COSC 4368
Fundamentals of Artificial Intelligence

Constraint Satisfaction Problems
September 18th, 2023

Constraint Satisfaction Problems

- Regular search problems:
  - Search in a state space
  - Each state is indivisible --- a black box with no internal structure

- Constraint satisfaction problem
  - Each state has a factored representation: a set of variables, each has a value
  - Goal condition consists of a set of sub-conditions: each variable should have a value that satisfies all the constraints on the variable

What is a Constraint Satisfaction Problem (CSP)

- A CSP consists of three components:
  - A finite set of variables:
  - A finite set of domains:
    - One for each variable; finite or infinite domain
  - A finite set of constraints:
    - Each constraint involves some subset of the variables
    - Specifies allowable combinations of values for that subset
    - E.g., <(,), >

What is a Constraint Satisfaction Problem (CSP)

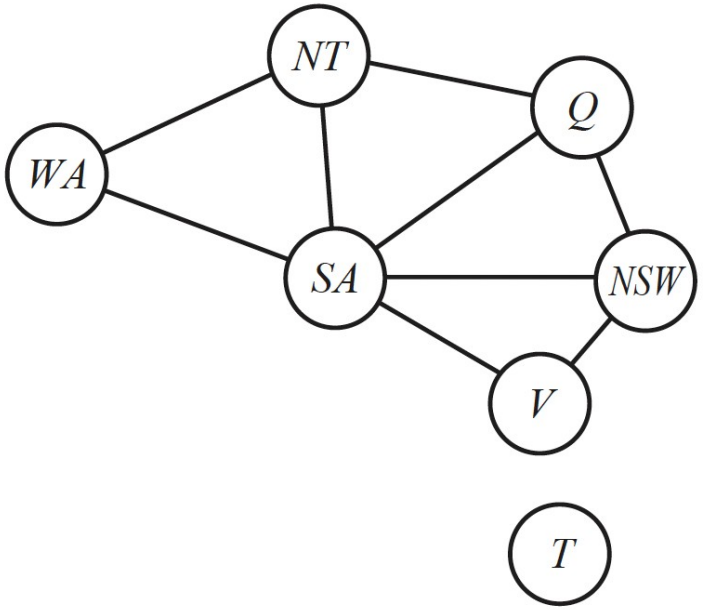
- **State space:** each state is defined by an *assignment* of values to some or all of the variables, e.g.,

- **Consistent assignment:** an assignment that does not violate any constraints

- **Complete (Partial) assignment:** an assignment where every variable is (a subset of variables are) assigned

- **Goal:** find a complete and consistent assignment
  - Find an assignment of the variables from their domains such that none of the constraints are violated

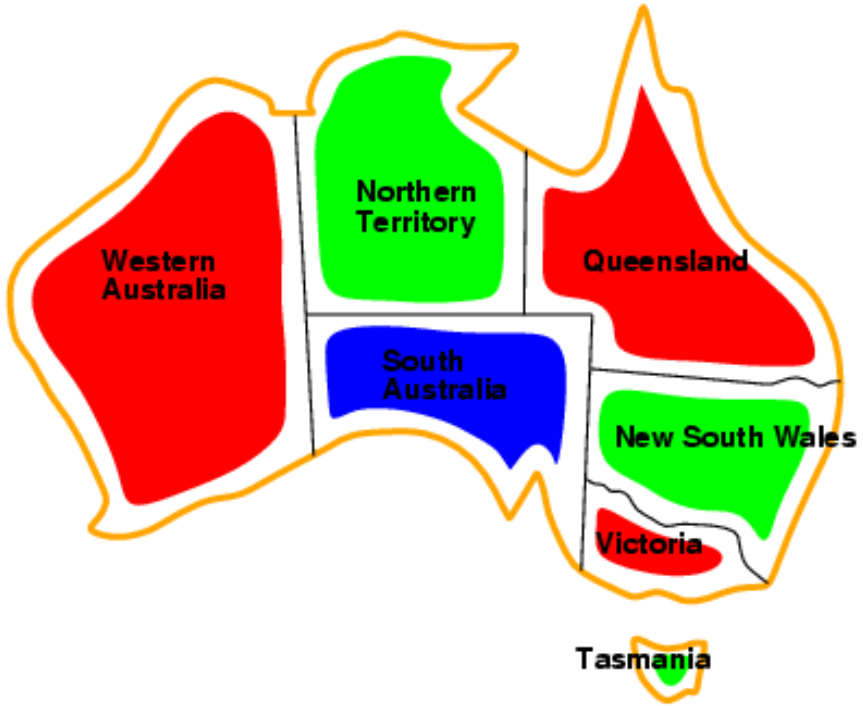
Example: Map Coloring



Constraint graph:
Node: variable
Edge: two variables in the same constraint

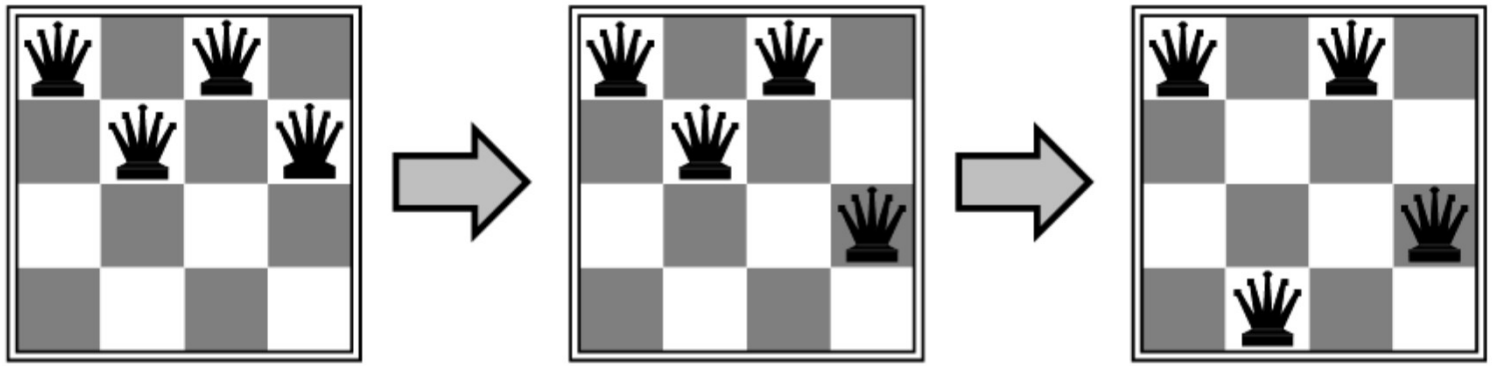
- **Task:** color each region either red, green, or blue such that no neighboring regions have the same color
- **Variables:**
- **Domains:**
- **Constraints:** adjacent regions must have different colors
  - or

Example: Map Coloring



- **Solution:** complete and consistent assignments that satisfy all constraints
  - Many possible solutions
  - e.g.,

Example: N-queens Problem



- Assume each queen is in one column, which row does each one go in?
- Variables: position of each queen in the column
- Domains: same (row value)
- Constraints:
  - Two queens cannot be in the same row:
  - Two queens cannot be in the same diagonal:

Example: Sudoku

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

- **Task:** fill in all the remaining squares with digits from 1 to 9 such that no digit appears twice in any row, column, or 3*3 box
- Variables: 81 variables, one for each square
  - A1, A2, A3,…, I7, I8, I9
  - Letter index rows, top to bottom; Digits index columns, left to right
- Domains: nine positive digits, e.g., A1
- Constraints: 27 *Alldiff* constraints, e.g., *Alldiff*{A1-A9}, *Alldiff*{A1-I1},…

Variations of CSP --- Type of Variables

- **Discrete variables**
  - Finite domains
    - Size complete assignments
    - E.g., map coloring, N-queens problem
  - Infinite domains
    - E.g., the set of integers
    - E.g., job-scheduling problem when we don't have a deadline
    - Need a constraint language without enumerating the available assignments, e.g,
    - Linear constraints: solvable
    - Nonlinear constraints: no general algorithm
- Continuous variables
  - E.g., circuit layout
  - Linear programming: constraints must be linear equalities or inequalities; can be solved in polynomial time

Variations of CSP --- Type of Constraints

- Unary constraint involves a single variable
  - e.g.,
- Binary constraint involves two variables
  - e.g.,
- High-order constraints involve 3 or more variables
  - Professors A, B, and C cannot be on a committee together
  - Can always be represented by multiple binary constraints
- Preference (soft) constraints indicate which solutions are preferred
  - Constraints can be violated, but with a higher cost on individual assignments
  - E.g., class-scheduling problem, Prof. A prefers teaching in morning; assigning an afternoon slot may be allowable, but can lead to a higher cost for the overall objective
  - Combination of optimization with CSPs

CSPs Only Need Binary Constraints

- Unary constraint: just change the domain by deleting the values
- Higher order: reduce to binary constraints
- Simple example:
  - Three variables: X, Y, Z
  - Domains:
  - Constraints:

  - Create an auxiliary variable, W, taking values as 3-tuples
  - Domain:
    - Exactly the tuples satisfying the constraints
  - Create three new constraints

How to Solve CSPs

- Example: integer CSP
- Variables A, B, C, D which take values in {1,…,10}
- Constraints:
  - C1:
  - C2:
  - C3:
- Solution:

Brute Force Search

- For A=1,…,10
  - For B=1, …, 10
    - For C=1, …, 10
      - For D=1, …, 10
        - If C1 and C2 and C3
          - WriteSolution(A, B, C, D)

How to reduce the complexity of this search?

