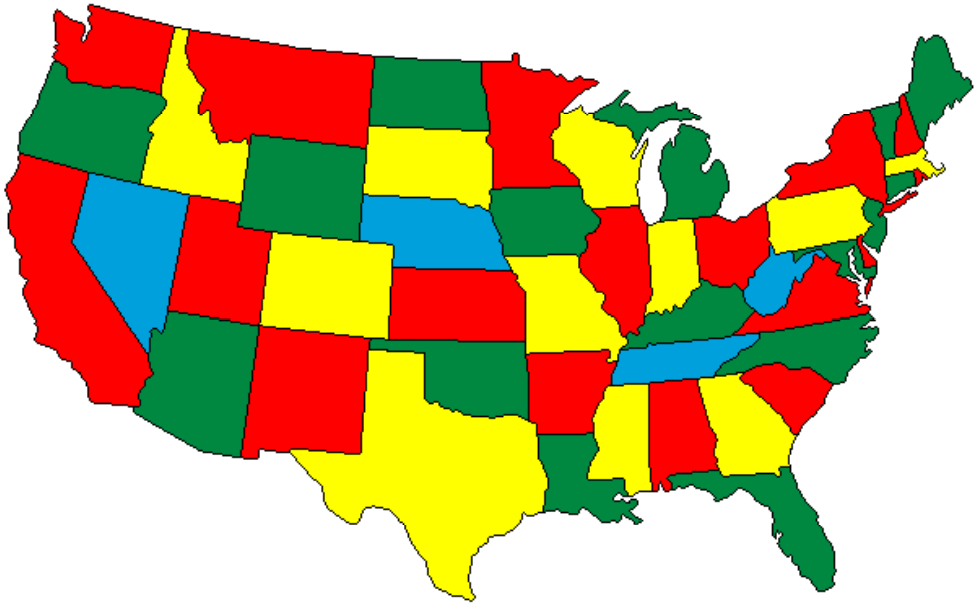
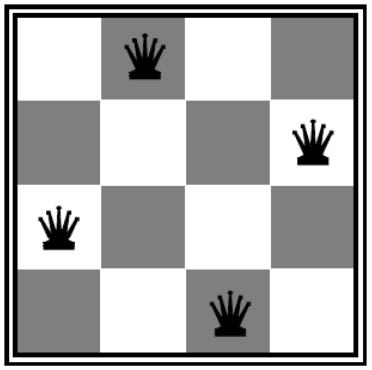
Constraint Satisfaction Problems





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

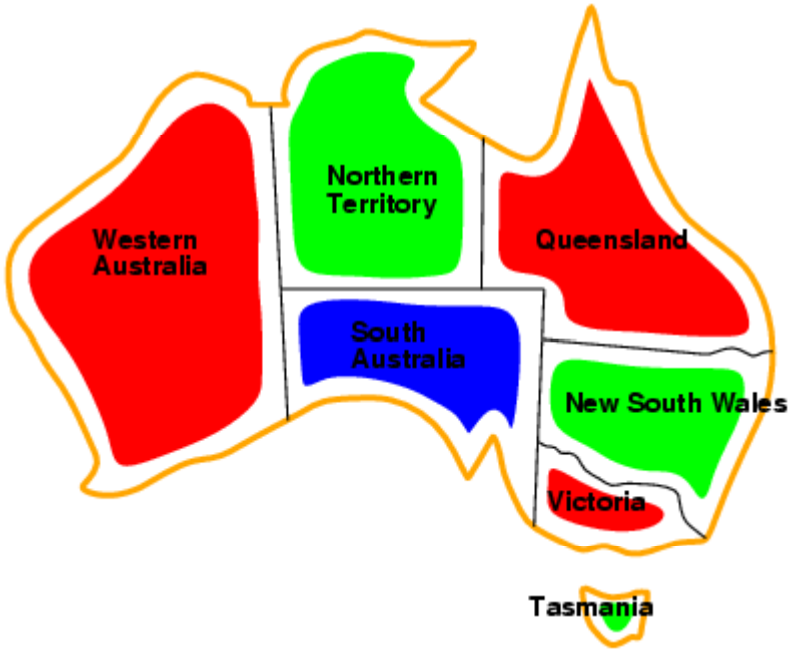
Constraint satisfaction problems (CSPs)

- Standard search problem:
  - **State** is a "black box" – any data structure that supports successor function, heuristic function, and goal test
- Constraint satisfaction problem:
  - **State** is defined by 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
- A simple example of a formal representation language
- Allows useful general-purpose algorithms with more power than standard search algorithms

