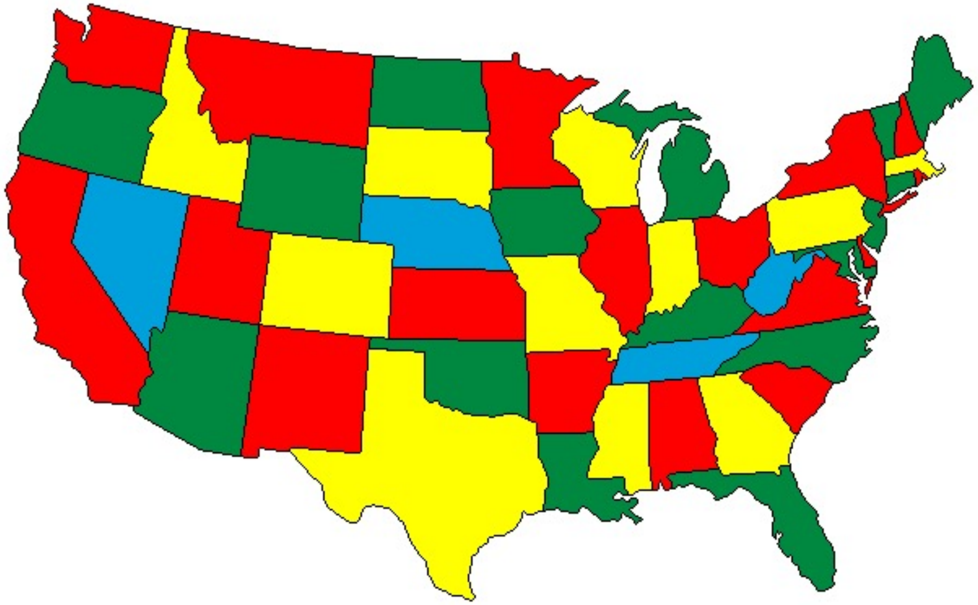
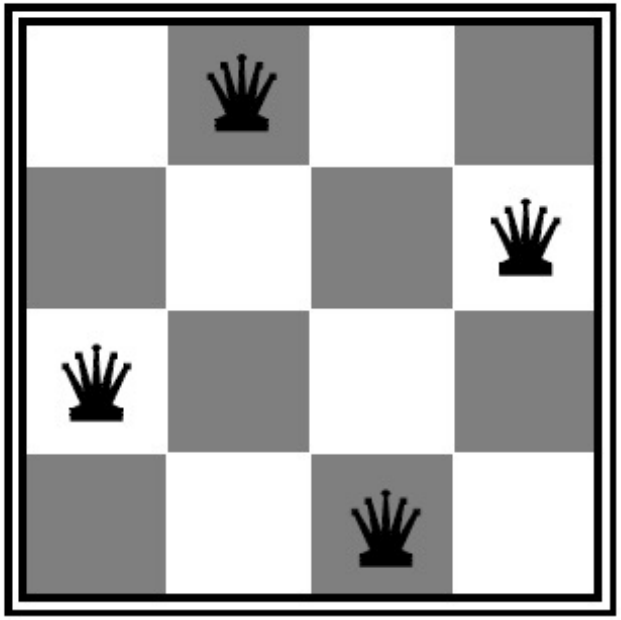
Constraint Satisfaction Problems





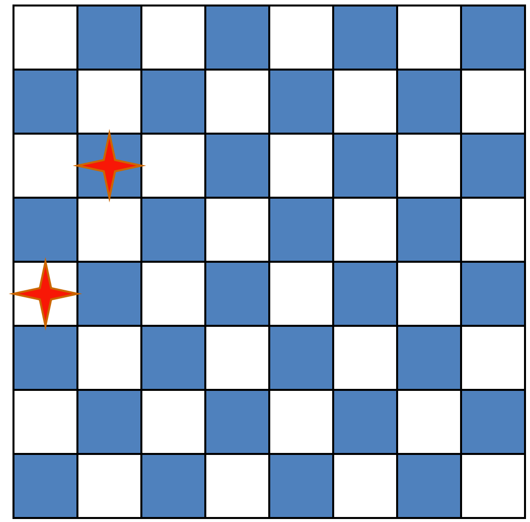
Sanja Lazarova-Molnar

Overview

- Constraint satisfaction is a powerful problem-solving paradigm
  - Problem: set of variables to which we must assign values satisfying problem-specific constraints
  - Constraint programming, constraint satisfaction problems (CSPs), constraint logic programming…
- Algorithms for CSPs
  - Backtracking
  - Constraint propagation
  - Variable and value ordering heuristics

Motivating example: 8 Queens

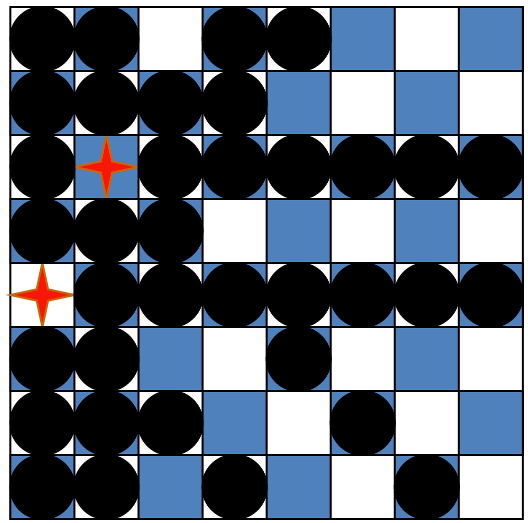
Place 8 queens on a chess board such
That none is attacking another.



Generate-and-test, with no
redundancies → "only" 8^8 combinations

8**8 is 16,777,216

Motivating example: 8-Queens



After placing these two queens, it's trivial to mark the squares we can no longer use

What more do we need for 8 queens?

- Not just a successor function and goal test
- But also
  - a way to propagate constraints imposed by one queen on others
  - an early failure test

CSP Definitions

- **Variables** - $X_1, X_2, \ldots, X_n$ each X_i having a non-empty domain D_i of possible values.
- **Constraints** - $C_1, C_2, \ldots, C_m$ consisting of some subset of variables and specifies allowable combinations of values for that subset.
- **State** - defined by an *assignment* of values to some or all variables ($X_1=v_1$, $X_j=v_j$, ...)
- **Consistent** - assignment that does not violate any constraints.
- **Solution** - complete assignment that satisfies all constraints

CSP Formulation

- **Initial state** - empty assignment: all variables are unassigned.
- **Successor function** - assigns value to an unassigned variable that does not conflict with previously assigned variables
- **Goal test** - complete current assignment
- **Path cost** - constant cost per step

