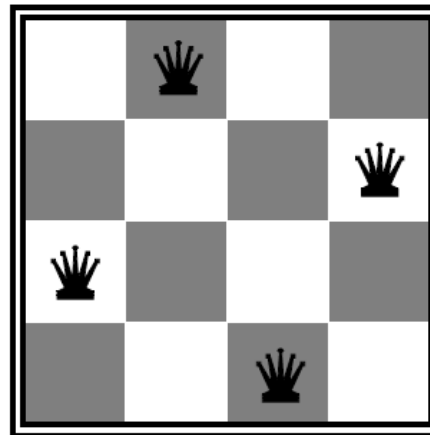
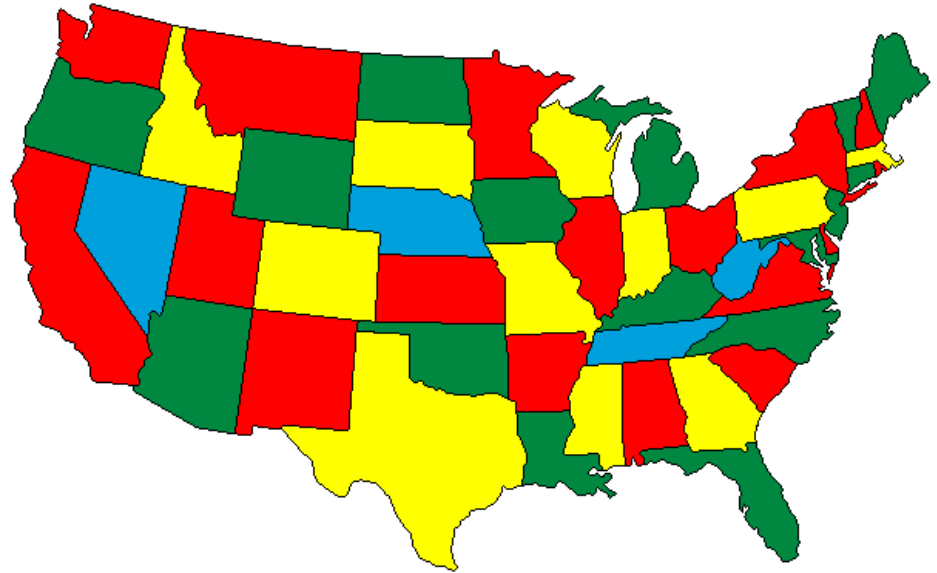


CS 5/7320 Artificial Intelligence

Constraint Satisfaction Problems

Slides by Michael Hahsler
based on Slides by
Svetlana Lazepnik



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

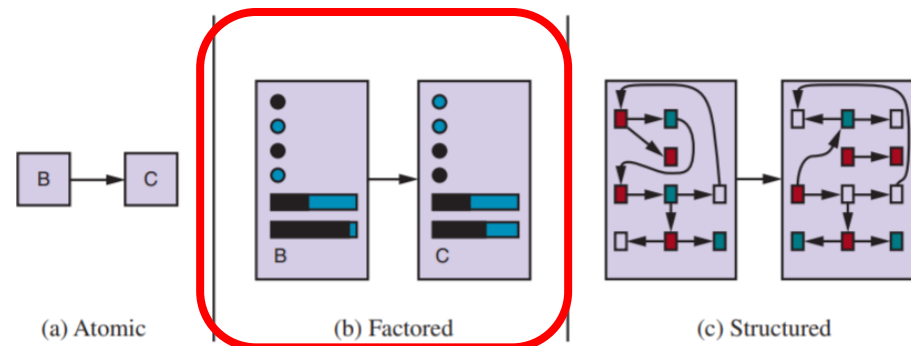
Constraint satisfaction problems (CSPs)

Definition:

