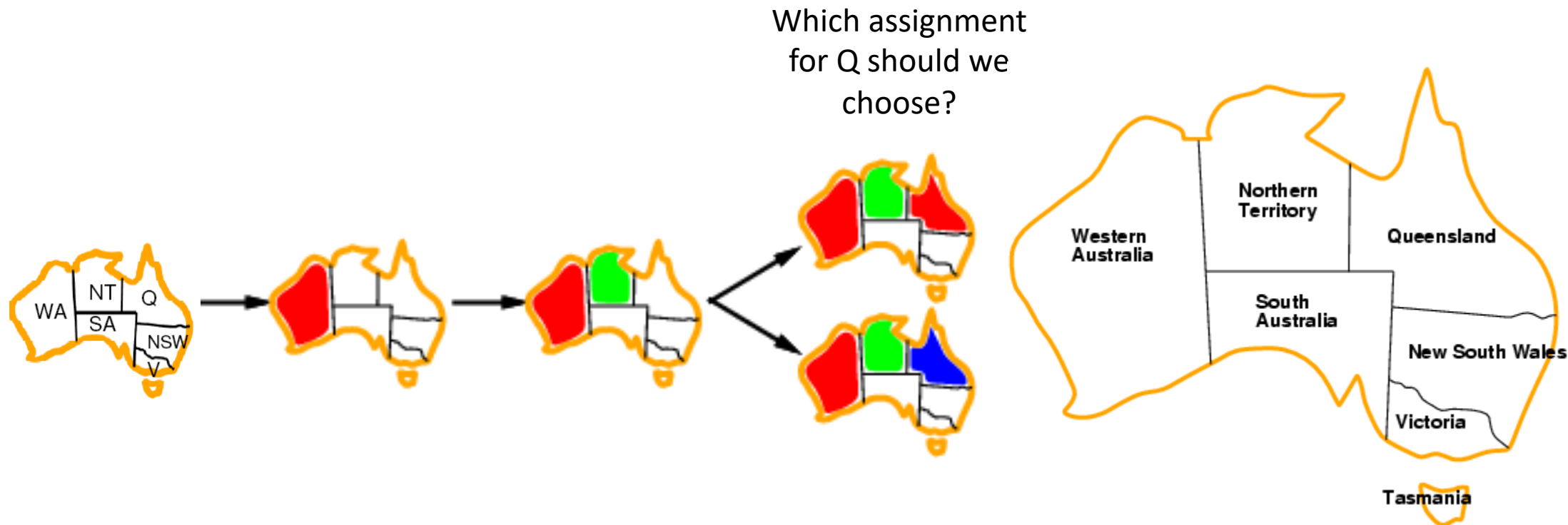


Given a variable, in which order should its values be tried?

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

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

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



Early detection of failure

```
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
```



Apply *inference* to reduce the space of possible assignments and detect failure early

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

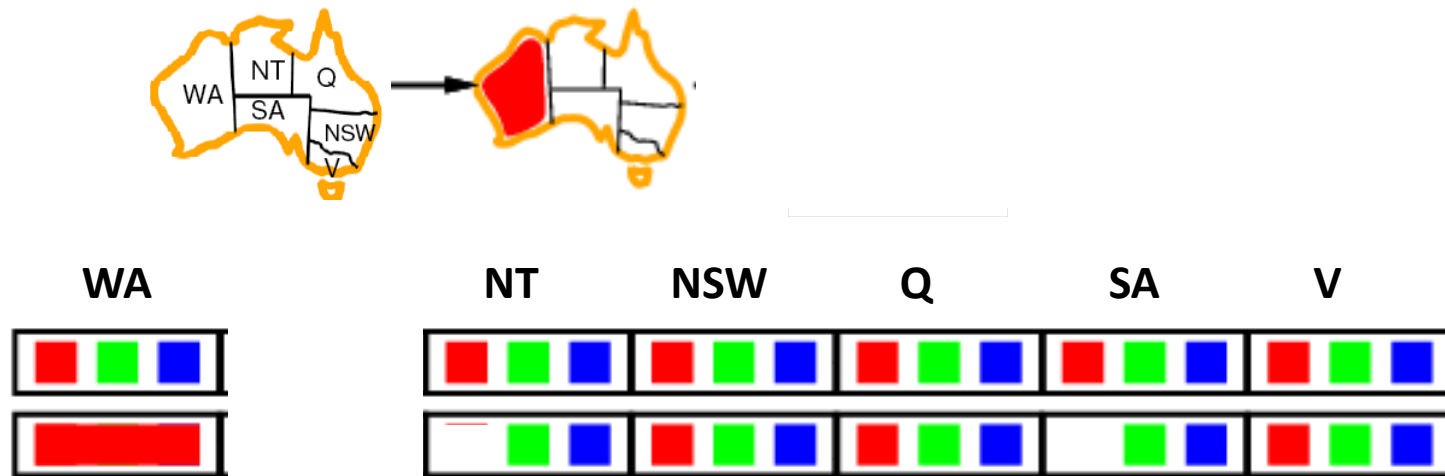
- Keep track of remaining legal values for unassigned variables
- Terminate search when **any variable has no legal values**