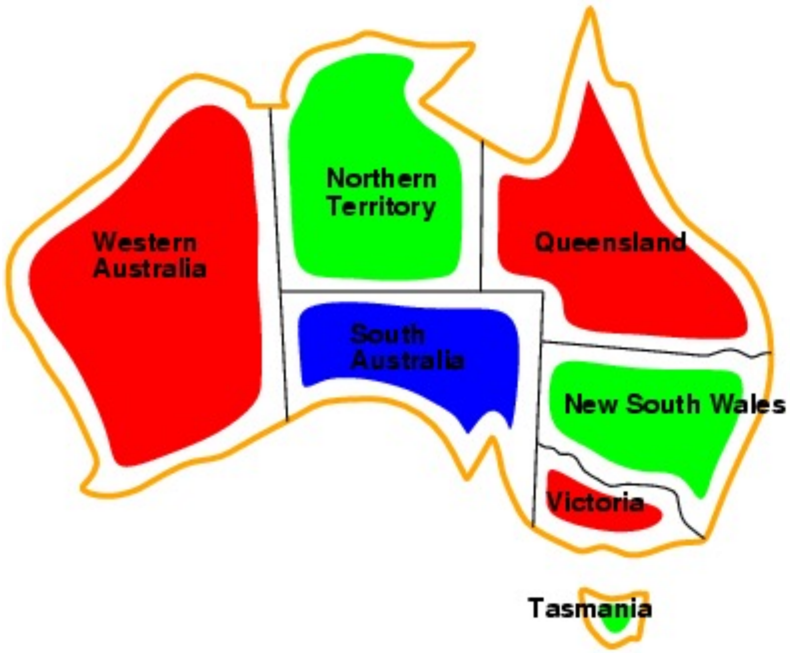
Example: Map Coloring

Variables: WA, NT, Q, NSW, V, SA, T

Domains: {red, green, blue}

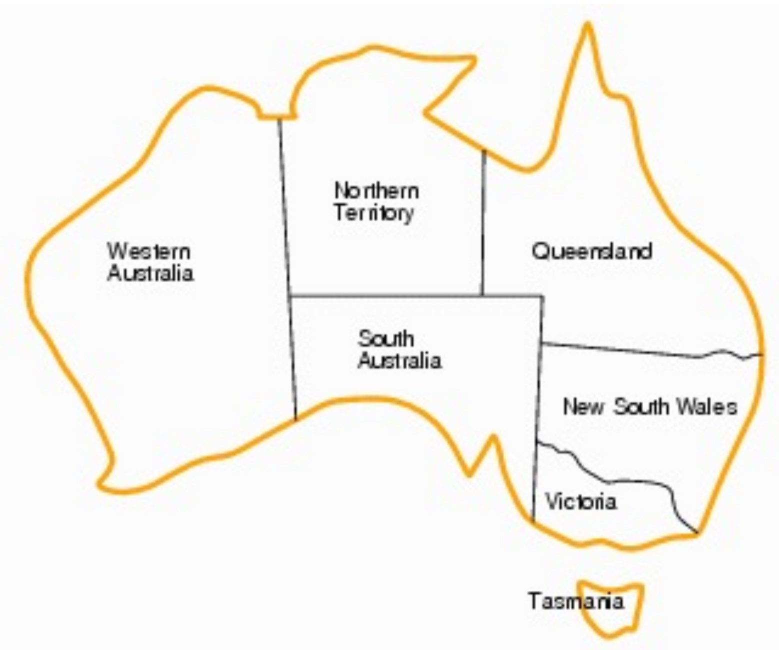
Constraints: adjacent regions must have different colors
e.g., WA ≠ NT, or (WA, NT) in {(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)}

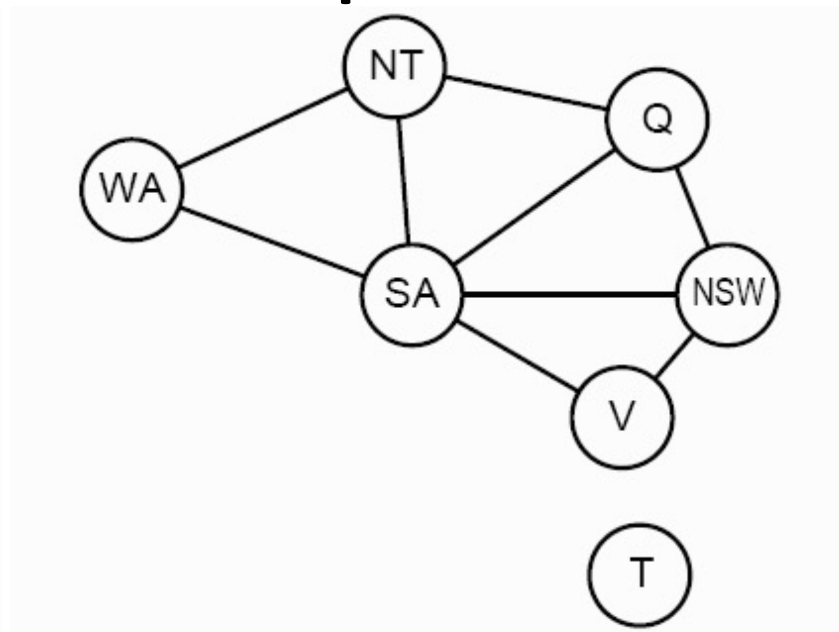
Example: Map Coloring



- State - one of many but not a solution, e.g. WA = red, NT = red, Q = red, NSW = red, V = red, SA = red, T = red
- **Solutions** are *complete* and *consistent* assignments, e.g., WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

Constraint Graph



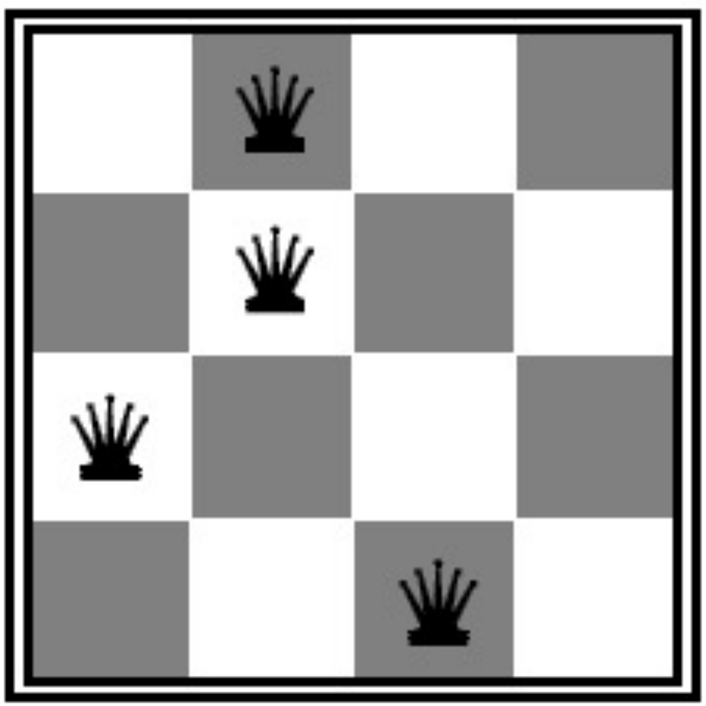


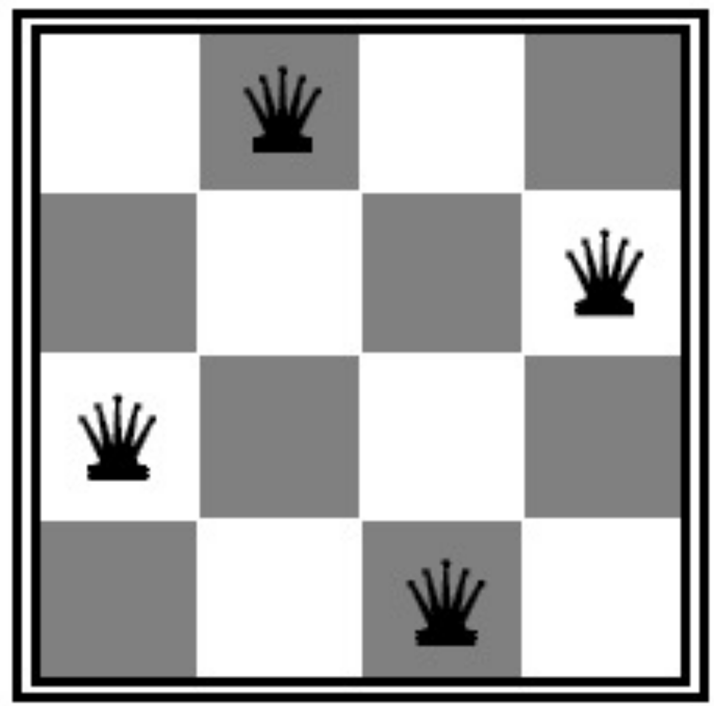
Nodes are variables, arcs show constraints.

- The constraint a != b in map coloring means that no adjacent Australian states are the same color.
- What is meaning of arcs from SA under the a != b constraint?
- What about the fact that there are no arcs to T.
- What is the implication of the arcs between WA, NT and SA?

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:**

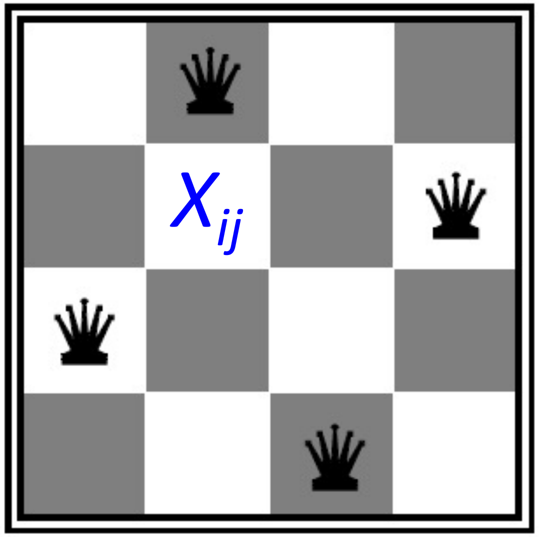
$\sum_{i,j} X_{ij} = N$

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

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

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

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



Example: Cryptarithmetic

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

  X_1, X_2

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

  $O + O = R + 10 * X_1$

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

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

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

  $T \neq 0, F \neq 0$

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

Example: Sudoku

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

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

					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					

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:**
  - 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)?

  n (this is good)

- 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 bad)