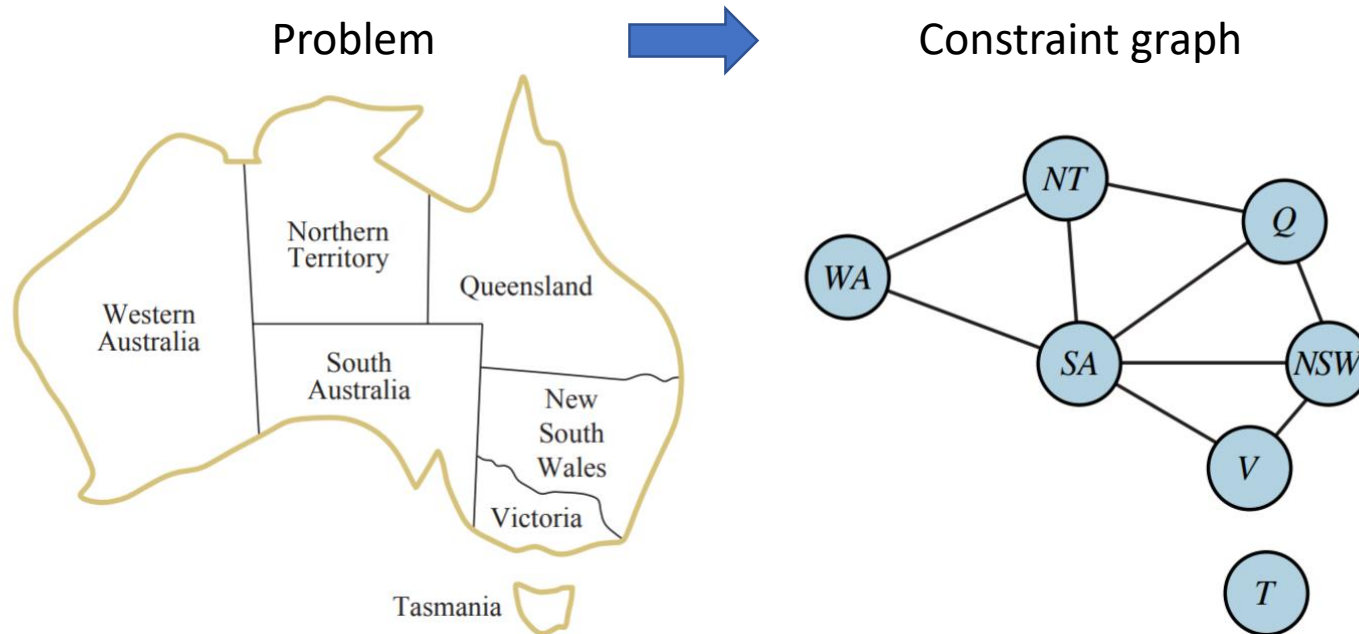
- **State** is defined by a set of **variables** X_i
 - Each variable can be **unassigned** (partial solution) or have a **value** from **domain** D_i .
- **Constraints** are a set of rules specifying allowable combinations of values for subsets of variables (e.g., $X_1 \neq X_7$ or $X_2 > X_9 + 3$)
- **Solution**: a state that is a
 - a) **Consistent assignment**: satisfies all constraints
 - b) **Complete assignment**: assigns value to each variable

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

- A **structured representation for states**, expressed by a set of variable/value combinations and constraints represented in a formal language.
- General-purpose algorithms with more power than standard search algorithms exist.