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

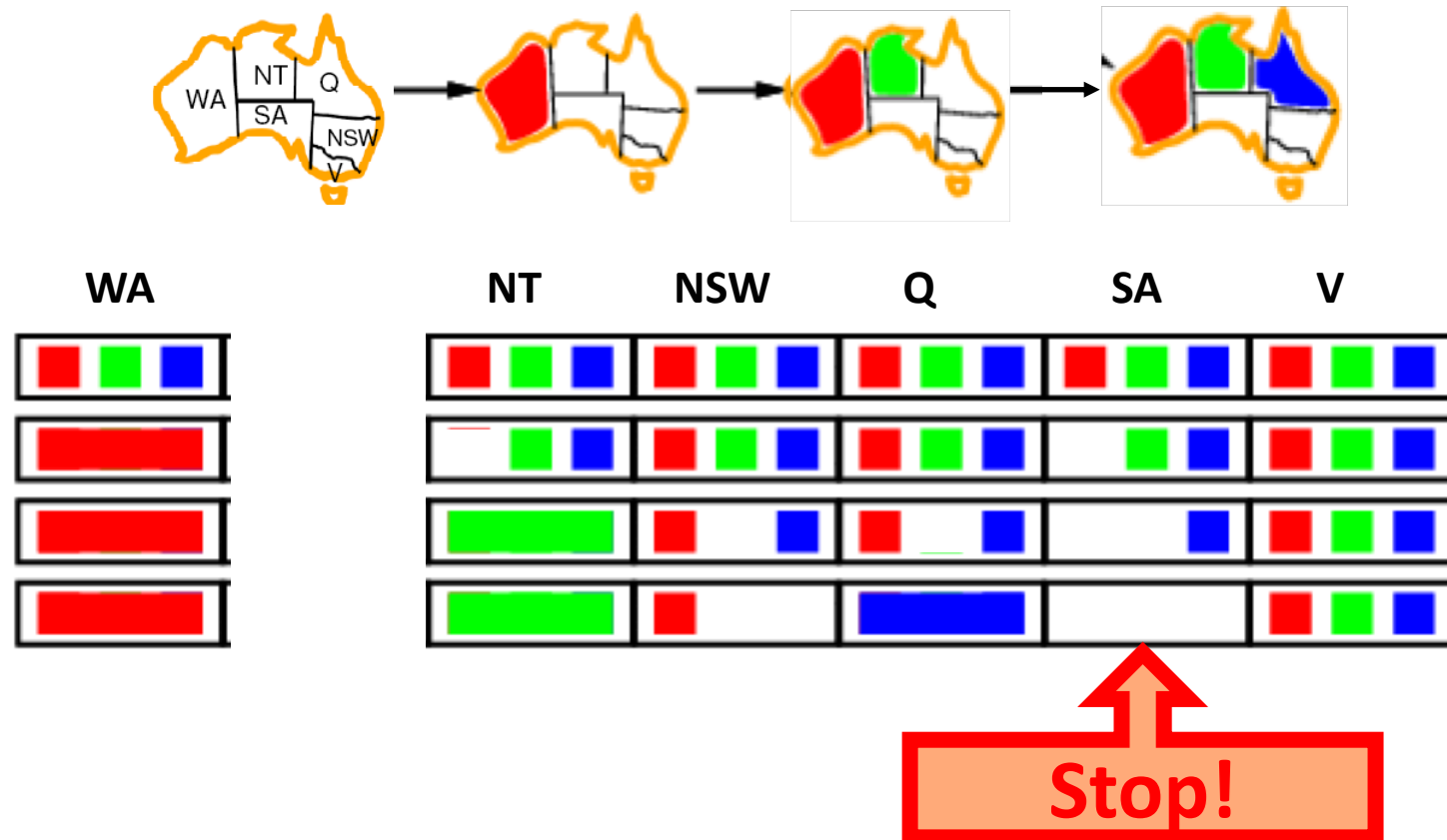
Forward checking

- 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

- **Constraint propagation:**
 - Check to make sure that **every PAIR of variables still has a pair-wise assignment that satisfies all constraints**

Early detection of failure: $O(N^2)$ checking

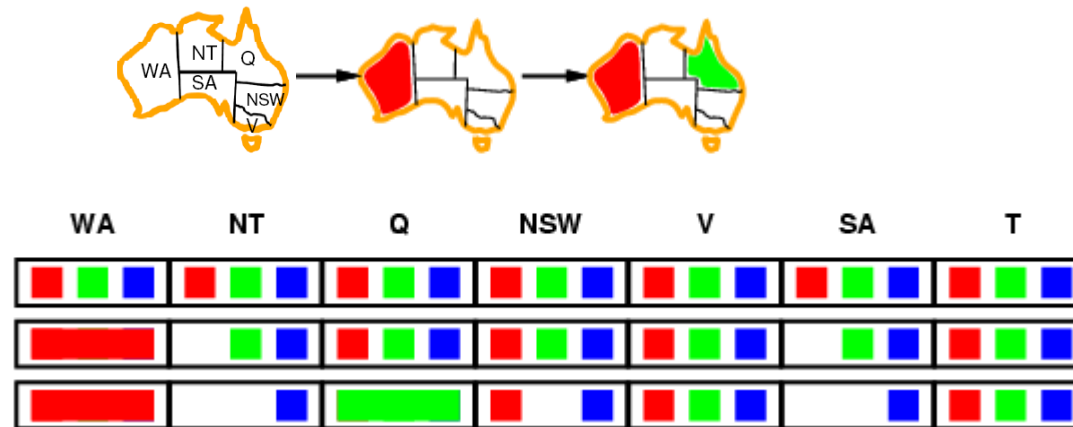


Apply *inference* to reduce the space of possible assignments and detect failure early

(Reminder: there are only three colors, RGB...)

Constraint propagation

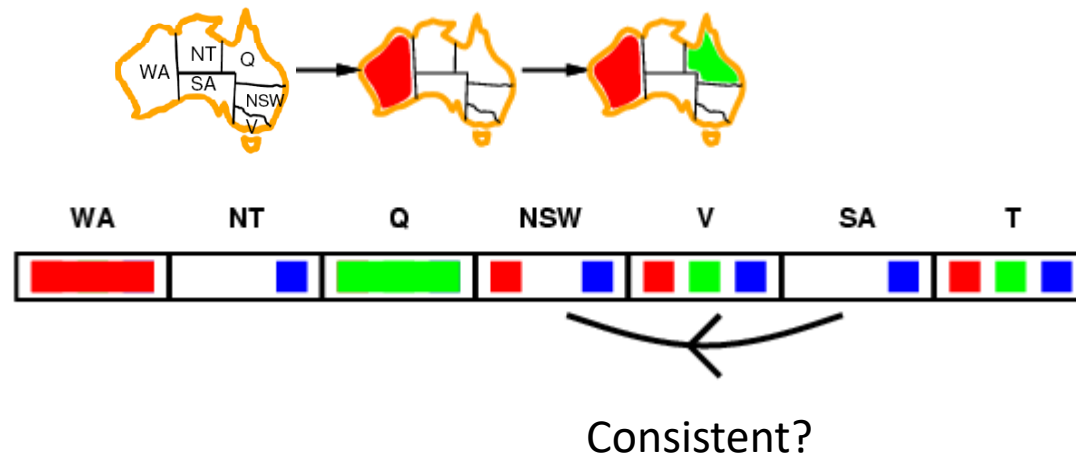
Forward checking propagates information from assigned to unassigned variables, but **doesn't provide early detection for all failures**



- NT and SA cannot both be blue!
- **Constraint propagation** repeatedly enforces constraints *locally*