Look into the Constraints

- Constraints:
  - C1:
  - C2:
  - C3:
- A more efficient loop:
  - For A=2,…,10
    - For B=1, …, A-1
      - For C=1, …, A-1
        - For D=1, …, A-1
          - If C1 and C2 and C3
            - WriteSolution(A, B, C, D)

We Can Do More…

- Constraints:
  - C1:
  - C2:
  - C3:
- **Variable elimination: Remove variable by replacing it by**
- New constraints with 3 variables:
  - C1’:
  - C2’:

- Find valid assignments for (B, C, D), and set A=B+C

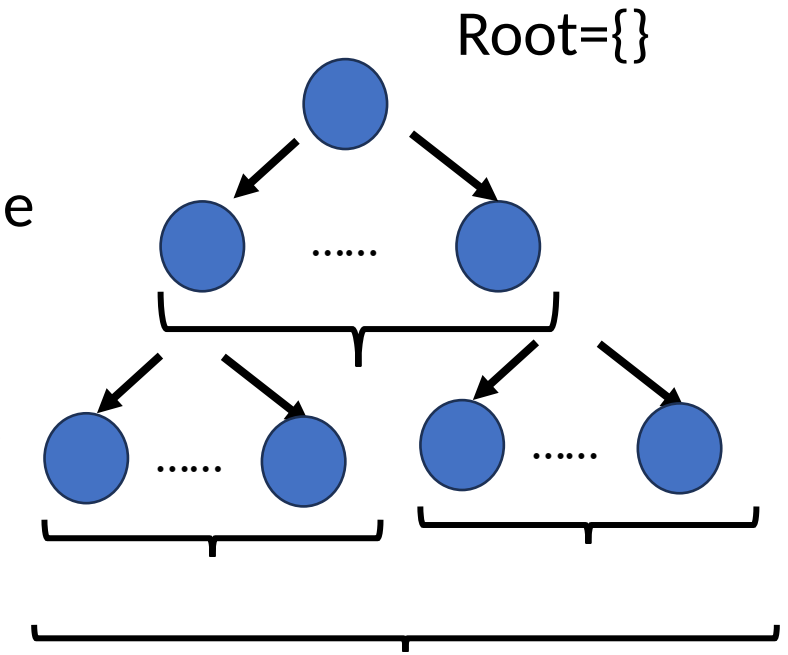
CSP as A Standard Search Problem

- A CSP can easily be expressed as a standard search problem

- States defined by the values assigned so far (partial assignments)
  - Initial state: the empty assignment
  - Successor function: assign a value to an unassigned variable provided that it does not violate a constraint
  - Goal test: the current assignment is complete and satisfies all the constraints
  - Path cost: not really relevant

CSP as A Standard Search Problem

- For a CSP with variables of domain size
- Solution can be found at depth

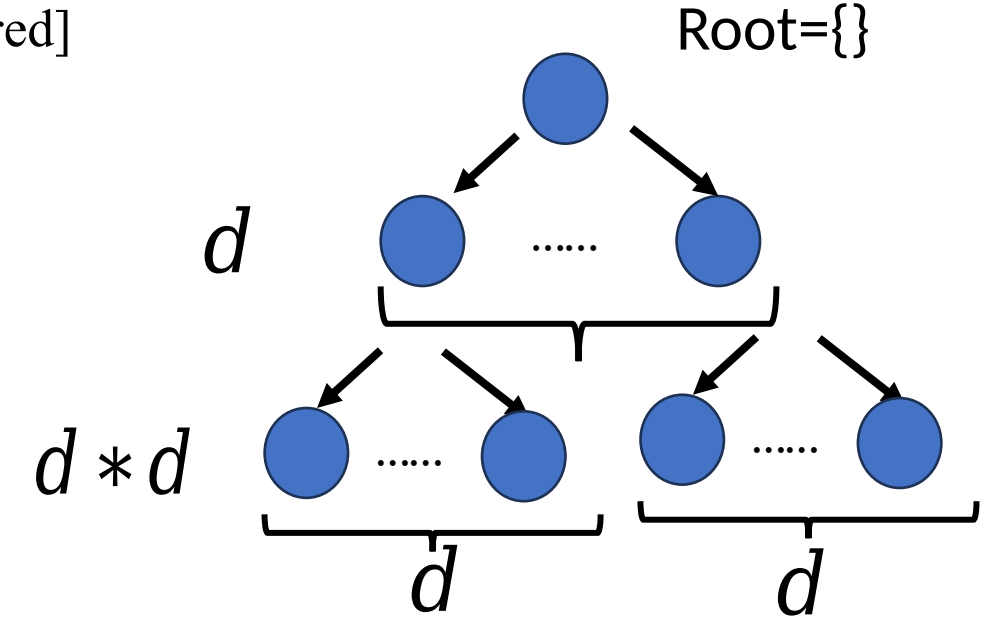


- End up with leaves at the *n-th* level
- But there are only possible complete assignments

What is missing

- Commutativity:
  - A problem is commutative if the order of any given set of actions has no effect on the outcome
  - CSPs are commutative: can reach the same partial assignment regardless of order
  - Example: map coloring for Australia
    - [WA=red then NT=green] same as [NT=green then WA=red]

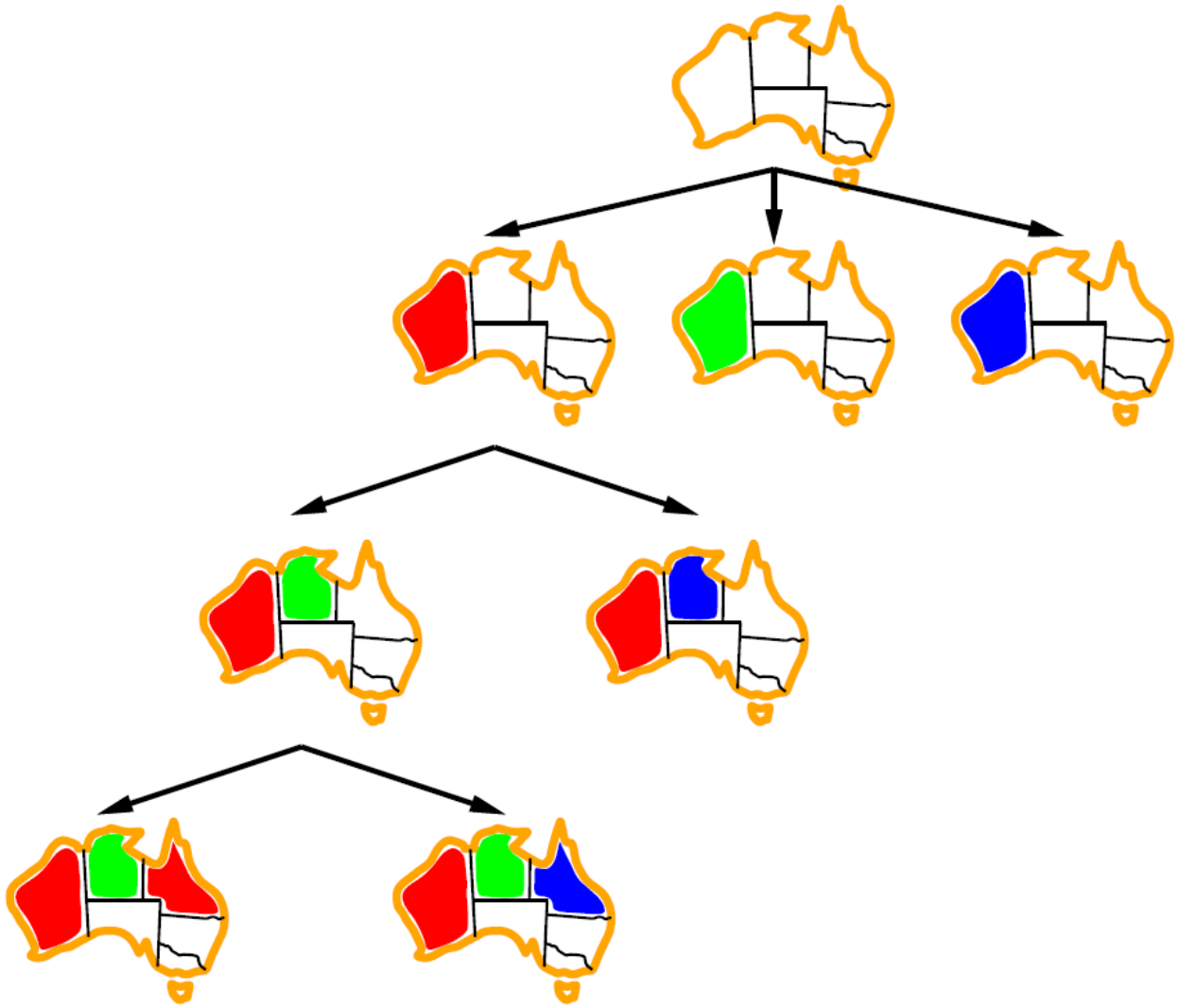
- In CSP, we need only consider a single variable at each node in the search tree
  - Might choose among SA=red, SA=green, SA=blue
  - Never choose between SA=red and WA=blue
- There are leaves
- (still need to figure out which variable to choose at each node)