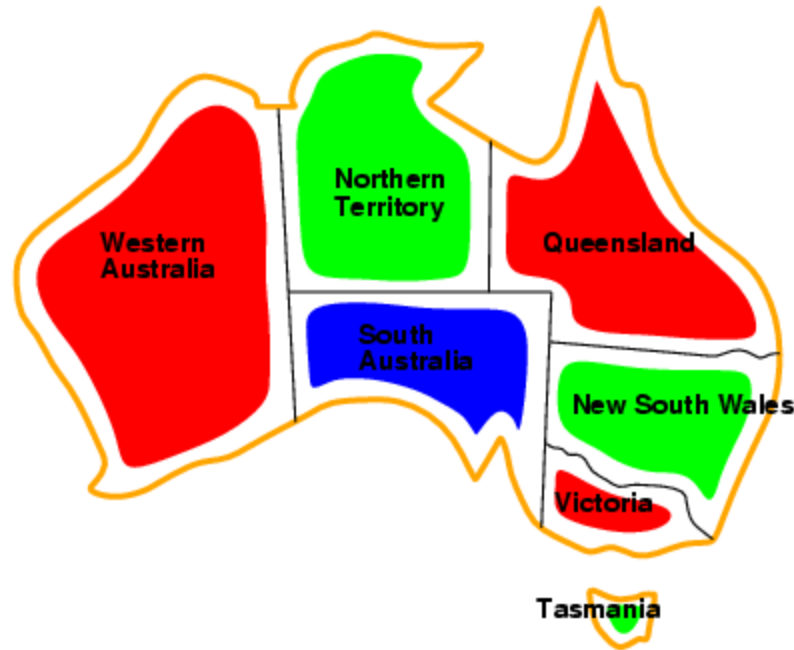


Example: Map Coloring (Graph coloring)



- **Variables representing state:** WA, NT, Q, NSW, V, SA, T
- **Variable Domains:** {red, green, blue}
- **Constraints:** adjacent regions must have different colors
e.g.,
$$WA \neq NT \Leftrightarrow (WA, NT) \in \{(\text{red}, \text{green}), (\text{red}, \text{blue}), (\text{green}, \text{red}), (\text{green}, \text{blue}), (\text{blue}, \text{red}), (\text{blue}, \text{green})\}$$

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

- **Variables:** X_{ij} for $i, j \in \{1, 2, \dots, N\}$
- **Domains:** $\{0, 1\}$ # Queen: no/yes

- **Constraints:**

$$\sum_{i,j} X_{ij} = N$$

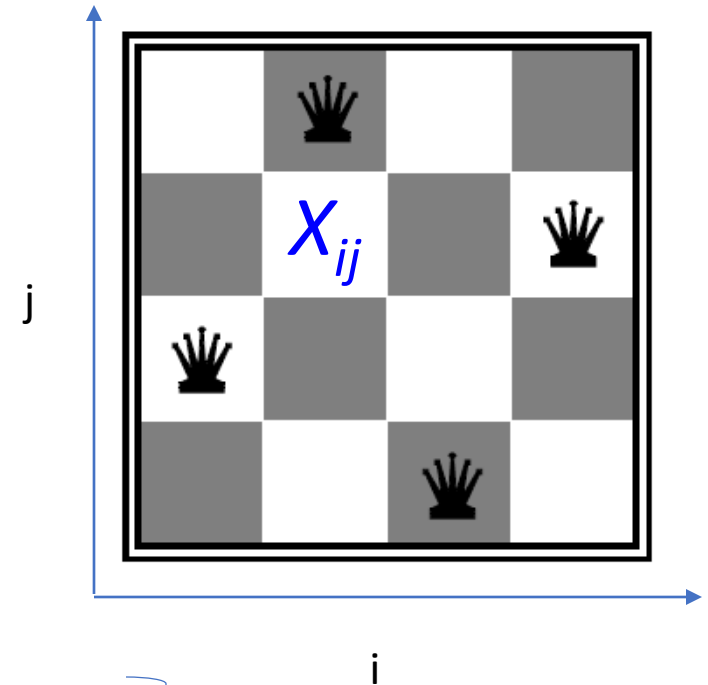
$(X_{ij}, X_{ik}) \in \{(0, 0), (0, 1), (1, 0)\}$ # cannot be in same col.

$(X_{ij}, X_{kj}) \in \{(0, 0), (0, 1), (1, 0)\}$ # cannot be in same row.