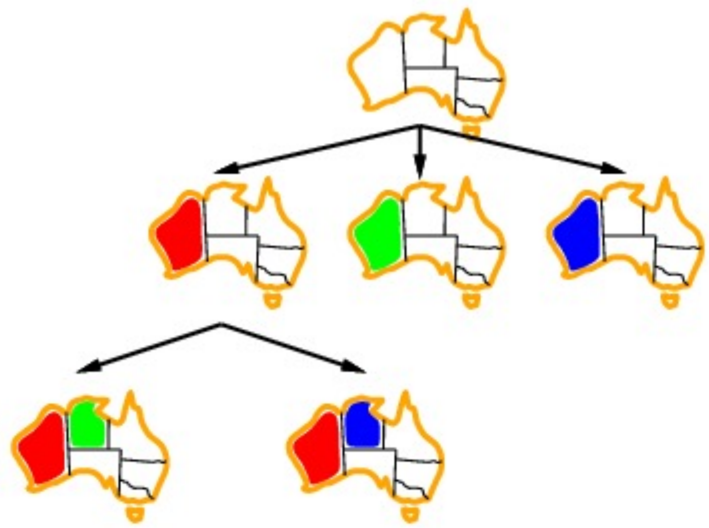
Backtracking search

- In CSPs, 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 $\boldsymbol{m}^{n}$ leaves (n – number of variables and m – number of values)
- Depth-first search for CSPs with single-variable assignments is called **backtracking search**

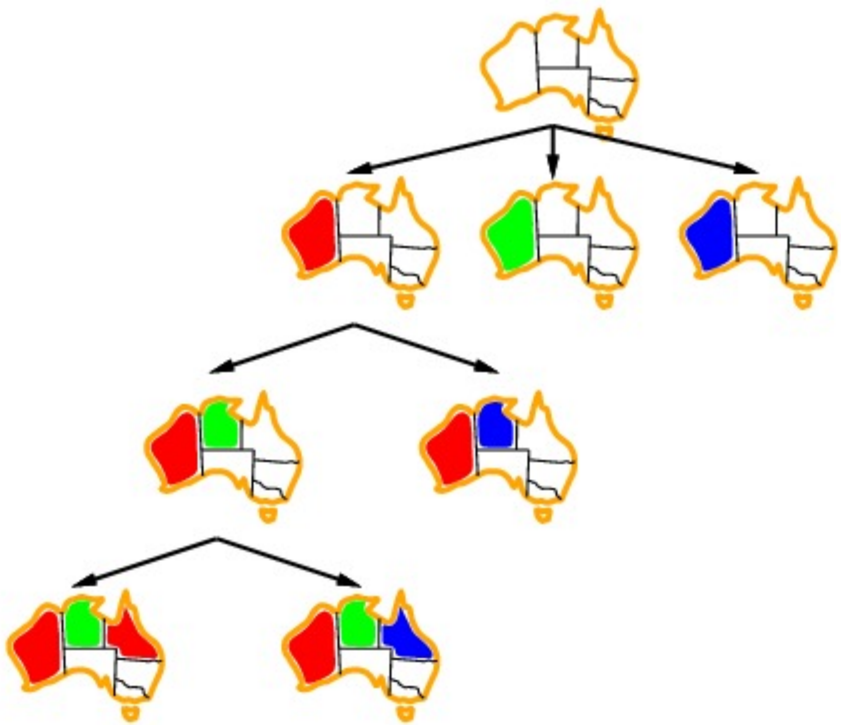
Example

Example

Example

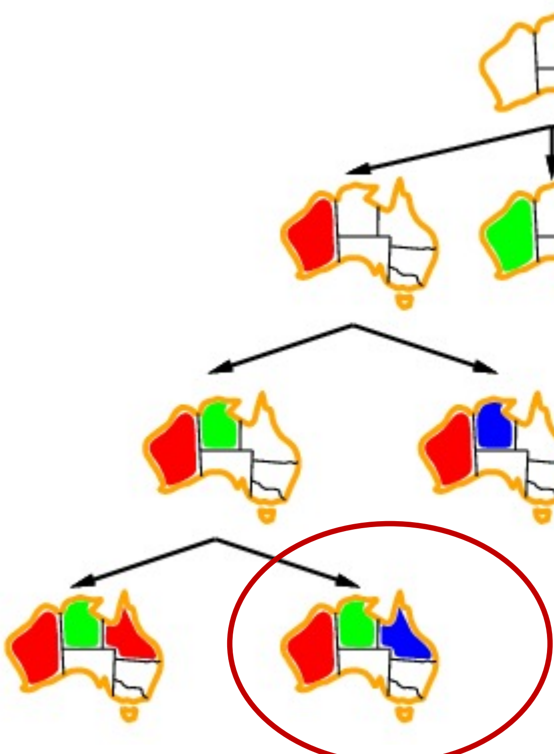


Example



Backtracking

- Constraints on SA will eventually cause failure when WA != Q. When not the same color (bottom right), SA cannot be assigned.
- The algorithm will backtrack to a node with unexplored states.
- For example, such as WA=red, NT=blue.



Basic backtracking algorithm

CSP-BACKTRACKING(PartialAssignment A)

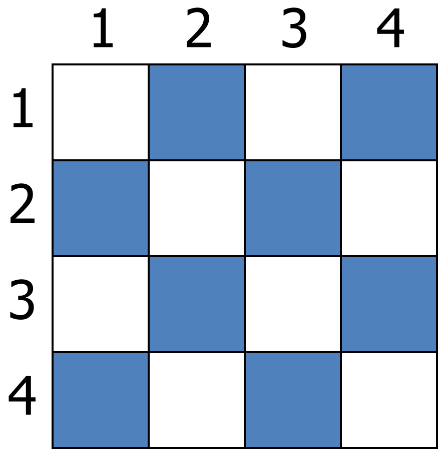
- If A is complete then return A
- X ← select an unassigned variable
- D ← select an ordering for the domain of X
- For each value v in D do

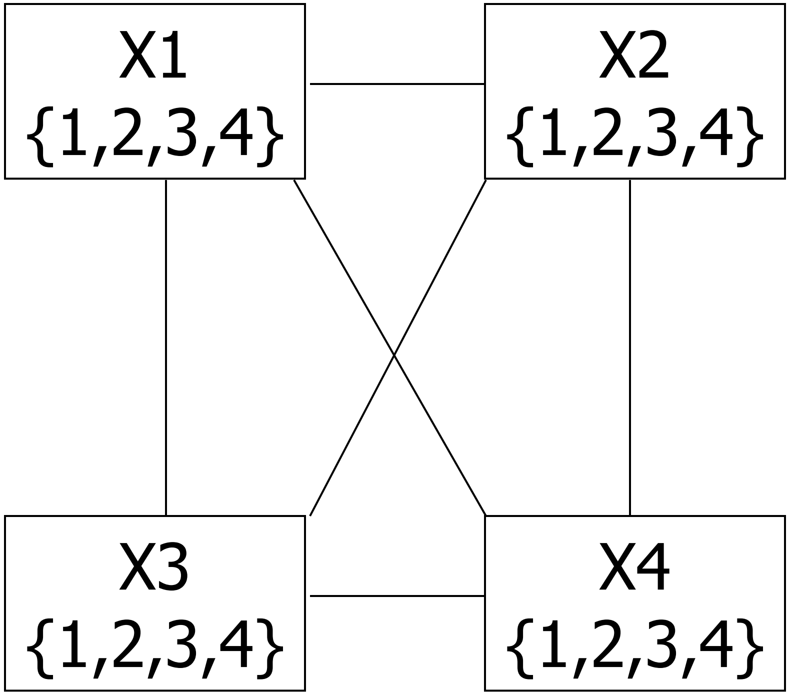
  If v consistent with A then

    - Add (X = v) to A
    - result ← CSP-BACKTRACKING(A)
    - If result ≠ failure then return result
    - Remove (X = v) from A
- Return failure

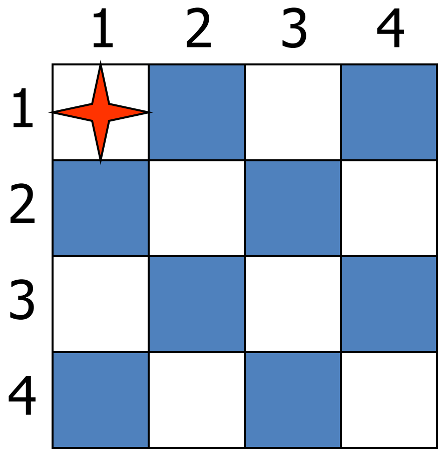
Start with CSP-BACKTRACKING({})