Example: Map Coloring

- **Variables:** WA, NT, Q, NSW, V, SA, T
- **Domains:** {red, green, blue}
- **Constraints:** adjacent regions must have different colors
  e.g., WA ≠ NT, or (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

- **Variables:** X_{ij}
- **Domains:** {0, 1}
- **Constraints:**

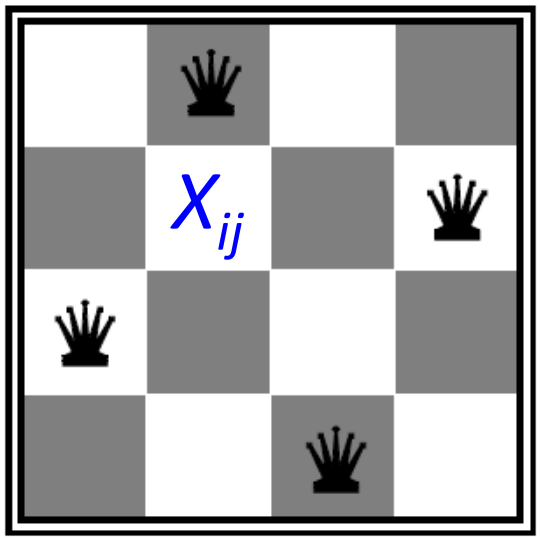
$\Sigma_{i,j} X_{ij} = N$

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

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

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

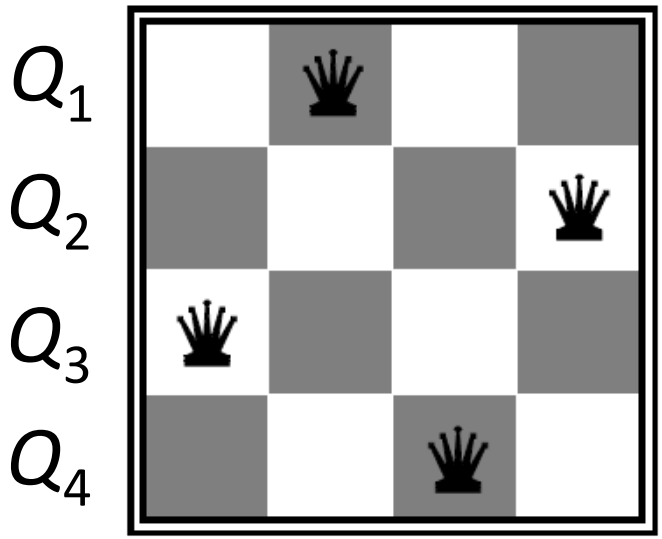
$(X_{ij}, X_{i+k,\, j-k}) \in \{(0, 0), (0, 1), (1, 0)\}$



N-Queens: Alternative formulation

- **Variables:** Q_i
- **Domains:** {1, … , N}
- **Constraints:**

  $\forall$ i, j non-threatening (Q_i , Q_j)



Example: Cryptarithmetic

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

  X_1, X_2
- **Domains**: {0, 1, 2, …, 9}
- **Constraints:**

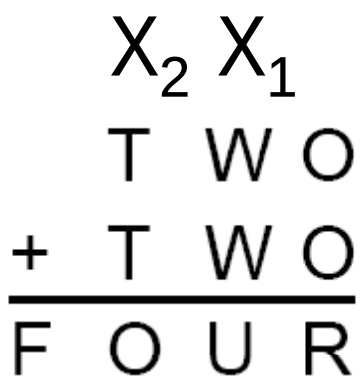
  Alldiff(T, W, O, F, U, R)