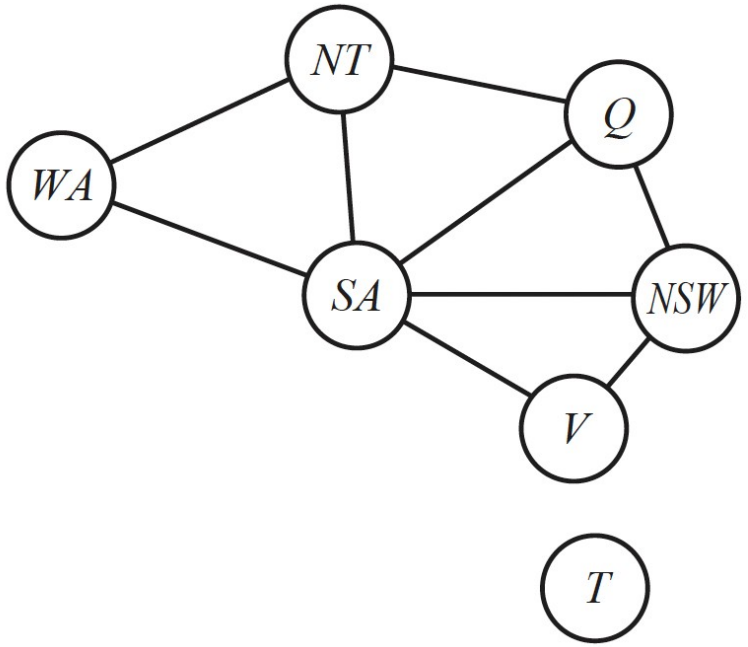


Backtracking

- Basic uninformed search algorithm to solve CSPs
- Similar to depth-first search, exploring one node at a time
- Chooses values for one variable at a time
- Backtracks when a variable has no legal values left to assign

Example





Backtracking Search

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    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) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            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

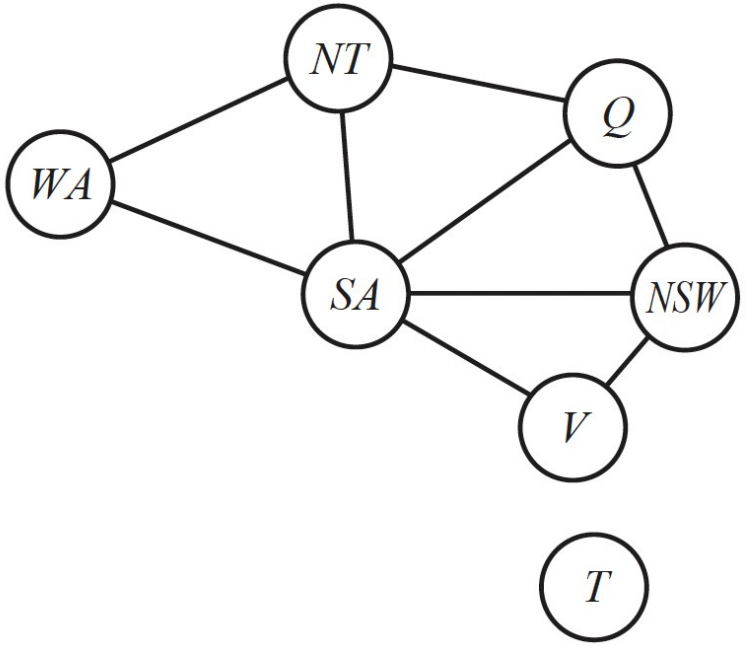
- Previous improvements on uninformed search
  - Use heuristics (domain knowledge): informed search

- General-purpose ideas can achieve huge gains in speed for backtracking

- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?

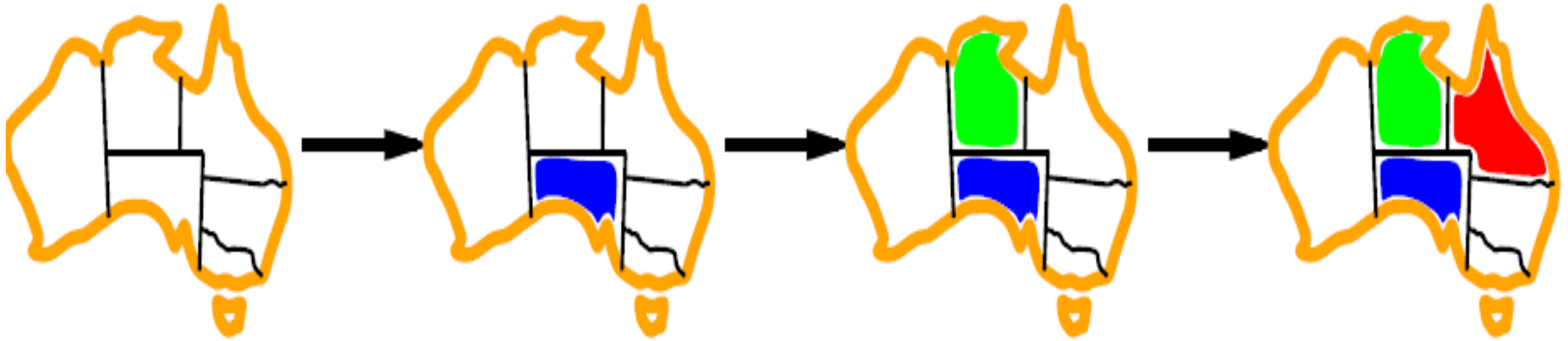
- Filtering: can we detect inevitable failure early?

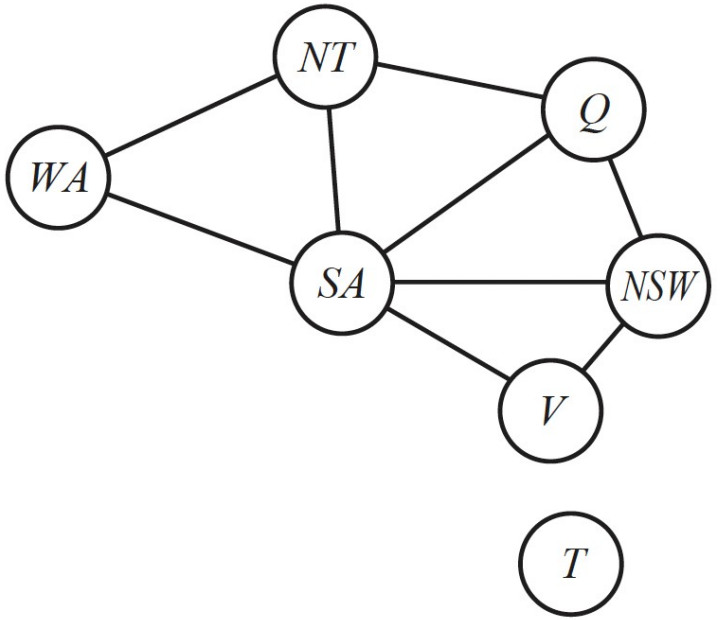
Variable Ordering: Minimum Remaining Values (MRV)