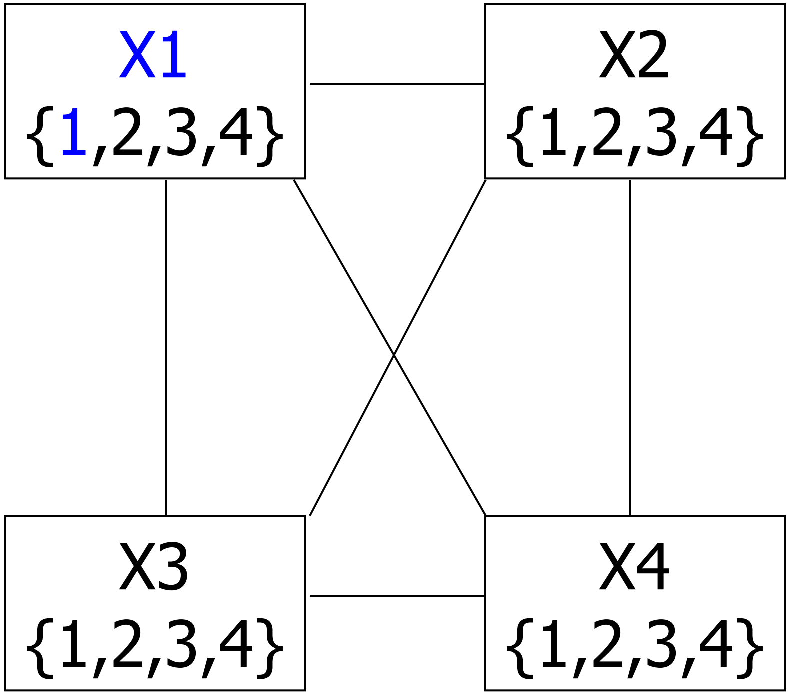
Example: 4-Queens Problem



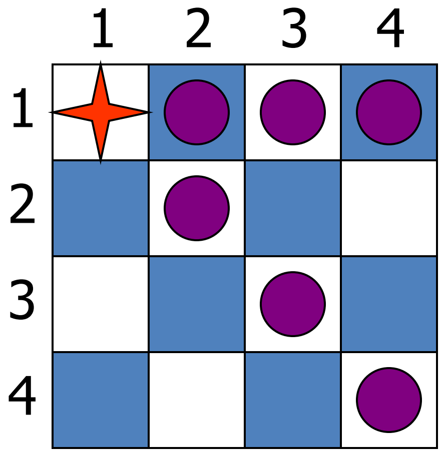


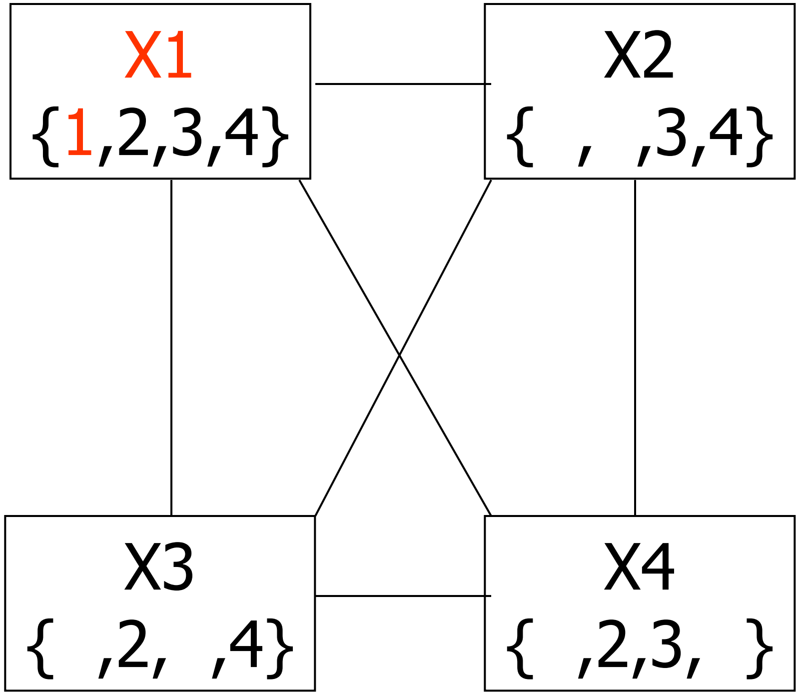
Example: 4-Queens Problem



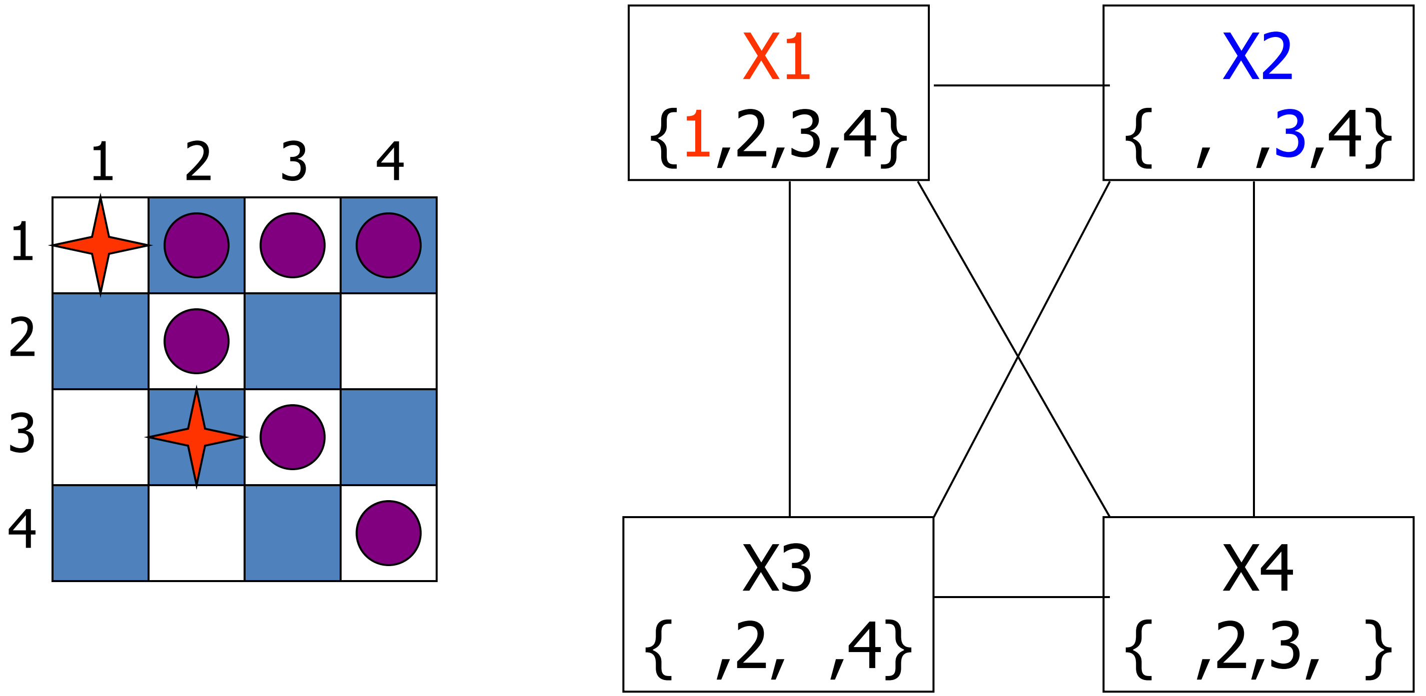


Example: 4-Queens Problem

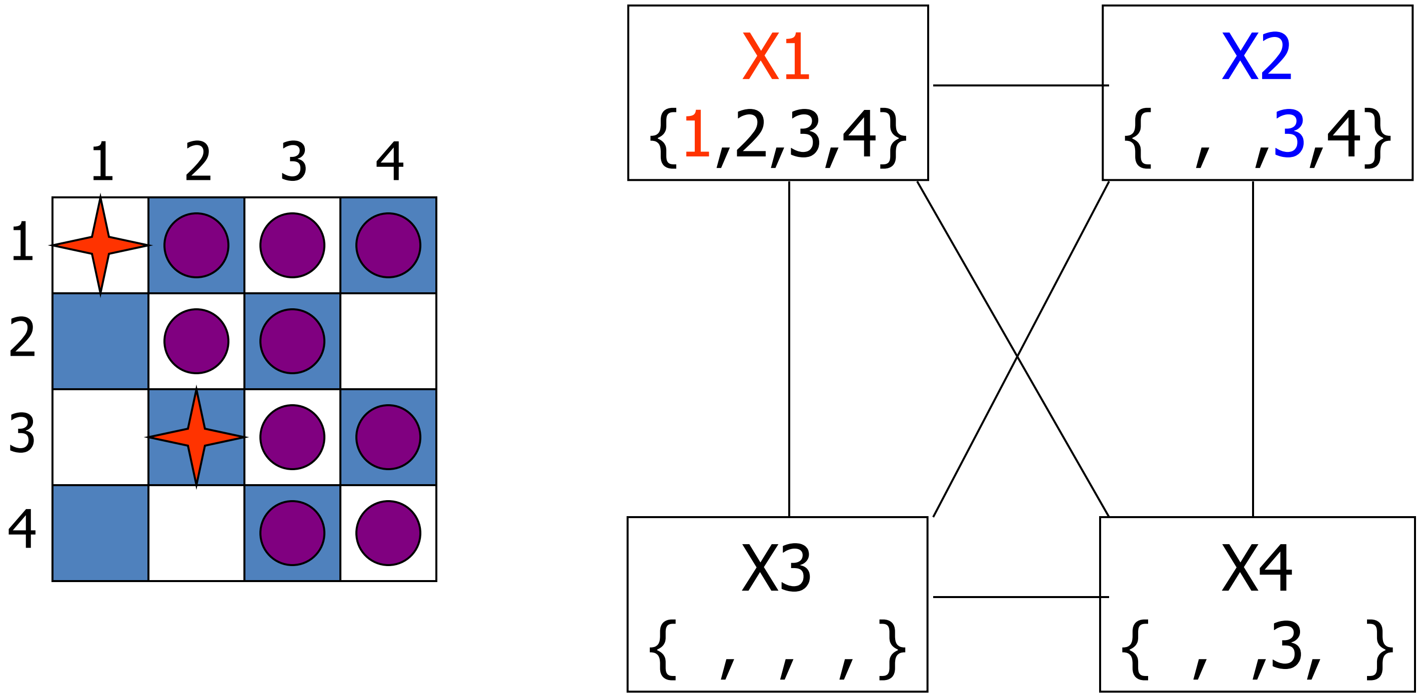




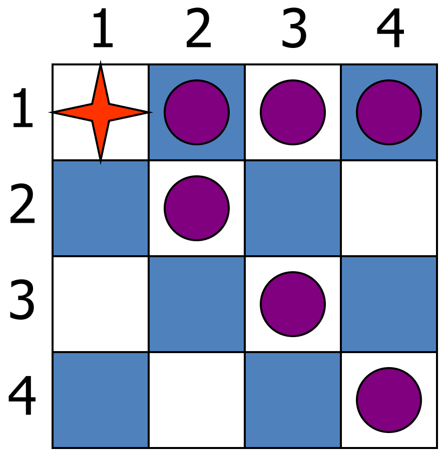
Example: 4-Queens Problem

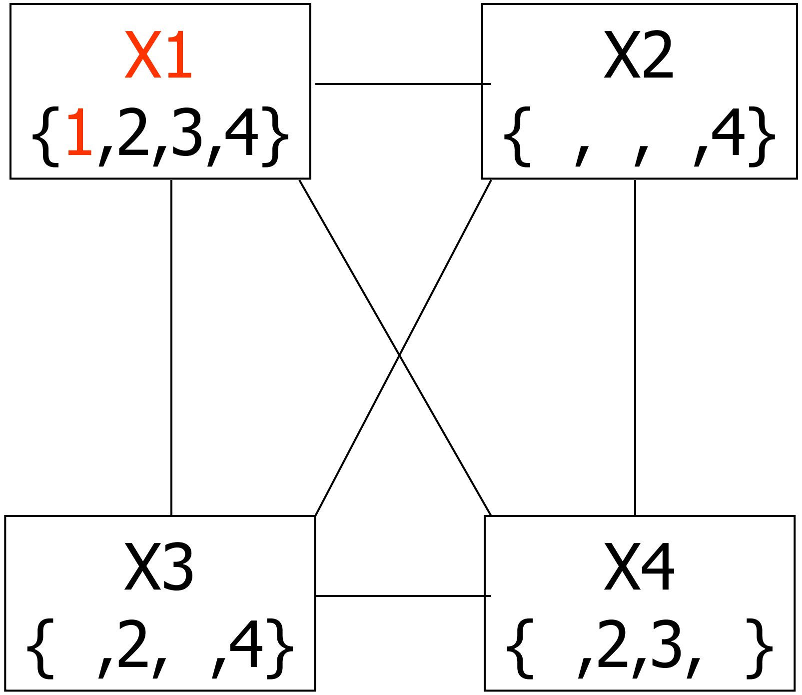


Example: 4-Queens Problem

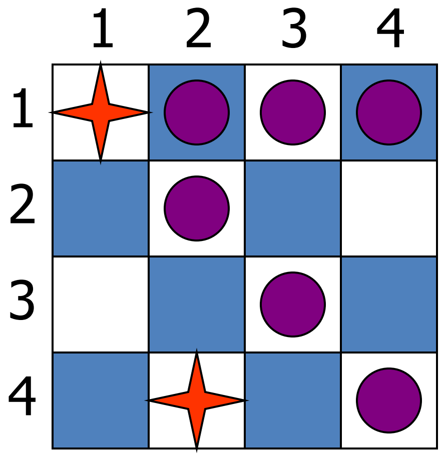


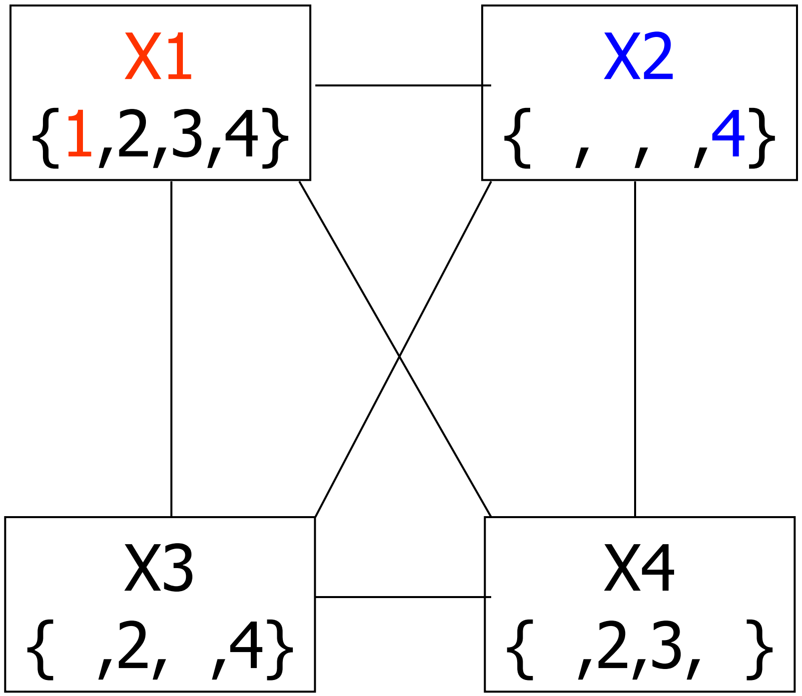
Example: 4-Queens Problem



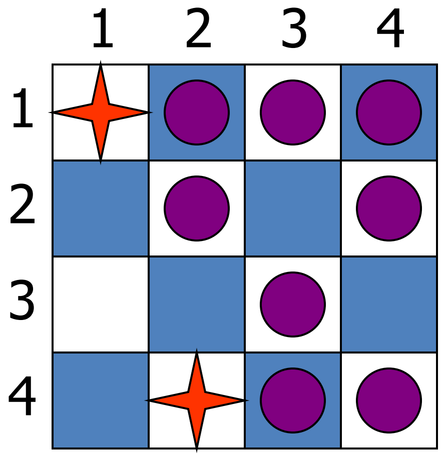


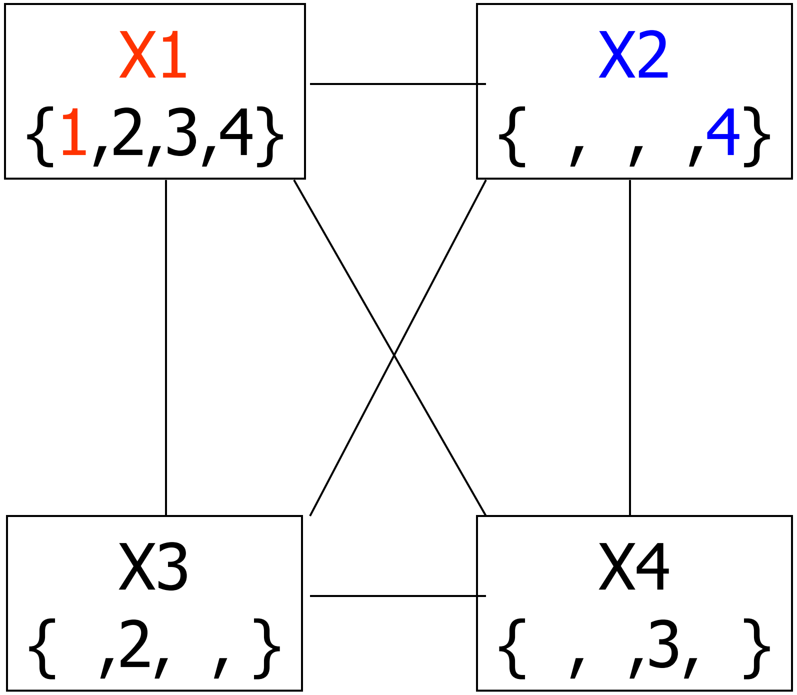
Example: 4-Queens Problem



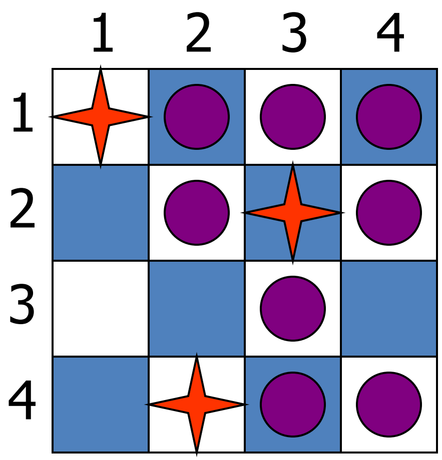


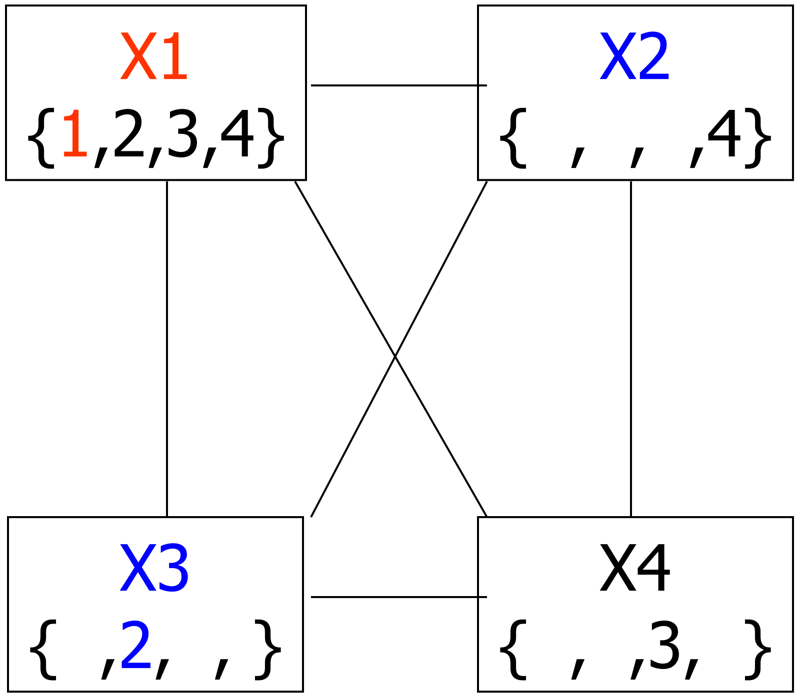
Example: 4-Queens Problem