$(X_{ij}, X_{i+k, j+k}) \in \{(0, 0), (0, 1), (1, 0)\}$ # cannot be diagonal

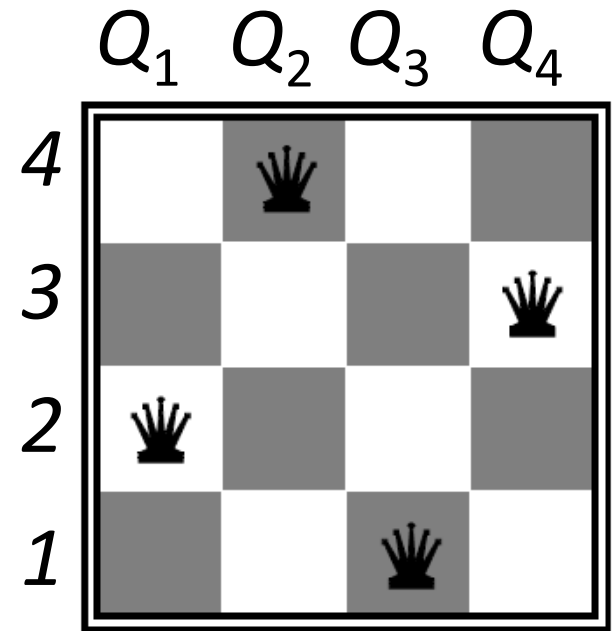
$(X_{ij}, X_{i+k, j-k}) \in \{(0, 0), (0, 1), (1, 0)\}$ # cannot be diagonal

for $i, j, k \in \{1, 2, \dots, N\}$



N-Queens: Alternative formulation

- **Variables:** Q_1, Q_2, \dots, Q_N
- **Domains:** $\{1, 2, \dots, N\}$ # row for each col.
- **Constraints:**
 $\forall i, j$ non-threatening (Q_i, Q_j)



Example:

$Q_1 = 2, Q_2 = 4, Q_3 = 1, Q_4 = 3$

Example: Cryptarithmic Puzzle

- **Variables:** T, W, O, F, U, R

X_1, X_2

- **Domains:** $\{0, 1, 2, \dots, 9\}$

- **Constraints:**

$\text{Alldiff}(T, W, O, F, U, R)$

$O + O = R + 10 * X_1$

$W + W + X_1 = U + 10 * X_2$

$T + T + X_2 = O + 10 * F$

$T \neq 0, F \neq 0$

Given Puzzle:

Find values for the letters.
Each letter stands for a
different digit.

$$\begin{array}{r} X_2 X_1 \\ T \quad W \quad O \\ + \quad T \quad W \quad O \\ \hline F \quad O \quad U \quad R \end{array}$$

Example: Sudoku

- **Variables:** X_{ij}
- **Domains:** $\{1, 2, \dots, 9\}$

- **Constraints:**

Alldiff(X_{ij} in the same *unit*)

Alldiff(X_{ij} in the same *row*)

Alldiff(X_{ij} in the same *column*)

					8			4
	8	4		1	6			
			5			1		
1		3	8			9		
6		8		X_{ij}		4		3
		2			9	5		1
		7			2			
			7	8		2	6	
2			3					

Some Popular Types of CSPs

- **Boolean Satisfiability Problem (SAT)**

Find variable assignments that makes a Boolean expression (often expressed in conjunctive normal form) evaluate as true.

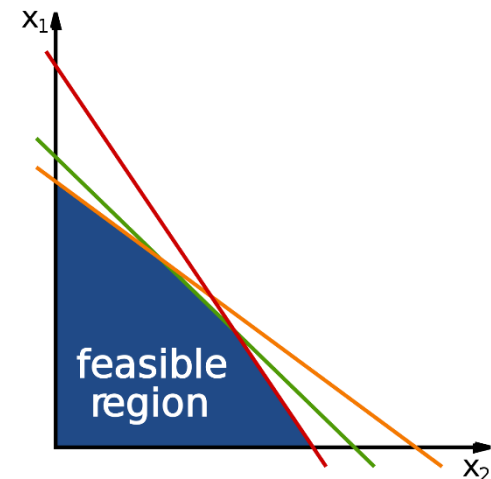
$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_1 = \text{True}$$

- **Integer Programming**

Variables are restricted to integers. Find a feasible solution that satisfies all constraints. The traveling salesman problem can be expressed as an integer program.