- Aka “most constrained variable” or “fail-first”:
  - Choose the variable with the fewest legal left values in its domain
  - If some variable X has no legal values left, select X and failure will be detected immediately --- avoiding pointless searches through other variables

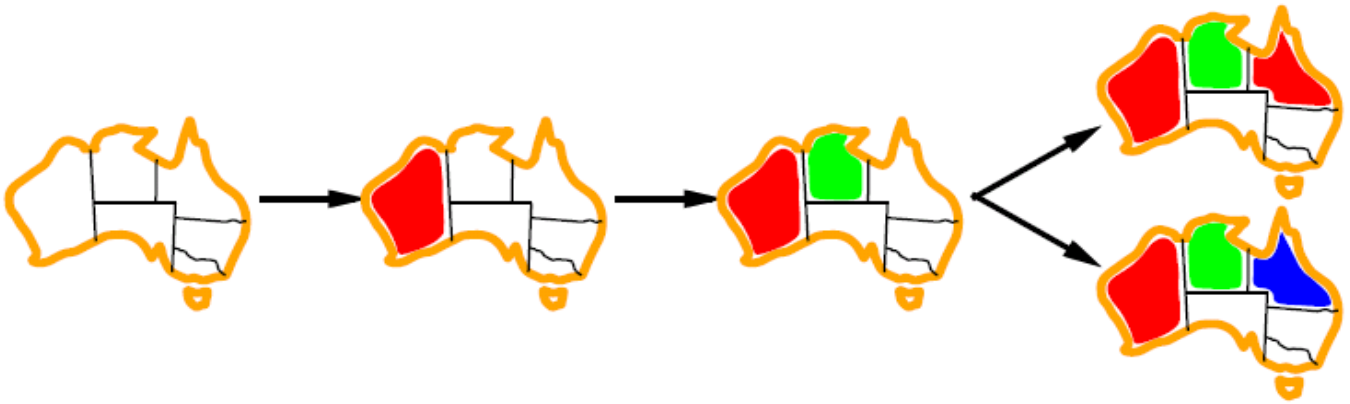
Degree Heuristic

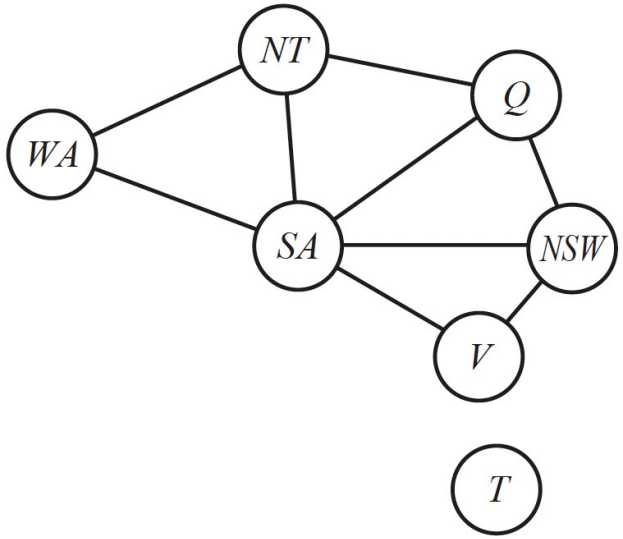




- MRV doesn't help in choosing the first region to color
- Degree heuristic: select variable that is involved in the largest number of constraints on other unassigned variables
- MRV is usually a more powerful guide, but degree heuristic can be useful as a tie breaker

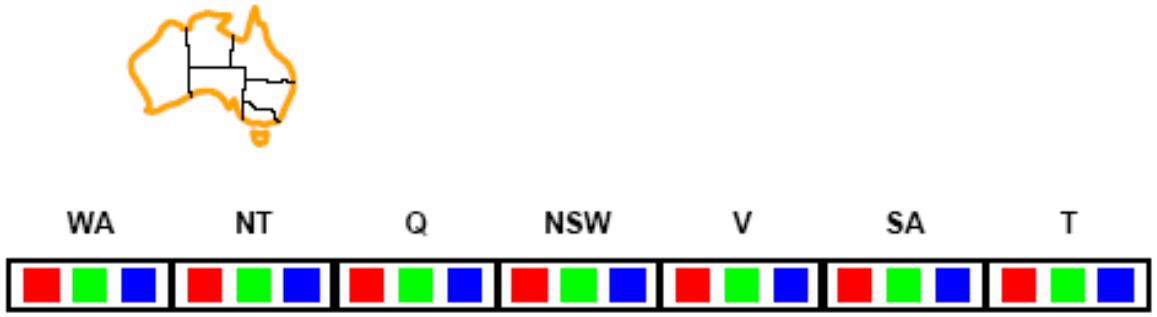
Value Ordering: Least Constraining Value

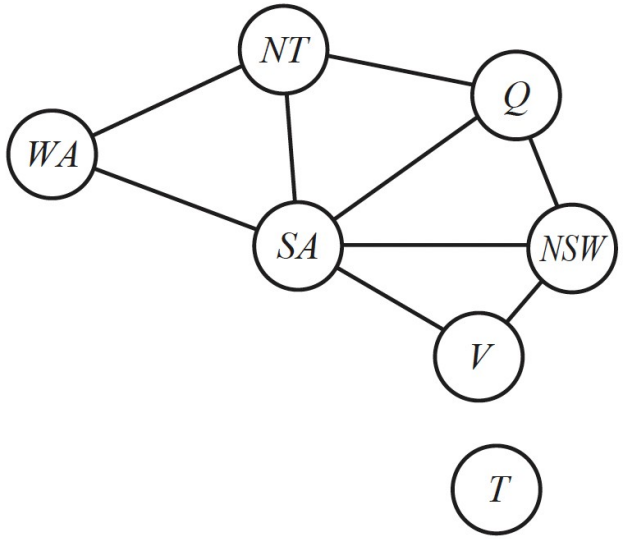




- Given a variable, choose the least constraining value
  - Prefer the value that rules out the fewest choices for the neighboring variables in the constraint graph
  - Leave the maximum flexibility for subsequent variable assignments
- Why least rather than most?
  - Only need one solution, look for the most likely values first

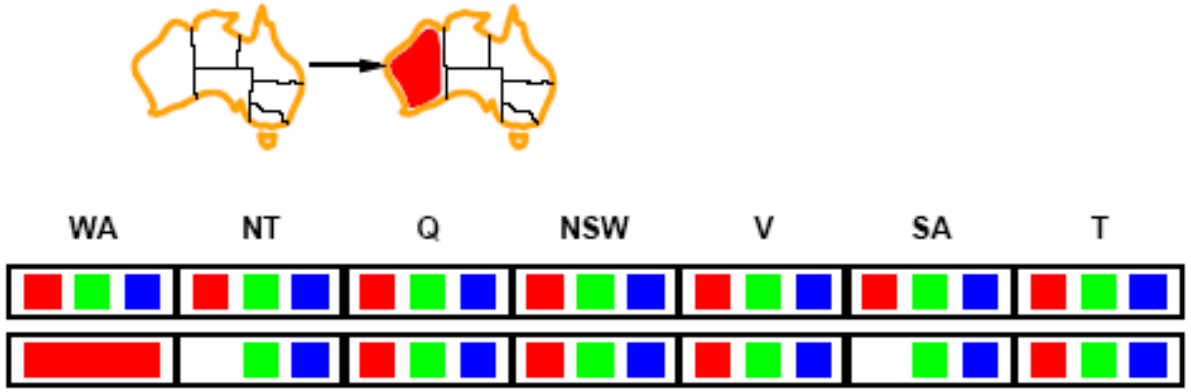
Filtering: Forward Checking

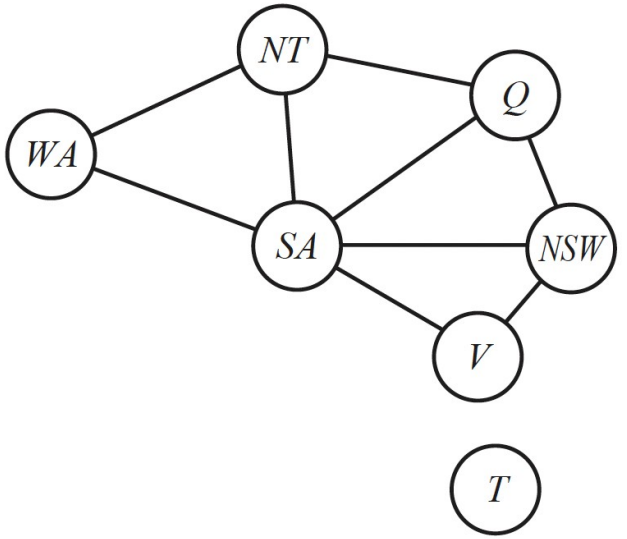




- Keep track of remaining legal values for unassigned variables and cross off bad options
- Terminate search when any variable has no legal values