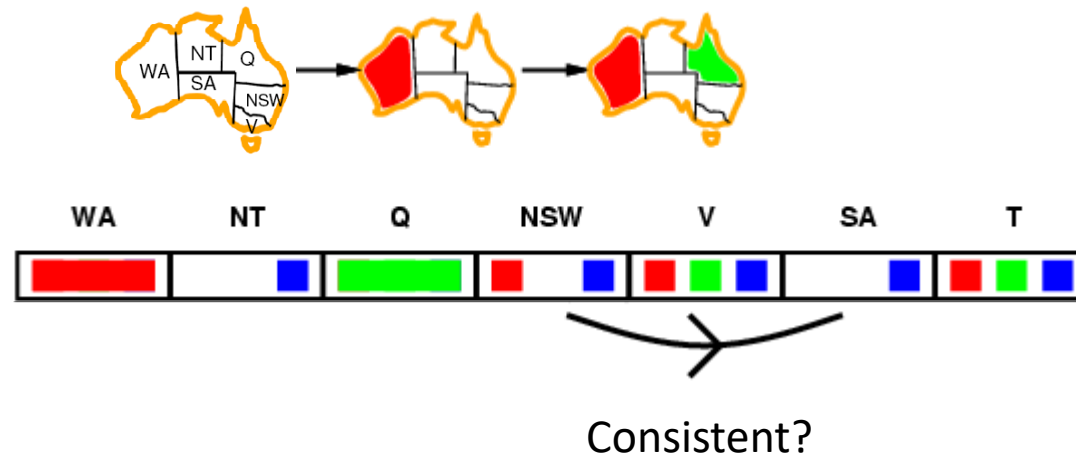
Constraint propagation algorithm: Arc consistency

- Simplest form of propagation makes each pair of variables **consistent**:
 - $X \rightarrow Y$ is consistent iff for **every** value of X there is **some** allowed value of Y



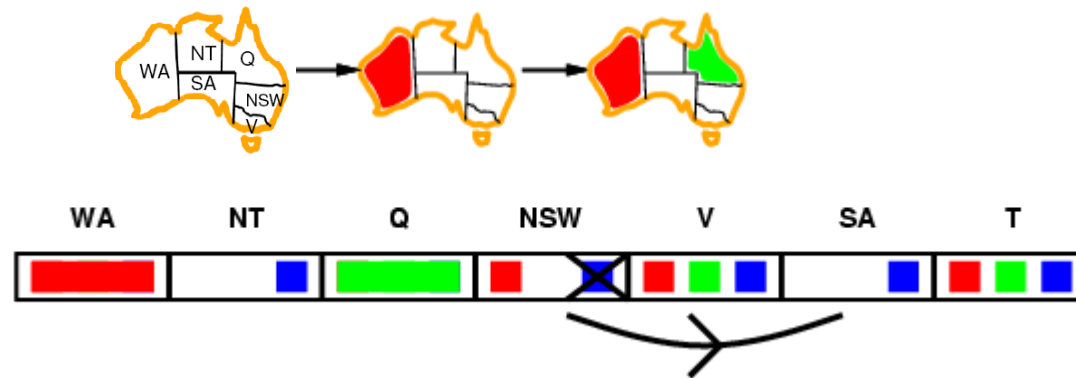
Constraint propagation algorithm: Arc consistency

- Simplest form of propagation makes each *pair of variables* **consistent**:
 - $X \rightarrow Y$ is consistent iff for **every** value of X there is **some** allowed value of Y



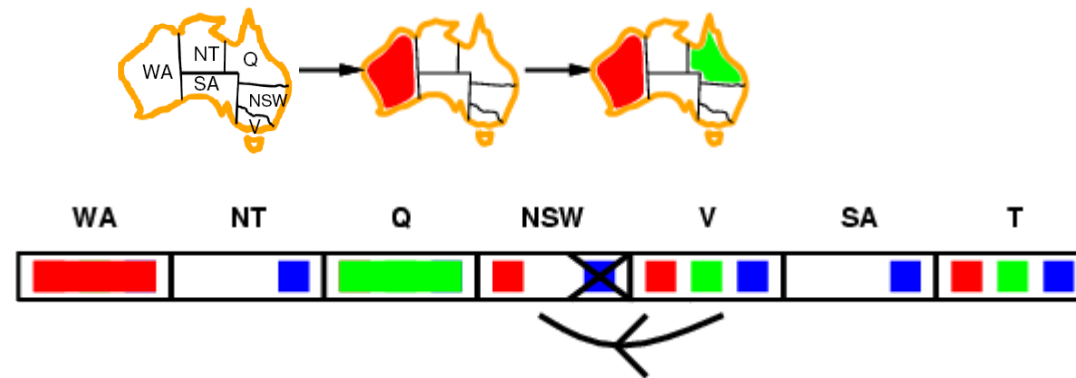
Constraint propagation algorithm: Arc consistency

- Simplest form of propagation makes each *pair of variables* **consistent**:
 - $X \rightarrow Y$ is consistent iff for **every** value of X there is **some** allowed value of Y
 - When checking $X \rightarrow Y$, throw out any values of X for which there isn't an allowed value of Y



Constraint propagation algorithm: Arc consistency

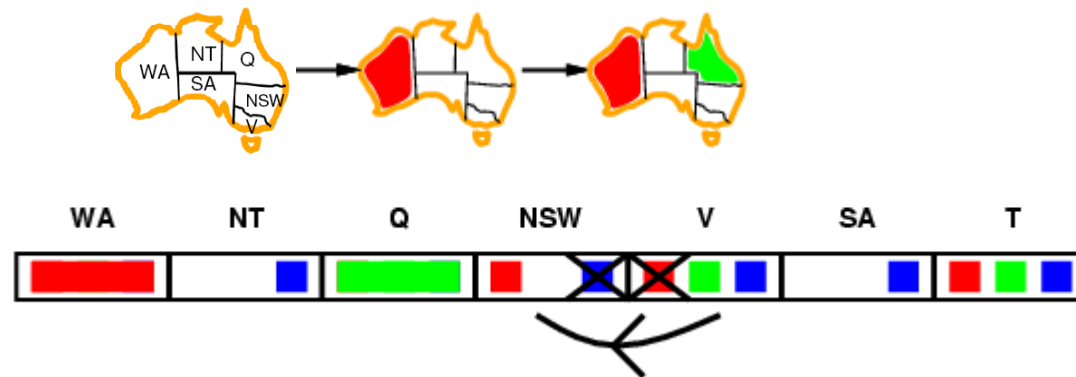
- Simplest form of propagation makes each *pair of variables* **consistent**:
 - $X \rightarrow Y$ is consistent iff for **every** value of X there is **some** allowed value of Y
 - When checking $X \rightarrow Y$, throw out any values of X for which there isn't an allowed value of Y