  $O + O = R + 10 * X_1$

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

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

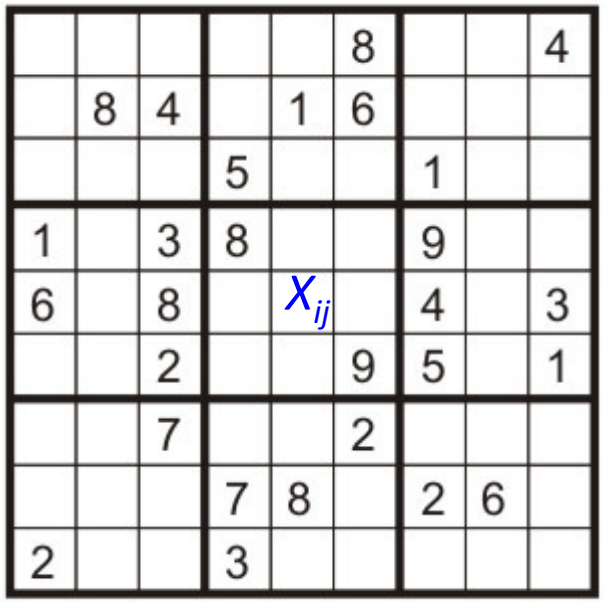
  $T \neq 0, F \neq 0$



Example: Sudoku

- **Variables:** X_{ij}
- **Domains:** {1, 2, …, 9}
- **Constraints:**

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



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/

Standard search formulation (incremental)

- **States**:
  - Values assigned so far
- **Initial state:**
  - The empty assignment { }
- **Successor function:**
  - 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?
  - n (with n variables assigned)
  - This is the good news (why?)
- Given that there are m possible values for any variable, how many paths are there in the search tree?
  - $n! \cdot m^n$
  - This is the bad news
- How can we reduce the branching factor?

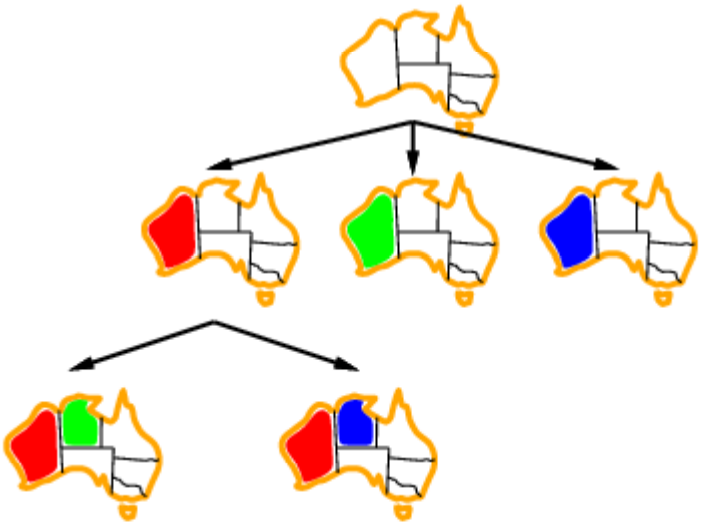
Backtracking search

- 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)
  - Then there are only m^n leaves
- Depth-first search for CSPs with single-variable assignments is called **backtracking search**

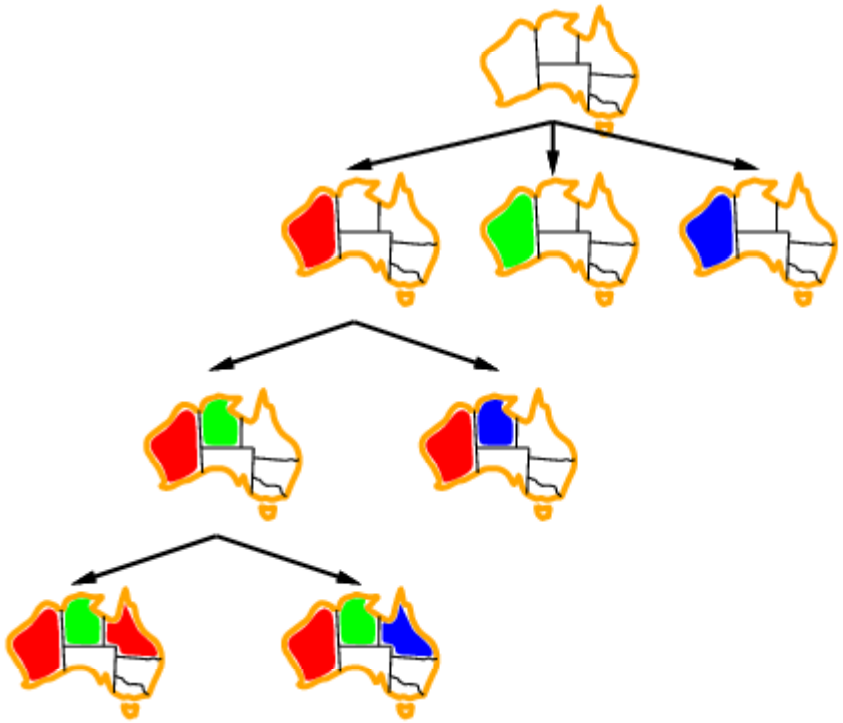
Example

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
```

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