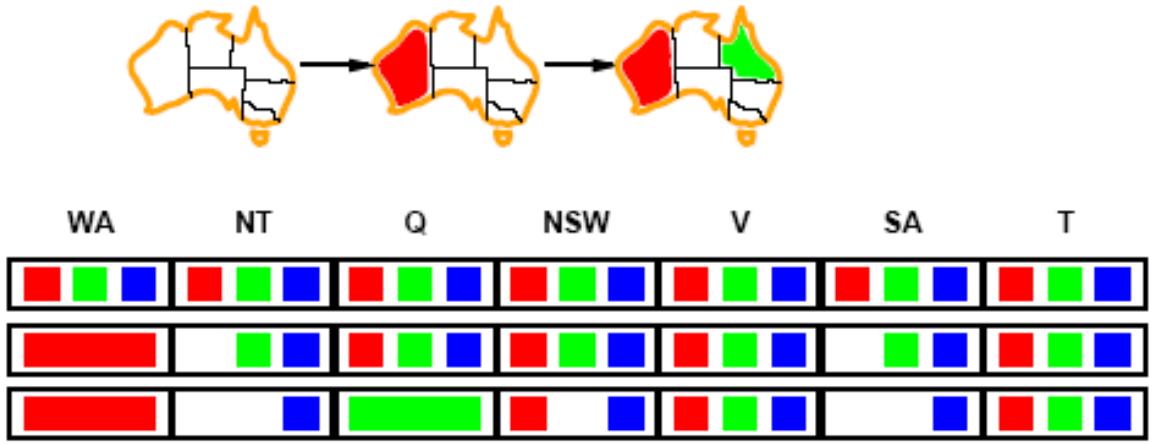
Filtering: Forward Checking

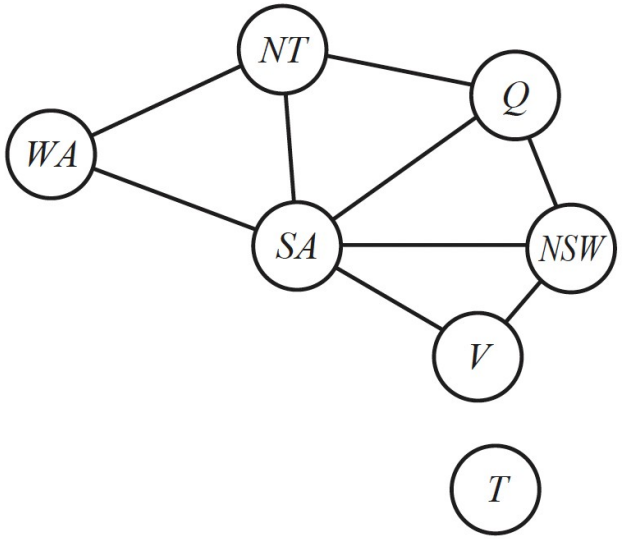




- Assign {WA=red}
- Effects on other variables connected by constraints with WA
  - NT can no longer be red
  - SA can no longer be red

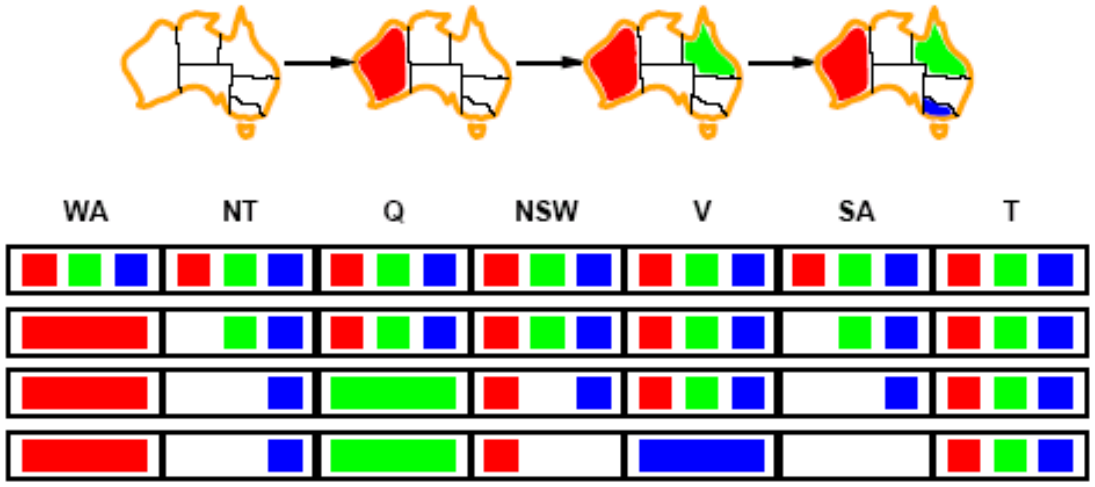
Filtering: Forward Checking

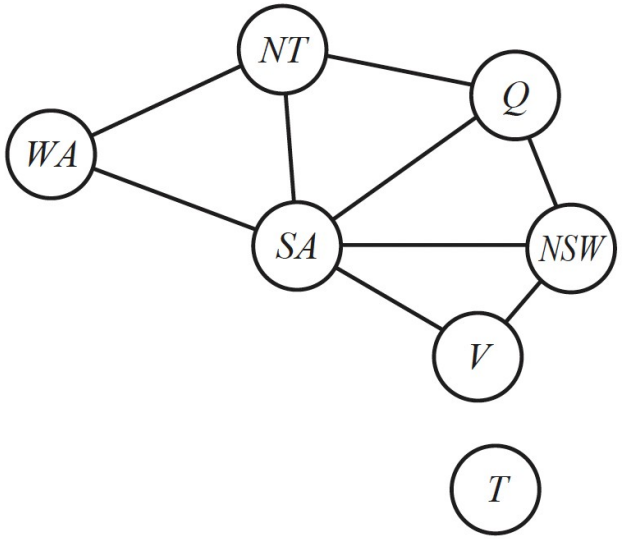




- Assign {Q=green}
- Effects on other variables connected by constraints with Q
  - NT can no longer be green
  - NSW can no longer be green
  - SA can no longer be green
- MRV can also be used for variable ordering