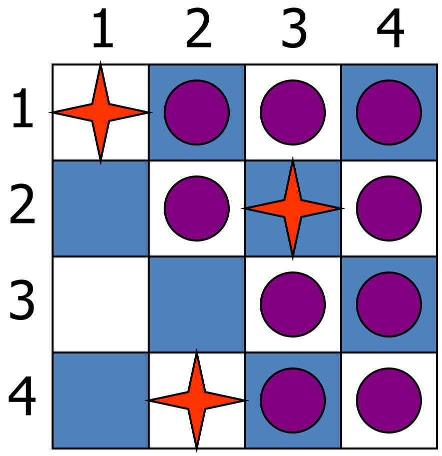


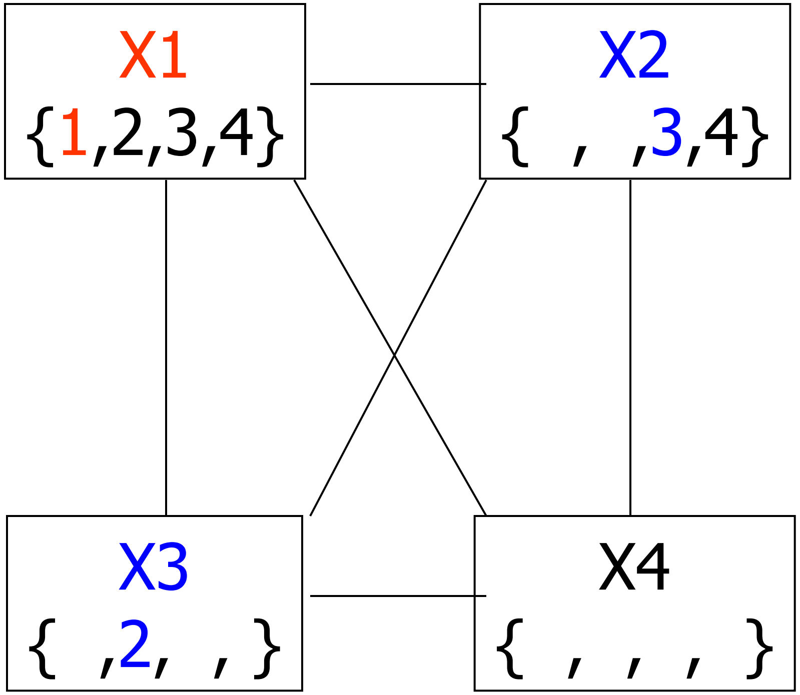
Example: 4-Queens Problem





Example: 4-Queens Problem





Improving backtracking efficiency

Questions:

- 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?

- **Most constrained variable:**
  - Choose the variable with the fewest legal values
  - A.k.a. **minimum remaining values** (MRV) heuristic

Which variable should be assigned next?

- **Most constrained variable:**
  - Choose the variable with the fewest legal values
  - A.k.a. **minimum remaining values** (MRV) heuristic

Which variable should be assigned next?

- **Most constraining variable:**
  - Choose the variable that imposes the most constraints on the remaining variables
  - Tie-breaker among most constrained variables