- If X loses a value, all pairs $Z \rightarrow X$ need to be rechecked

Constraint propagation algorithm: Arc consistency

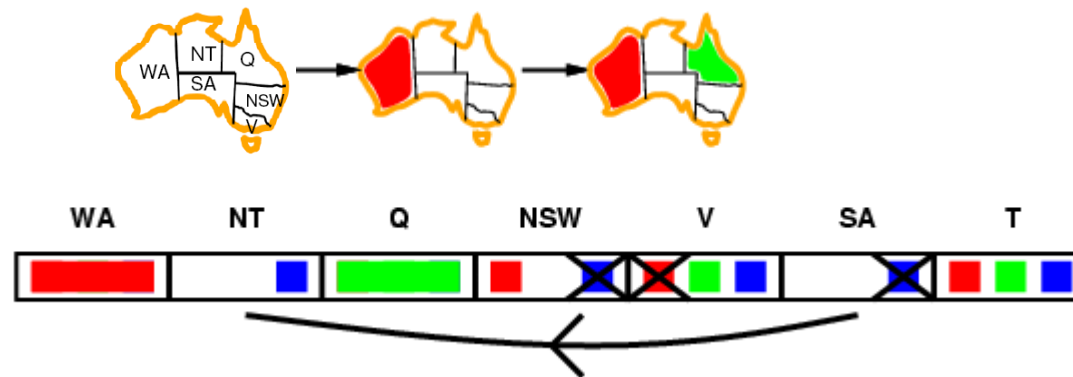
- Simplest form of propagation makes each *pair of variables* **consistent**:
 - $X \rightarrow Y$ is consistent iff for **every** value of X there is **some** allowed value of Y
 - When checking $X \rightarrow Y$, throw out any values of X for which there isn't an allowed value of Y



- If X loses a value, all pairs $Z \rightarrow X$ need to be rechecked

Constraint propagation algorithm: Arc consistency

- Simplest form of propagation makes each *pair of variables* **consistent**:
 - $X \rightarrow Y$ is consistent iff for **every** value of X there is **some** allowed value of Y
 - When checking $X \rightarrow Y$, throw out any values of X for which there isn't an allowed value of Y



- Arc consistency detects failure earlier than forward checking
- Can be run before or after each assignment

Arc consistency algorithm AC-3

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains

inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \dots, X_n\}$

local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

while *queue* is not empty

$(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\textit{queue})$

if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**

for each X_k **in** NEIGHBORS[X_i] **do**

 add (X_k, X_i) to *queue*

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff succeeds

removed \leftarrow false

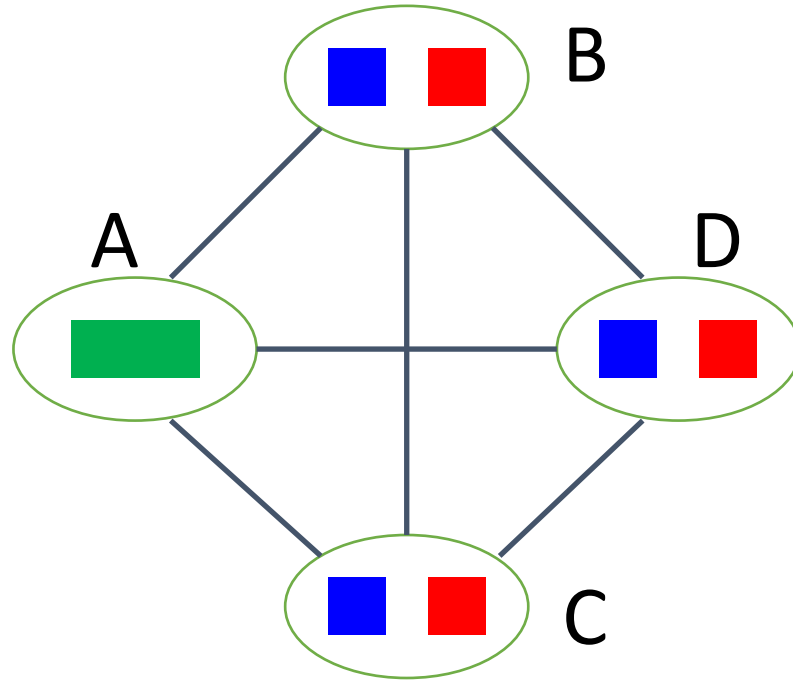
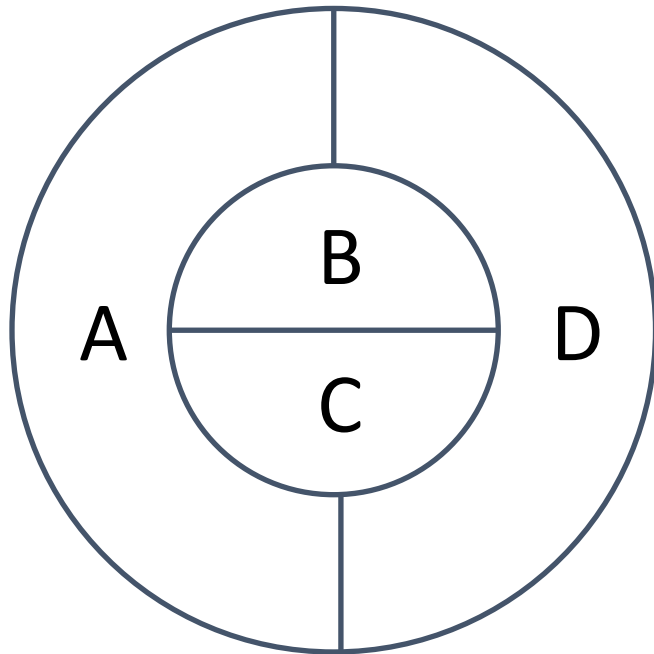
for each x **in** DOMAIN[X_i]

if no value y in DOMAIN[X_j] allows (x, y) to satisfy the constraint $X_i \leftrightarrow X_j$

then delete x from DOMAIN[X_i]; *removed* \leftarrow true

return *removed*

Does arc consistency always detect the lack of a solution?