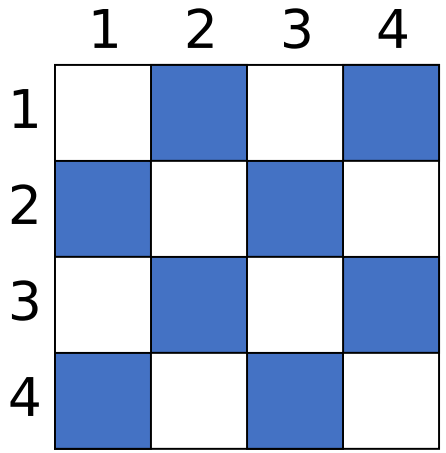
Filtering: Forward Checking

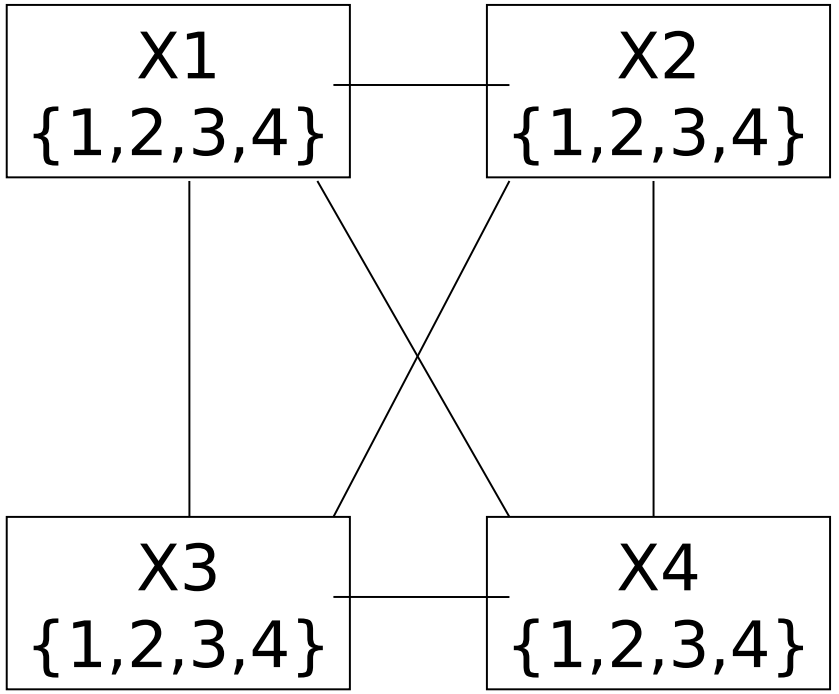




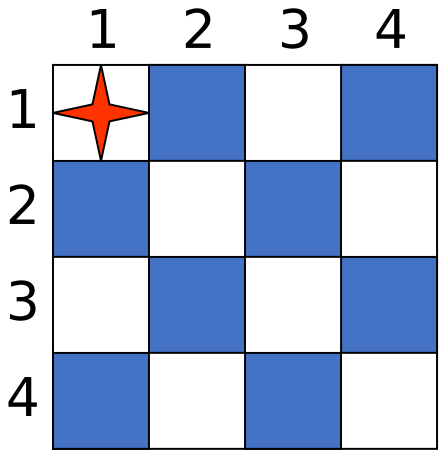
- If assign {V=blue}
- Effects on other variables connected by constraints with V
  - NSW can no longer be blue
  - SA is empty
- Forward checking detected that the partial assignment is inconsistent with the constraints and backtracking can occur