Which variable should be assigned next?

- **Most constraining variable:**
  - Choose the variable that imposes the most constraints on the remaining variables
  - Tie-breaker among most constrained variables

  N.B. Among the variables with the smallest remaining domains, select the one that appears in the largest number of constraints on variables not in the current assignment

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

- Choose the **least constraining value**:
  - The value that rules out the fewest values in the remaining variables

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

- Choose the **least constraining value**:
  - The value that rules out the fewest values in the remaining variables

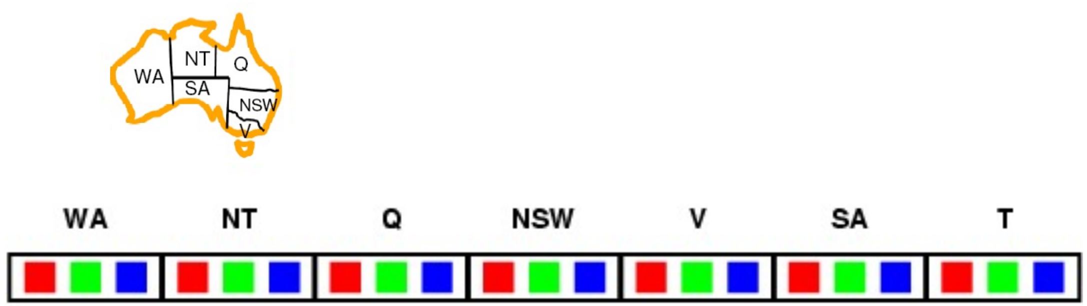
Which assignment for Q should we choose?