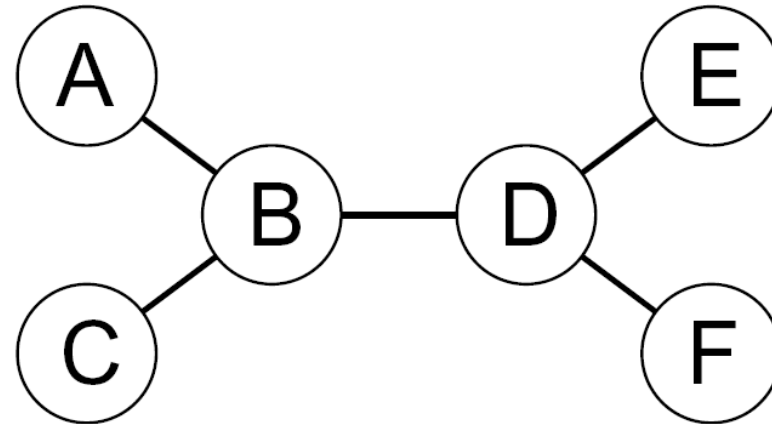


- There exist stronger notions of consistency (path consistency, k-consistency), but we won't worry about them

Tree-structured CSPs

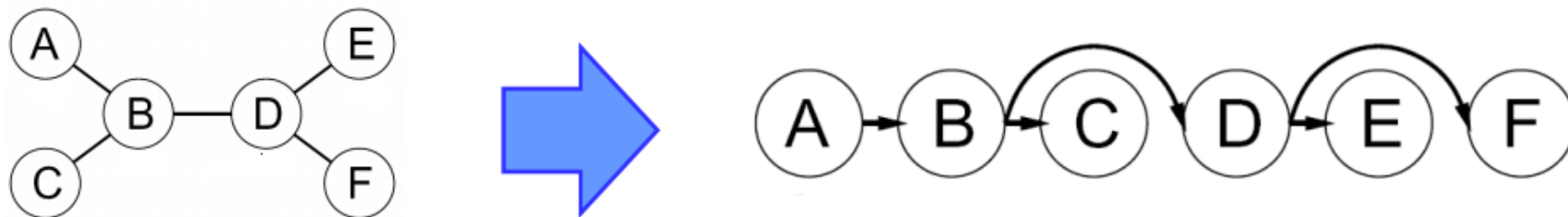
Tree-structured CSPs

- Certain kinds of CSPs can be solved without resorting to backtracking search!
- *Tree-structured CSP*: constraint graph does not have any loops



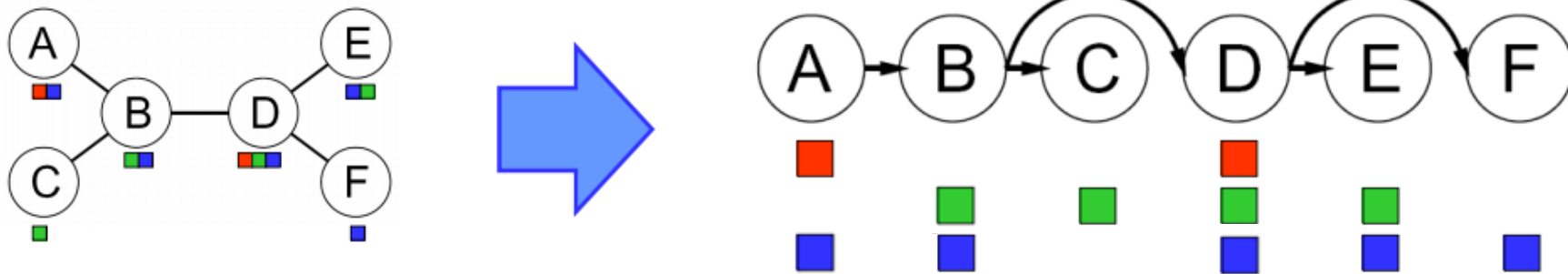
Algorithm for tree-structured CSPs

- Choose one variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering



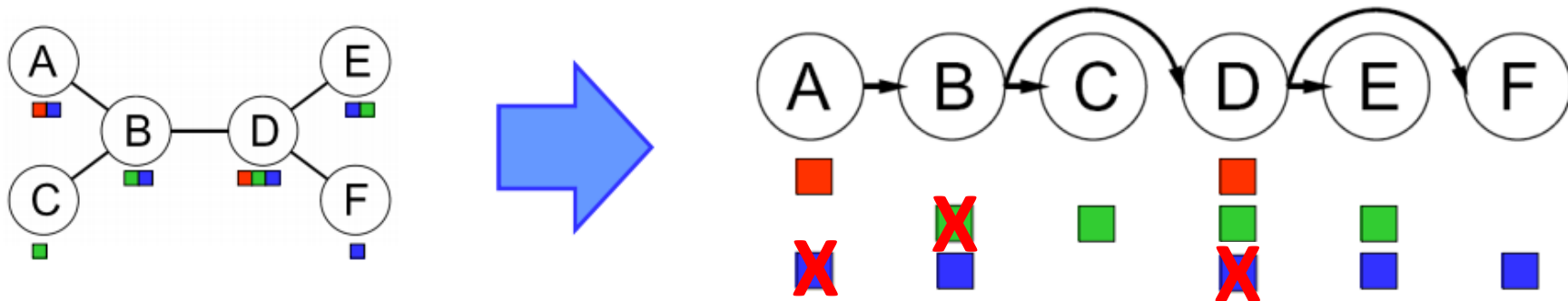
Algorithm for tree-structured CSPs

- Choose one variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering
- Create a graph listing all of the values that can be assigned to each variable.



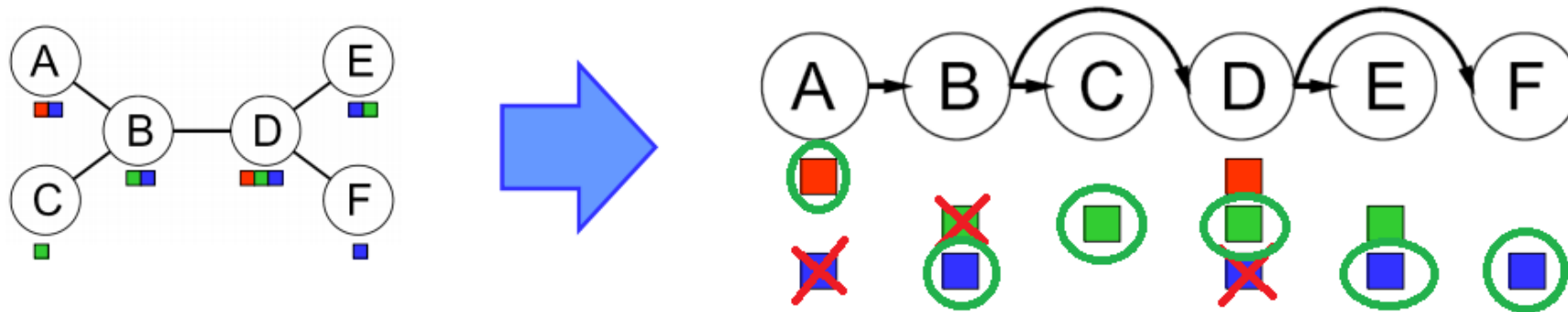
Algorithm for tree-structured CSPs

- Choose one variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering
- Create a graph listing all of the values that can be assigned to each variable.
- **BACKWARD ARC CONSISTENCY:** check arc consistency starting from the rightmost node and going backwards