- **Linear Programming**

Variables are continuous and constraints are linear (in)equalities. Find a feasible solution.



Real-world CSPs

- Assignment problems
e.g., who teaches what class for a fixed schedule. Teacher cannot be in two classes at the same time!
- Timetable problems
e.g., which class is offered when and where? No two classes in the same room at the same problem.
- Scheduling in transportation and production (e.g., order of production steps).
- Many problems can naturally also be formulated as CSPs.
- More examples of CSPs: <http://www.csplib.org/>

CSP as a Standard Search Formulation

State:

- Values assigned so far

Initial state:

- The empty assignment $\{ \}$ (all variables are unassigned)

Successor function:

- Choose an unassigned variable and assign it a value that does not violate any constraints
- Fail if no legal assignment is found

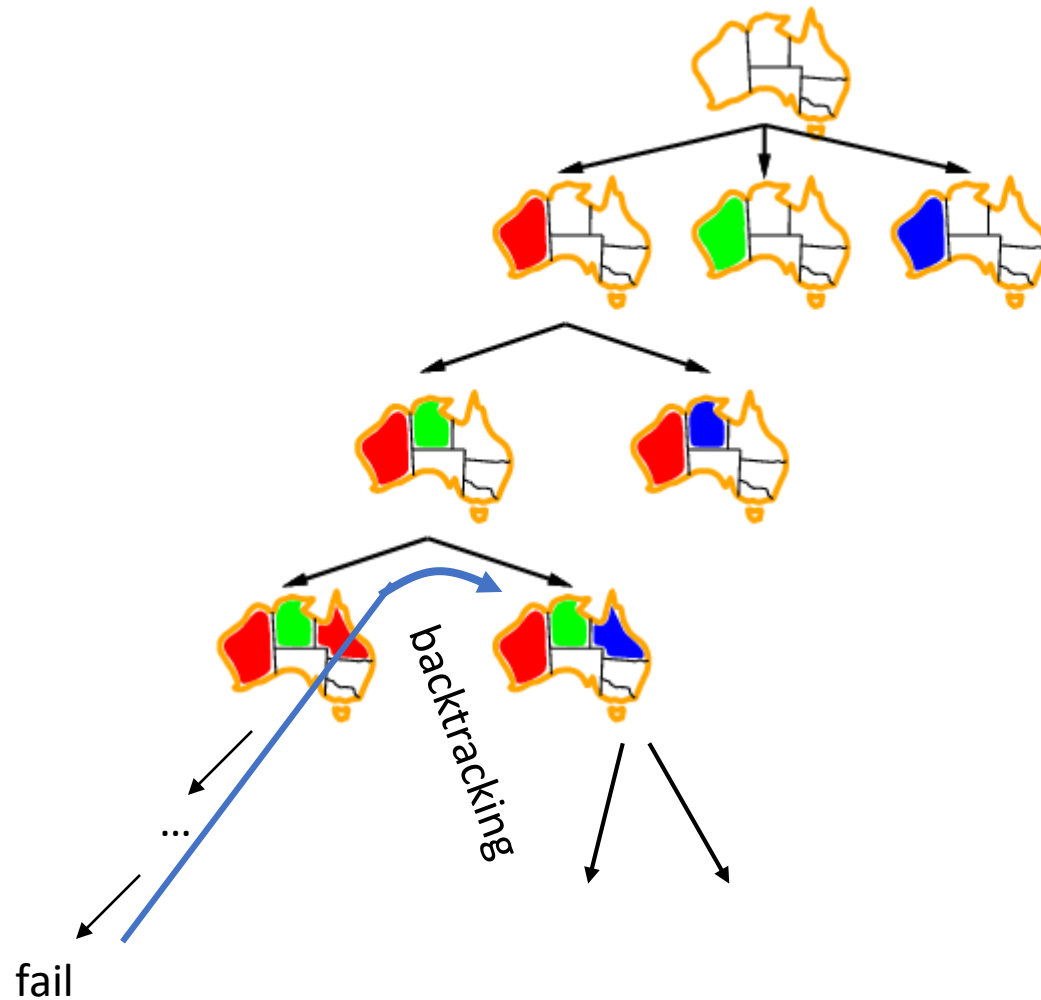
Goal state:

- Any complete and consistent assignment.

Backtracking search

- In CSP's, variable assignments are **commutative**
For example,
 $[WA = \text{red then } NT = \text{green}]$ is the same as
 $[NT = \text{green then } WA = \text{red}]$. \rightarrow Order is not important
- We can build a search tree that assigns the value to one variable per level.
 - Tree depth n (number of variables)
 - Number of leaves: d^n (d is the number of values per variable)
- Depth-first search for CSPs with single-variable assignments is called **backtracking search**

Example: Backtracking search (DFS)



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
```

Call: Recursive-Backtracking({}, *csp*)

Improving backtracking efficiency:

- 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? In which order should its values be tried?

- **Most constrained variable:**

- Keep track of remaining legal values for unassigned variables (using constraints)
- Choose the variable with the fewest legal values left
- A.k.a. **minimum remaining values** (MRV) heuristic

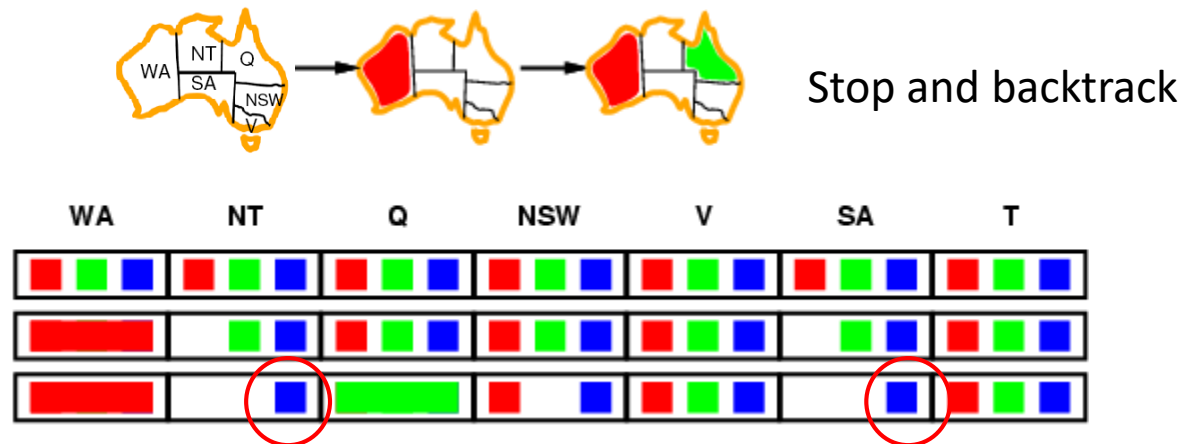
- Choose the **least constraining value:**

- The value that rules out the fewest values in the remaining variables

Early detection of failure – Forward checking

Node consistency

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values (i.e., minimum remaining values = 0)

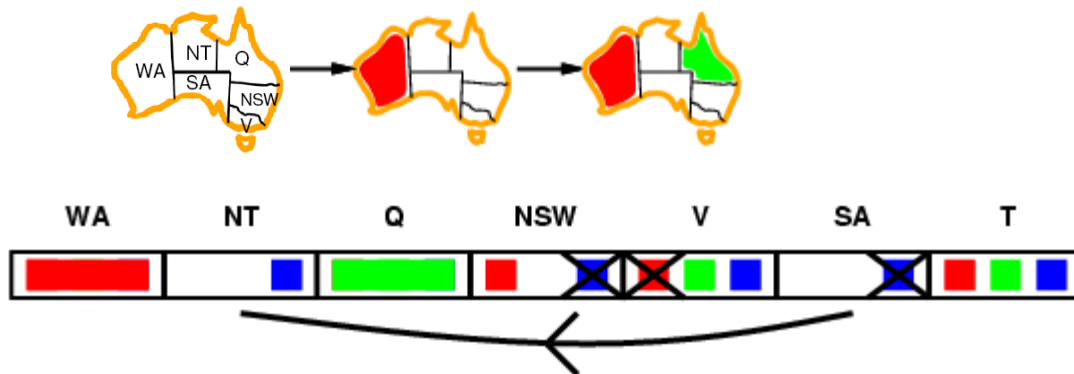


- NT and SA cannot both be blue! This violates the constraint.

Early detection of failure – Forward checking

Arc consistency

- X is arc consistent wrt Y iff for **every** value of X there is **some** allowed value of Y .
- Make X arc consistent wrt Y by throwing out any values of X for which there is no allowed value of Y .




1. NSW cannot be blue because SA has to be blue.
2. V cannot be red because NSW has to be red.
3. SA cannot be blue because NT is blue.
4. Fail and backtrack

- Arc consistency detects failure earlier than node consistency
- There are more consistency checks (path consistency, K-consistency)

Backtracking search with inference