Early detection of failure: Forward checking

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

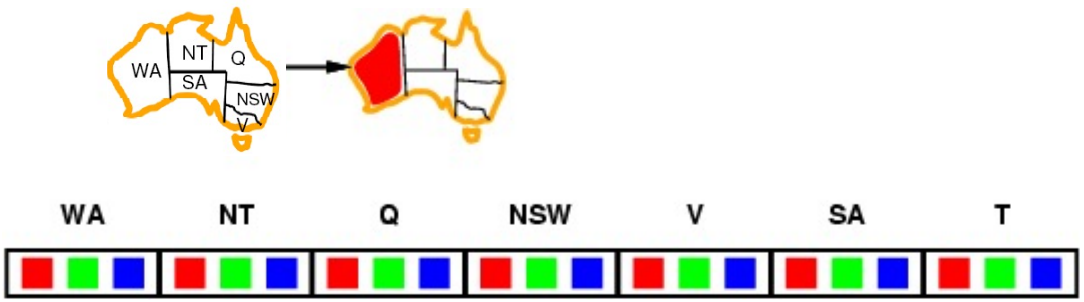
Early detection of failure: Forward checking

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



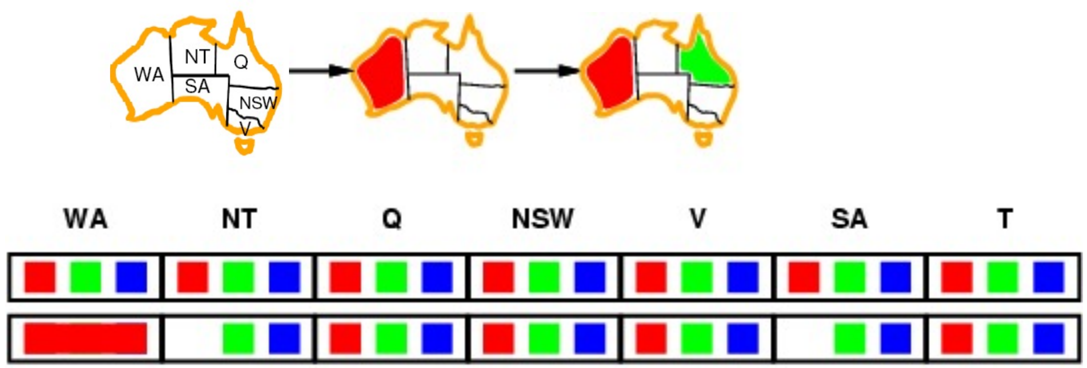
Early detection of failure: Forward checking

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



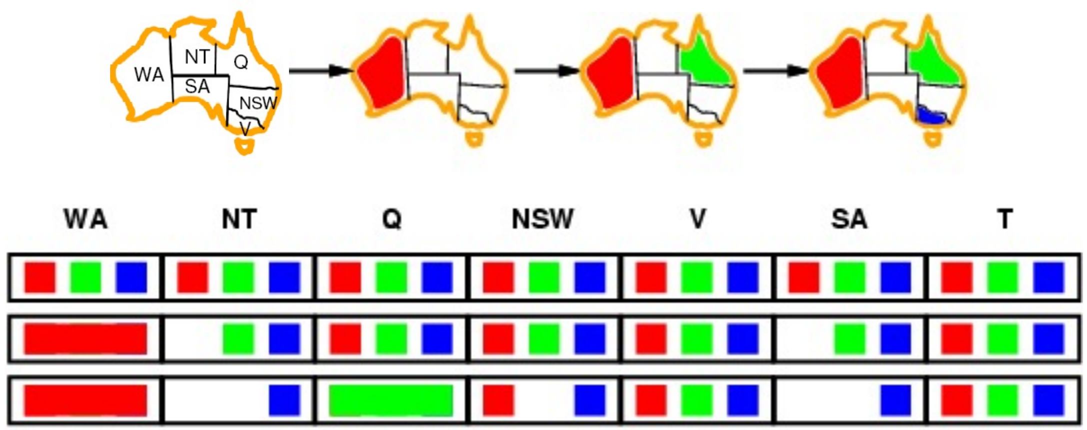
Early detection of failure: Forward checking

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



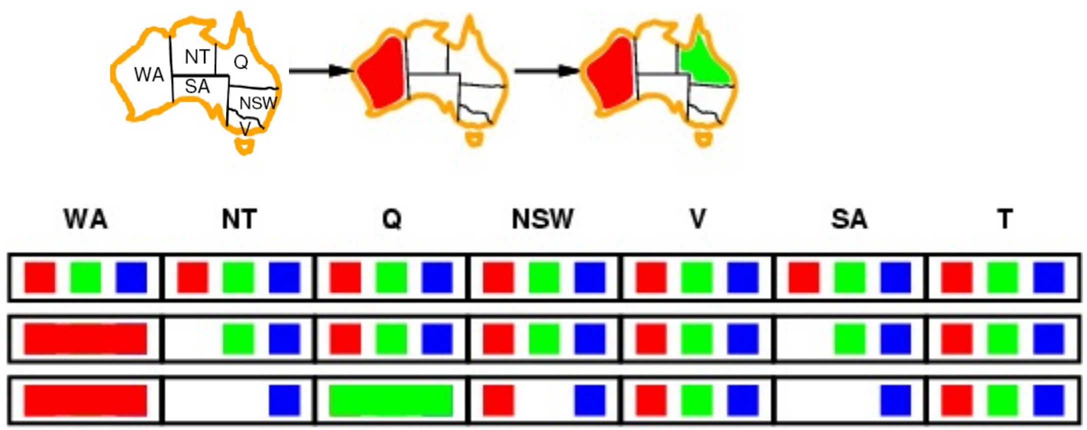
Early detection of failure: Forward checking

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



Constraint propagation

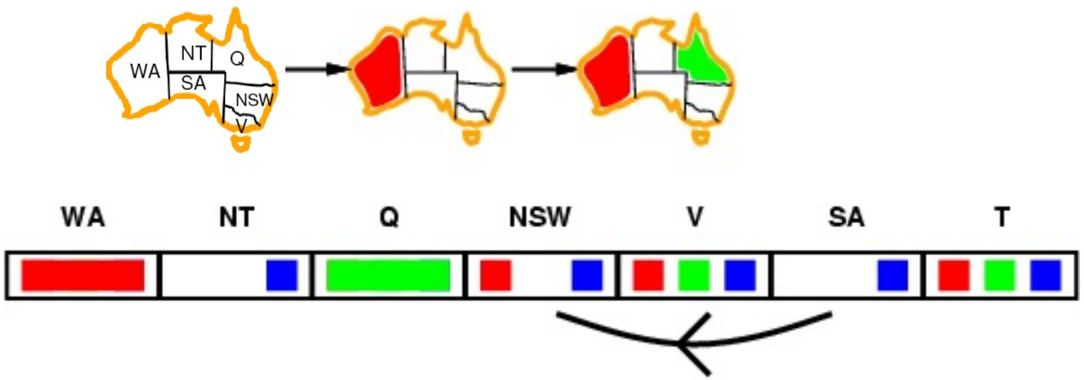
- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures



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

Arc consistency

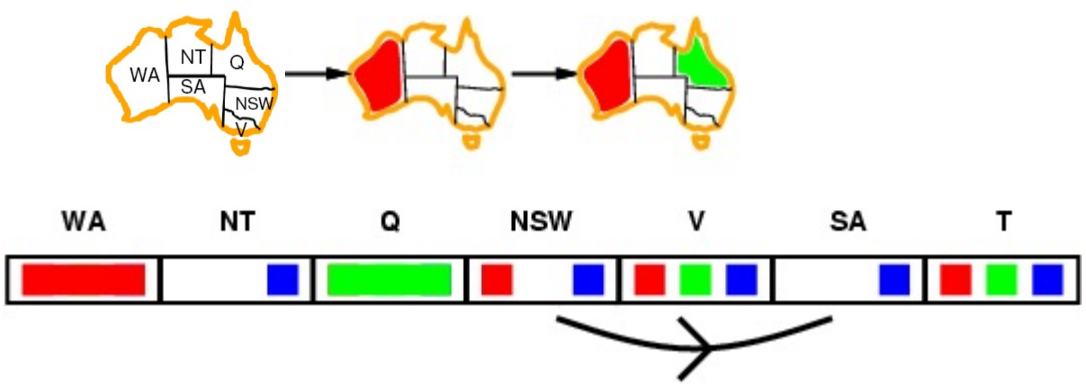
- Simplest form of propagation makes each pair of variables **consistent:**
  - *X* → *Y* is consistent iff for every value of *X* there is some allowed value of *Y*



Consistent!