Algorithm for tree-structured CSPs

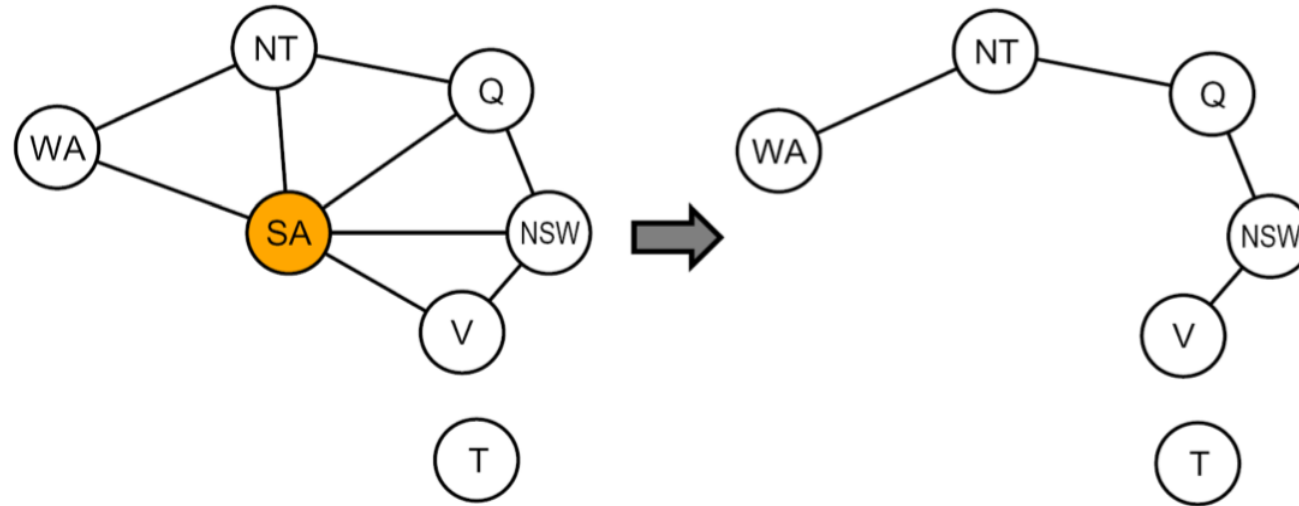
- Choose one variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering
- Create a graph listing all of the values that can be assigned to each variable.
- BACKWARD ARC CONSISTENCY: check arc consistency starting from the rightmost node and going backwards
- FORWARD ASSIGNMENT PHASE: select an element from the domain of each variable going left to right. We are guaranteed that there will be a valid assignment because each arc is consistent



Algorithm for tree-structured CSPs

- If N is the number of variables and D is the domain size, what is the running time of this algorithm?
 - $O(ND^2)$: we have to check arc consistency once for every node in the graph (every node has one parent), which involves looking at pairs of domain values

Nearly tree-structured CSPs



- **Cutset conditioning:**
find a subset of variables whose removal makes the graph a tree,
instantiate that set in all possible ways,
prune the domains of the remaining variables
and try to solve the resulting tree-structured CSP
- Cutset size c gives runtime $O(D^c (N - c)D^2)$

NP-Completeness and the SAT Problem

Algorithm for tree-structured CSPs

- Running time is $O(ND^2)$
(N is the number of variables, D is the domain size)
 - We have to check arc consistency once for every node in the graph (every node has one parent), which involves looking at pairs of domain values
- What about backtracking search for general CSPs?
 - Worst case $O(D^M)$
- Can we do better?

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?

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

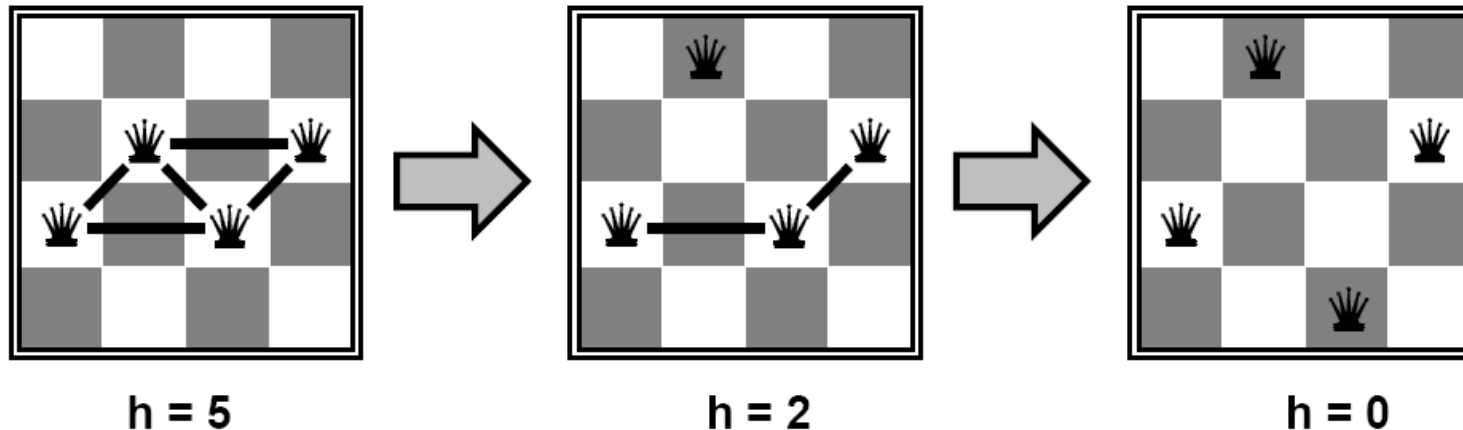
SAT is NP-complete

- **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 a) in NP, and b) 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

Local search,
e.g., hill climbing

Local search for CSPs

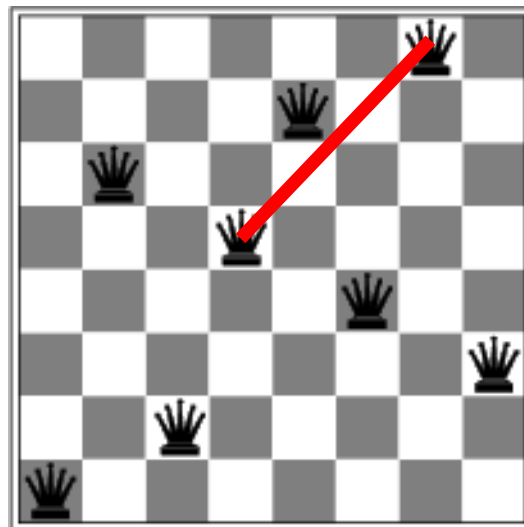
- Start with “**complete**” states (all variables assigned)
- Allow states with unsatisfied constraints (those have **conflicts**)
- Attempt to **improve** states by reassigning variable values
- **Hill-climbing search:**
 - In each iteration, randomly select any conflicted variable and choose **the value that violates the fewest constraints**
 - I.e., attempt to **greedily minimize the total number of violated constraints**



h = number of conflicts

Local search for CSPs

- Start with “**complete**” states (all variables assigned)
- Allow states with unsatisfied constraints (those have **conflicts**)
- Attempt to **improve** states by reassigning variable values
- **Hill-climbing search:**
 - In each iteration, randomly select any conflicted variable and choose **the value that violates the fewest constraints**
 - I.e., attempt to **greedily minimize the total number of violated constraints**
 - **Problem: Local minima**