```
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
```



Call: Recursive-Backtracking({}, csp)

```
If (inference(csp, var, assignment) == failure)  
  return failure
```

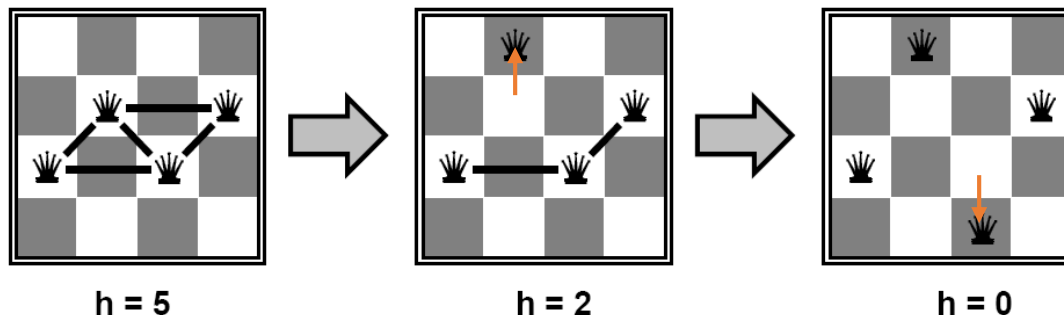
Check consistency here (called “inference”) and backtrack if we know that the branch will lead to failure.

Local search for CSPs

Local search is often very effective for CSPs.

Hill-climbing and simulated annealing work only with “complete” states, i.e., all variables assigned:

- Allow states with unsatisfied constraints
- Attempt to improve states by the **min-conflicts** heuristic:
 1. Select a conflicted variable and
 2. Choose a new value that produces violates the fewest constraints (local improvement step)
 3. Repeat till all constraints are met.



Summary

- CSPs are a special type of search problem:
 - States are **structured** and defined by a set of variables and values assignments
 - Variables can be unassigned
 - Goal test defined by
 - **Consistency** with constraints
 - **Completeness** of assignment
- **Backtracking search** = depth-first search where a successor state is generated by a consistent value assignment to a single unassigned variable
 - Starts with {} and only considers consistent assignments.
 - **Variable ordering** and **value selection** heuristics can help significantly
 - **Forward checking** prevents assignments that guarantee later failure
- Local search can be used to search the space of all complete assignments for consistent assignments = **min-conflicts heuristic**.