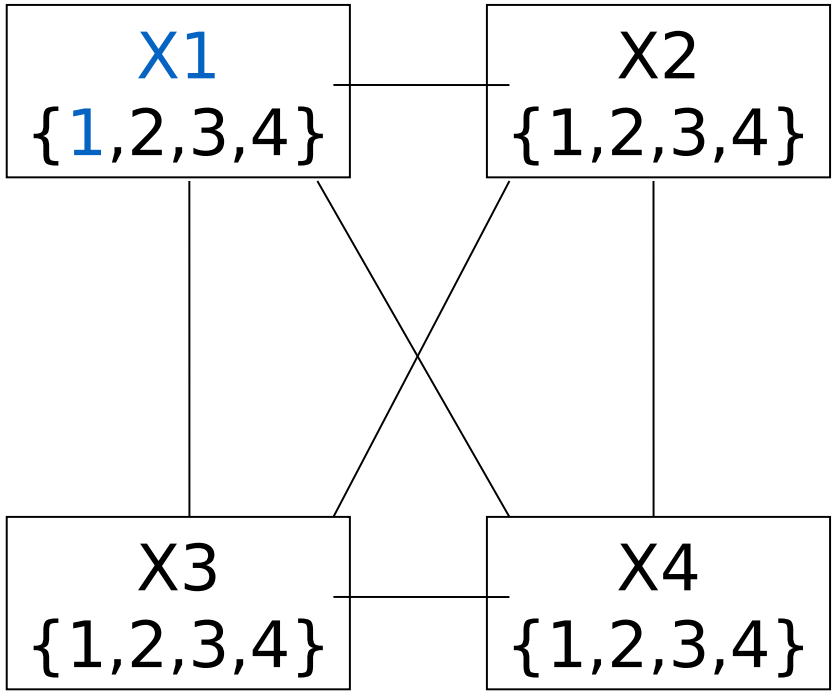
Example: 4-queens



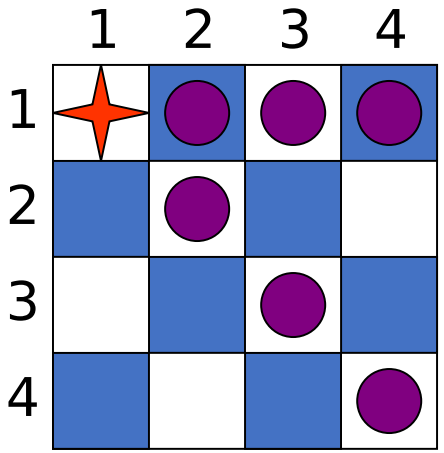


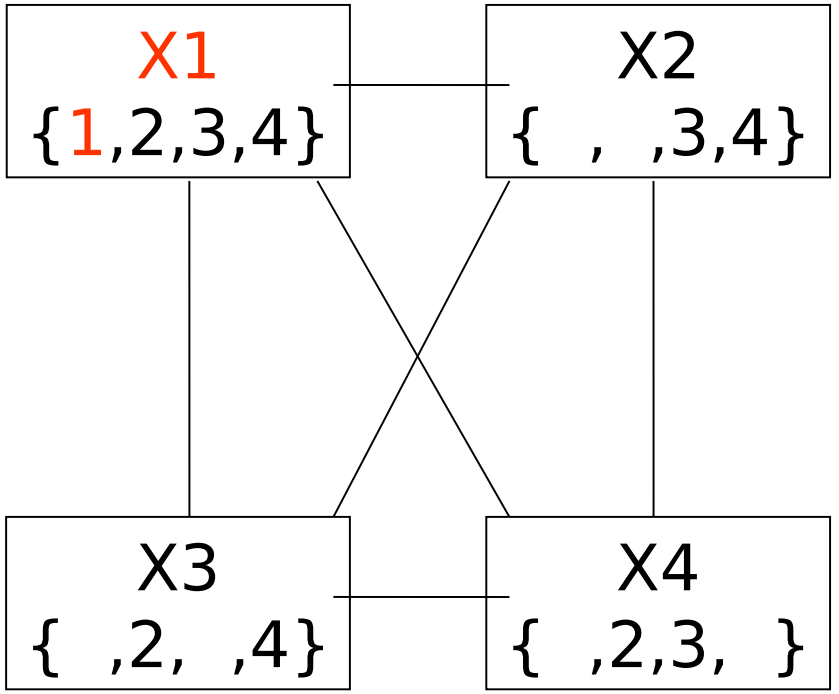
Example: 4-queens



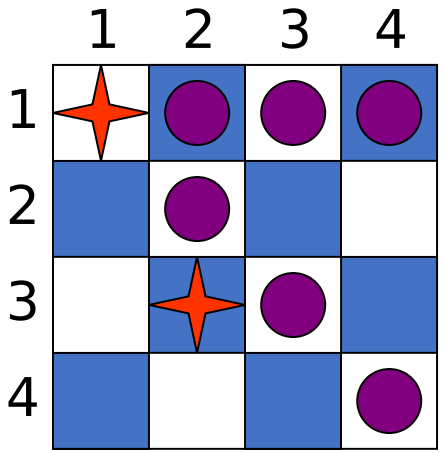


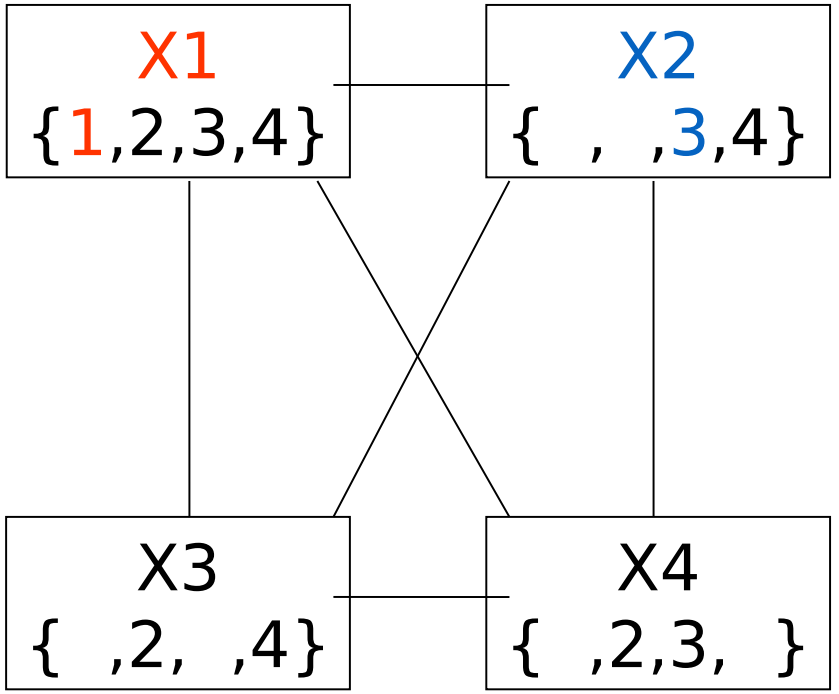
Example: 4-queens



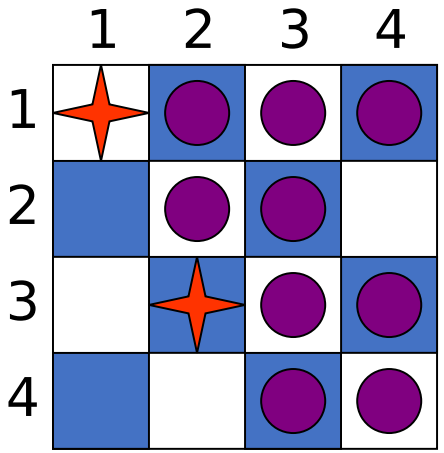


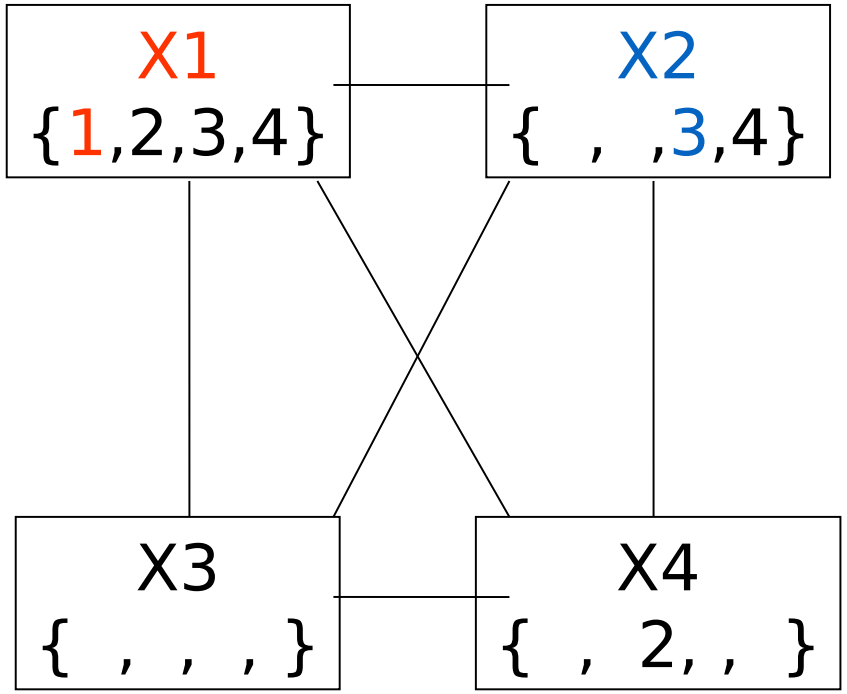
Example: 4-queens



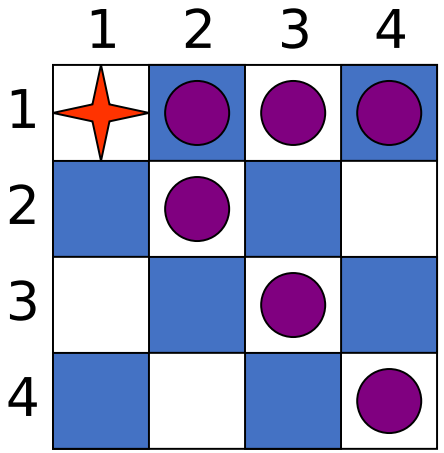


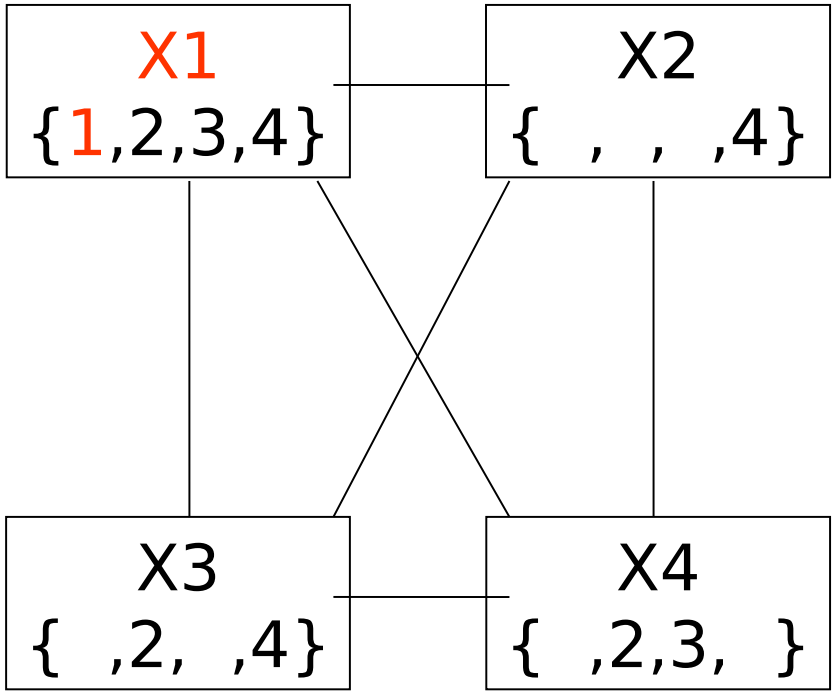
Example: 4-queens