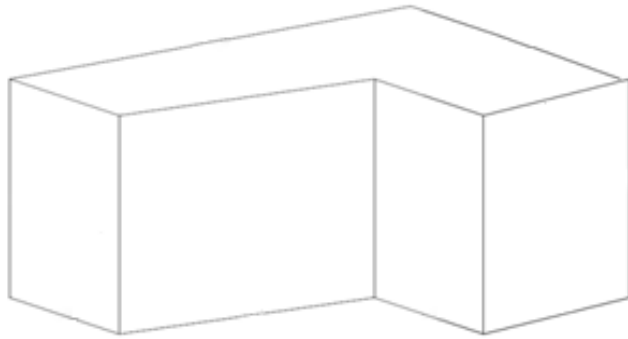


$h = 1$

Applications that look a lot
like intelligence...

CSP in computer vision: Line drawing interpretation

An example polyhedron:



Variables: edges

Domains:

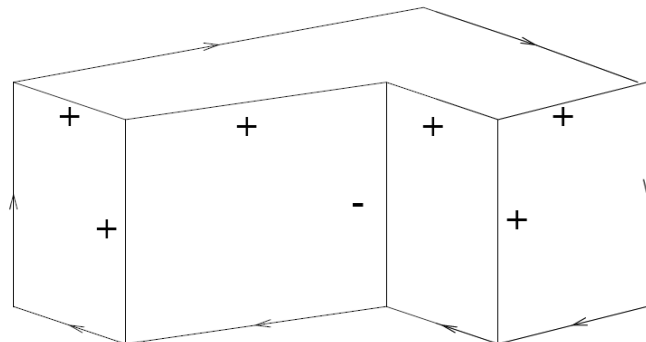
+ (convex),

- (concave),

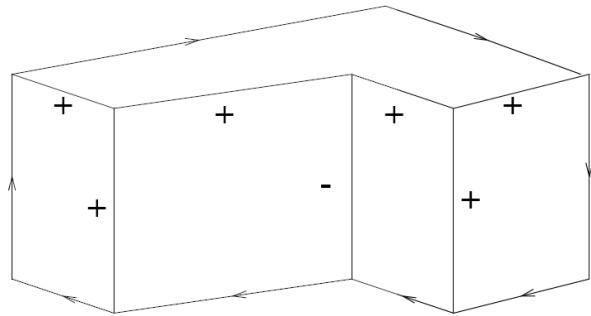
→ (occluding),

← (occluding)

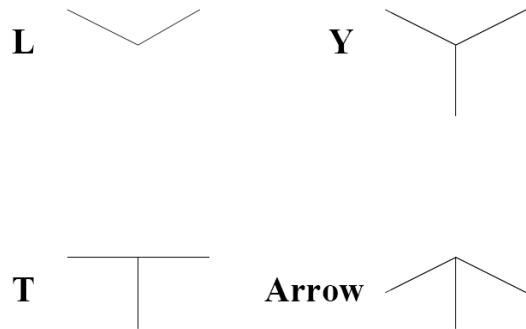
Desired output:



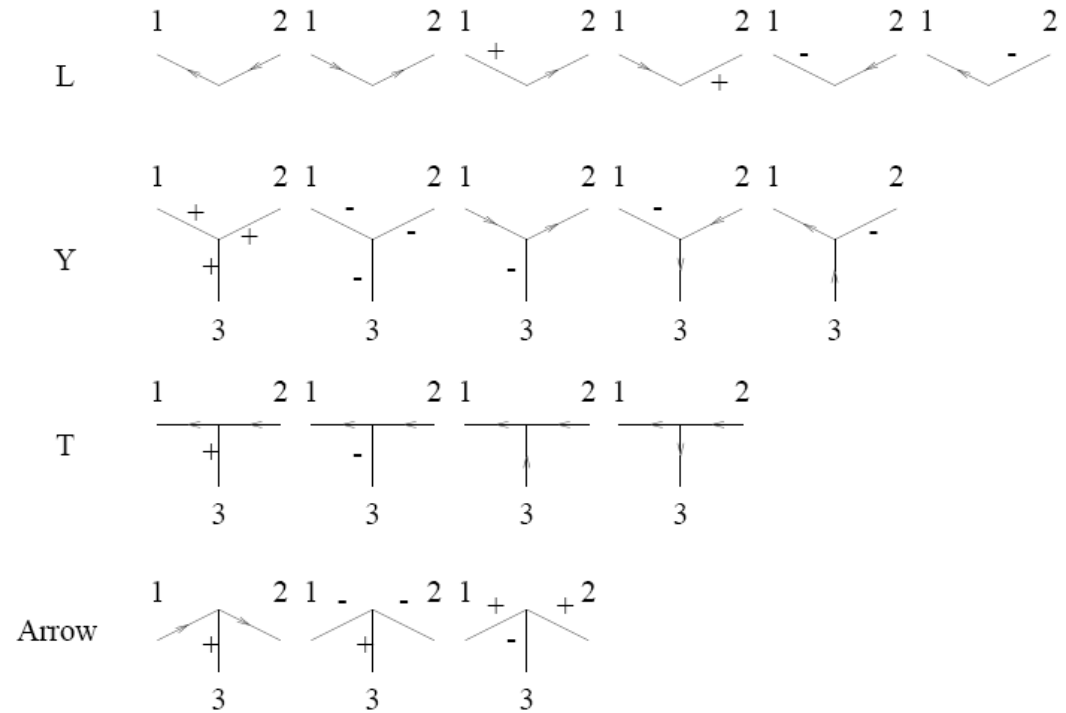
CSP in computer vision: Line drawing interpretation



Four vertex types:



Constraints imposed by each vertex type:



CSP in computer vision: 4D Cities

1. When was each photograph taken?
2. When did each building first appear?
3. When was each building removed?

Set of Photographs:



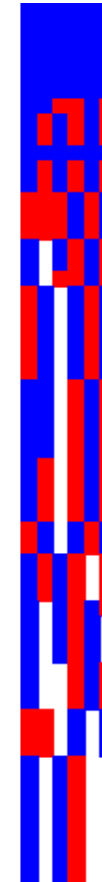
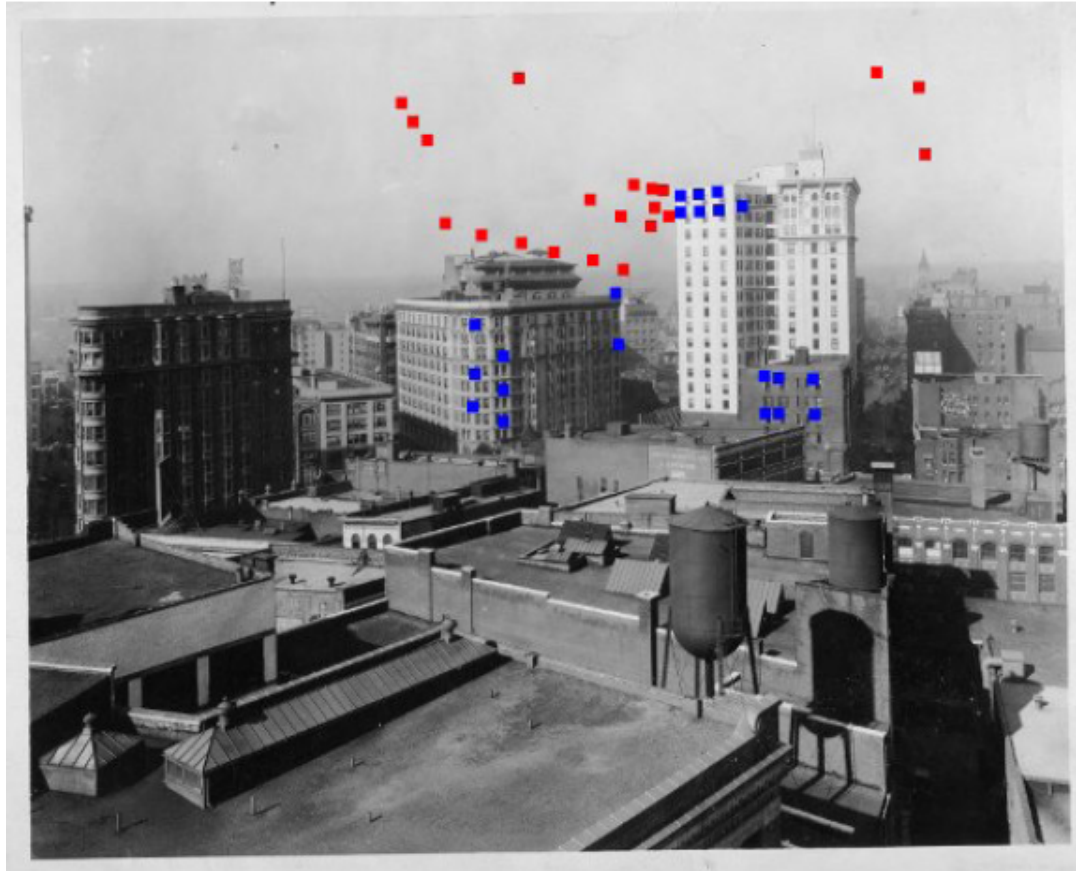
**Set of Objects:
Buildings**

G. Schindler, F. Dellaert, and S.B. Kang, [Inferring Temporal Order of Images From 3D Structure](#), IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), 2007.

<http://www.cc.gatech.edu/~phlosoft/>

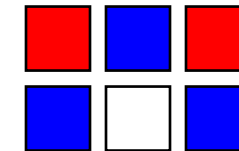
CSP in computer vision: 4D Cities

■ observed ■ missing □ occluded

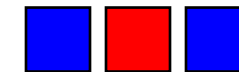


Columns: images
Rows: points

Satisfies constraints:



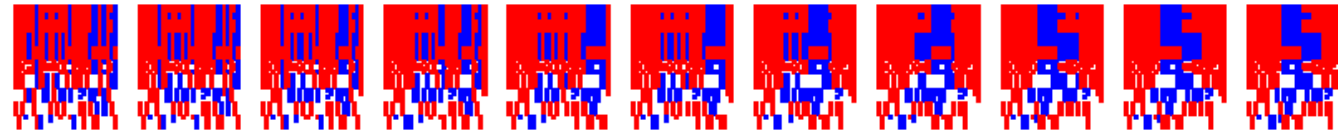
Violates constraints:



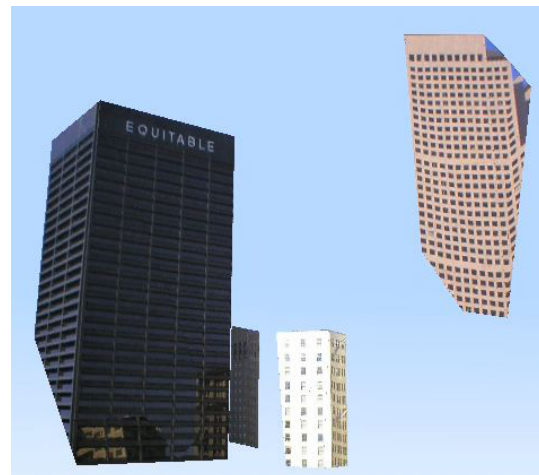
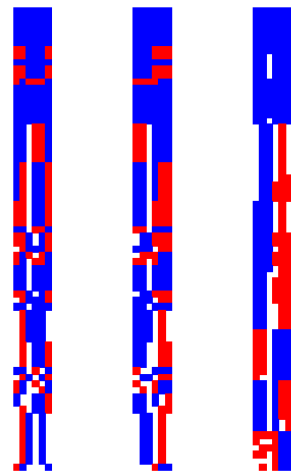
- Goal: reorder images (columns) to have as few violations as possible

CSP in computer vision: 4D Cities

- **Goal:** reorder images (columns) to have as few violations as possible
- **Local search:** start with random ordering of columns, swap columns or groups of columns to reduce the number of conflicts



- Can also reorder the rows to group together points that appear and disappear at the same time – that gives you buildings



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** and **value selection** heuristics can help significantly
 - **Forward checking** prevents assignments that guarantee later failure
 - **Constraint propagation** (e.g., arc consistency) does additional work to constrain values and detect inconsistencies
- Complexity of CSPs
 - NP-complete in general (exponential worst-case running time)
 - Efficient solutions possible for special cases (e.g., tree-structured CSPs)
- Alternatives to backtracking search: local search