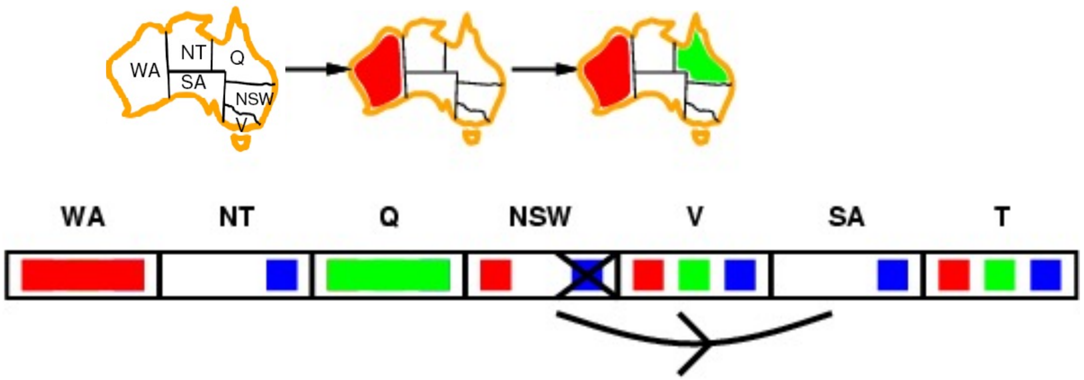
Arc consistency

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



Arc consistency

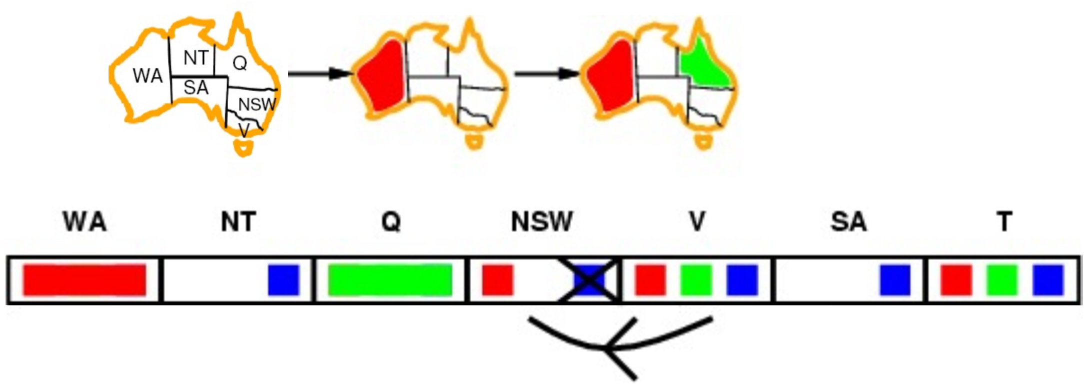
- Simplest form of propagation makes each pair of variables **consistent:**
  - *X* → *Y* is consistent iff for every value of *X* there is some allowed value of *Y*
  - When checking *X* → *Y*, throw out any values of X for which there isn't an allowed value of Y



- If *X* loses a value, all pairs *Z* → *X* need to be rechecked

Arc consistency

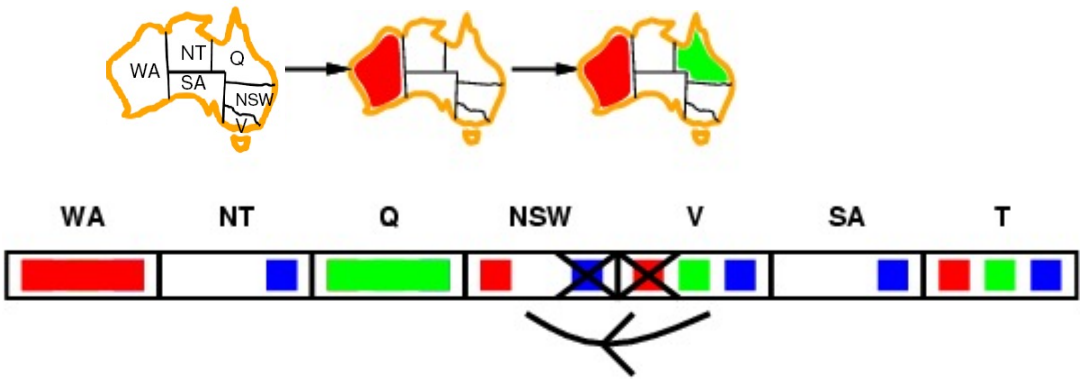
- Simplest form of propagation makes each pair of variables **consistent:**
  - *X* → *Y* is consistent iff for every value of *X* there is some allowed value of *Y*
  - When checking *X* → *Y*, throw out any values of X for which there isn't an allowed value of Y



- If *X* loses a value, all pairs *Z* → *X* need to be rechecked

Arc consistency

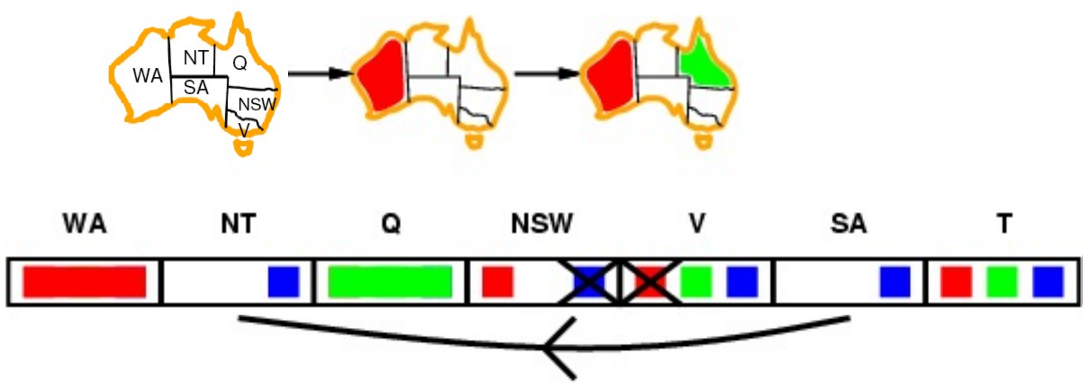
- Simplest form of propagation makes each pair of variables **consistent:**
  - *X* → *Y* is consistent iff for every value of *X* there is some allowed value of *Y*
  - When checking *X* → *Y*, throw out any values of X for which there isn't an allowed value of Y



- If *X* loses a value, all pairs *Z* → *X* need to be rechecked

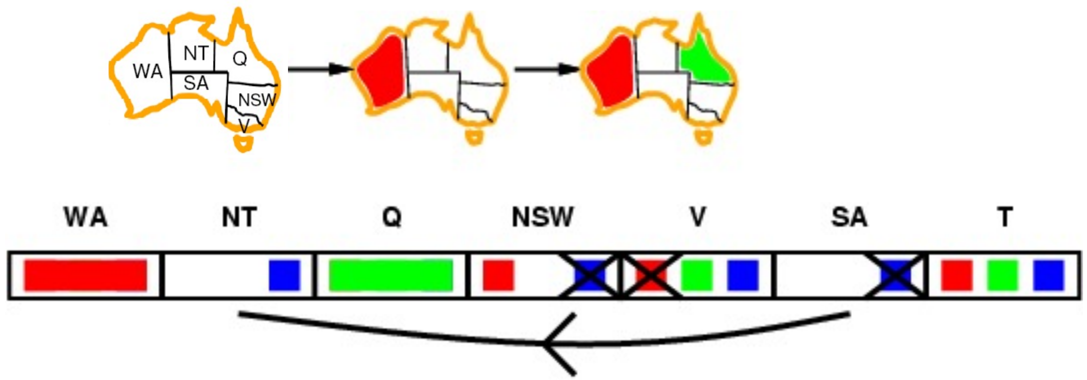
Arc consistency

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



Arc consistency

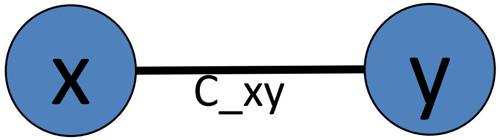
- Simplest form of propagation makes each pair of variables **consistent:**
  - *X* → *Y* is consistent iff for every value of *X* there is some allowed value of *Y*
  - When checking *X* → *Y*, throw out any values of X for which there isn't an allowed value of Y



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

Arc consistency - Example

- Domains
  - D_x = {1, 2, 3}
  - D_y = {1, 2, 3}



- Constraint: X must be <u>less than</u> Y (C_xy)
- C_xy not arc consistent w.r.t. x or y; enforcing arc consistency, we get reduced domains:
  - D'_x = {1, 2}
  - D’_y = {2, 3}

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) simple yet powerful to constrain values and detect inconsistencies
- Complexity of CSPs
  - NP-complete in general (exponential worst-case running time)