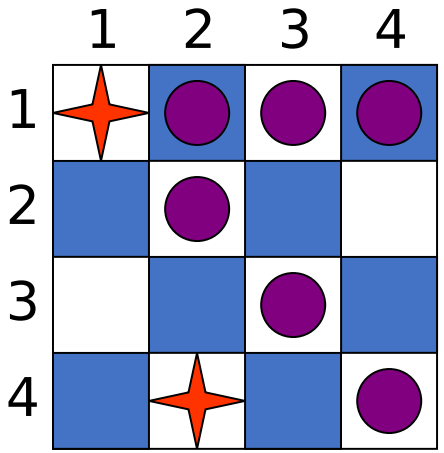


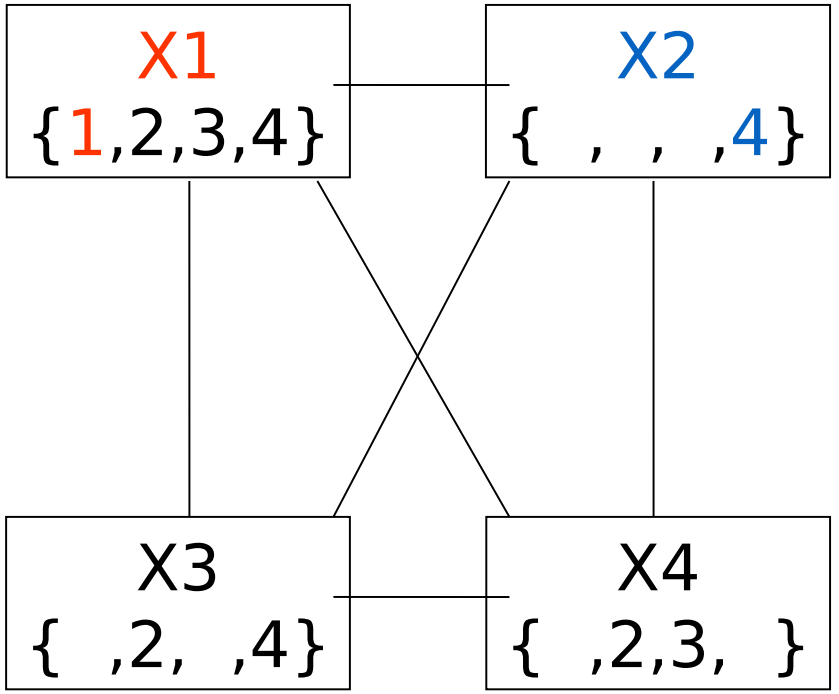
Example: 4-queens



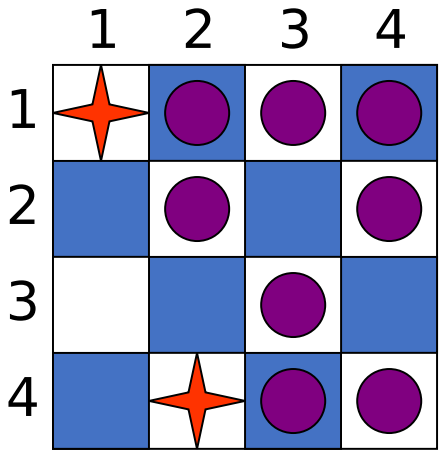


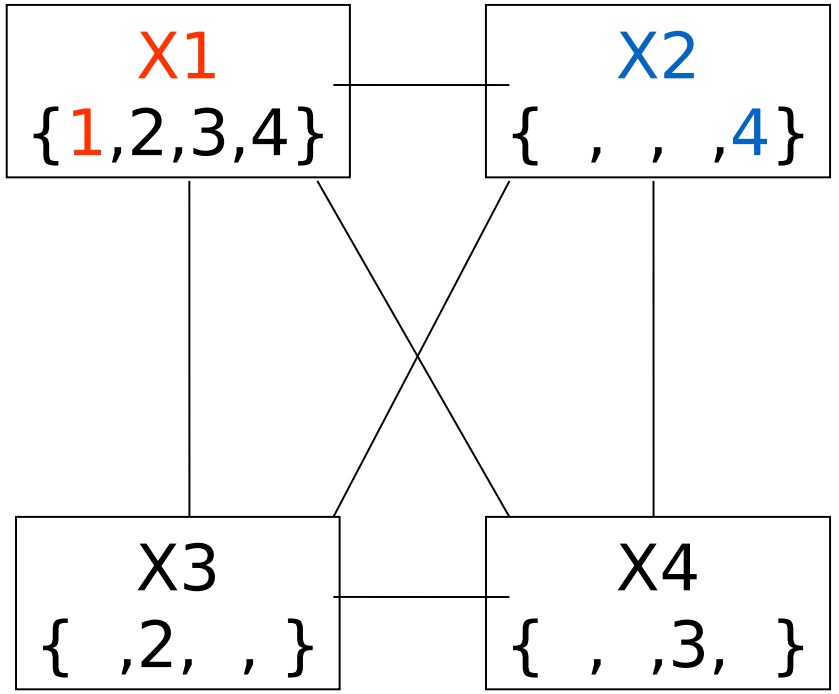
Example: 4-queens



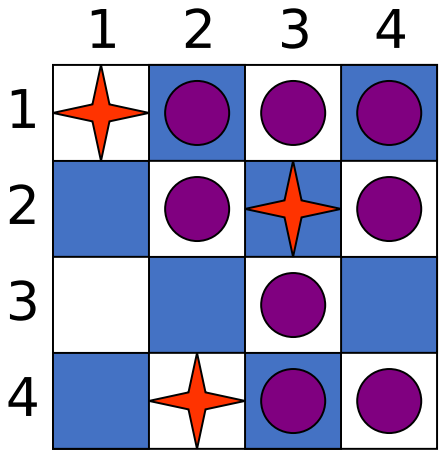


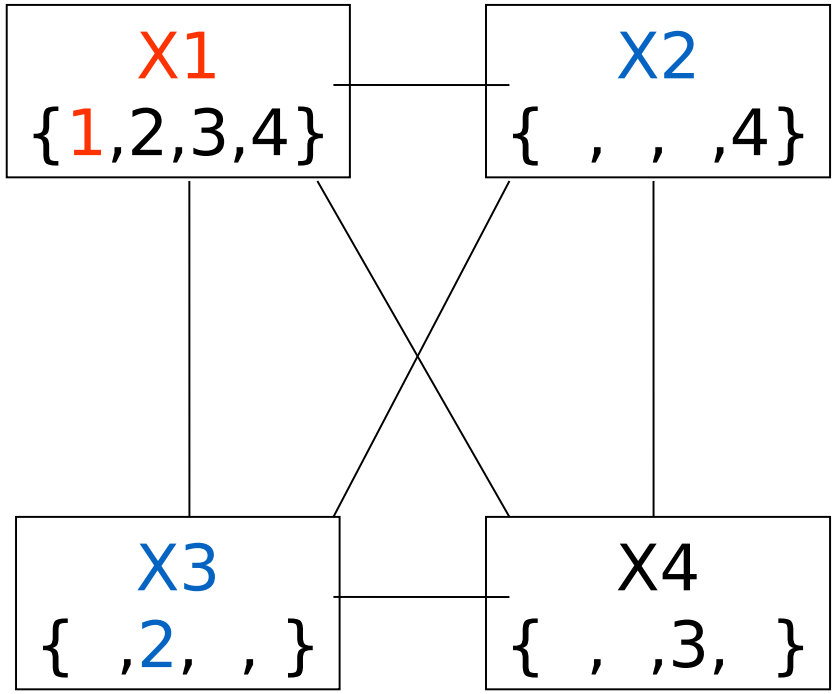
Example: 4-queens



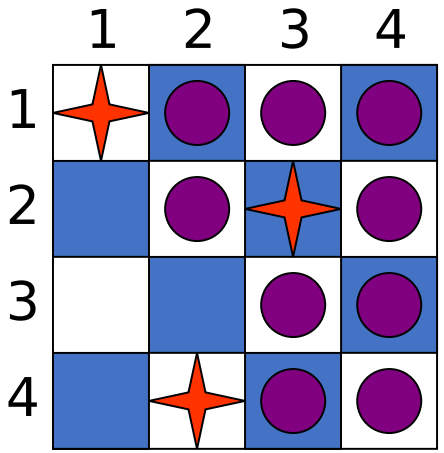


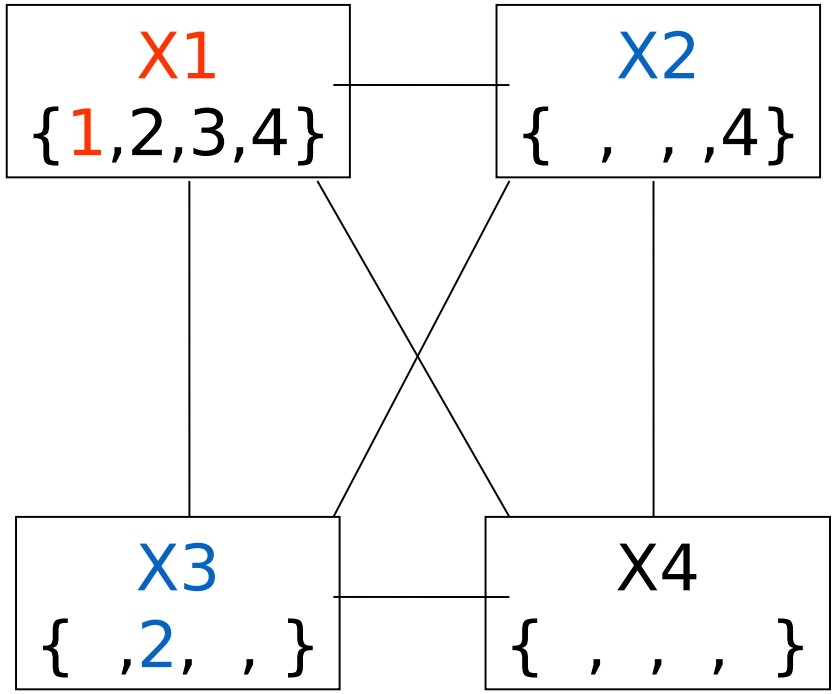
Example: 4-queens



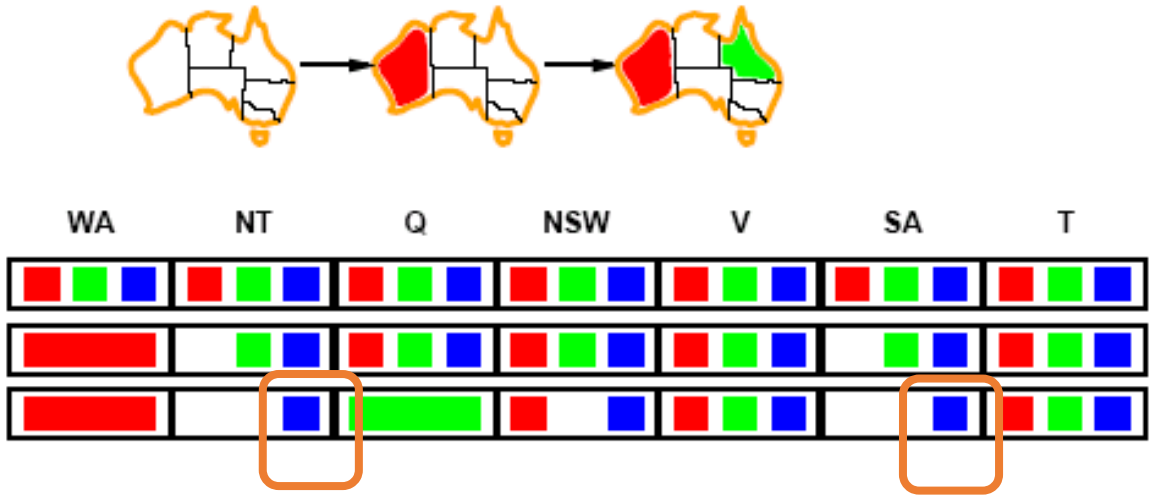


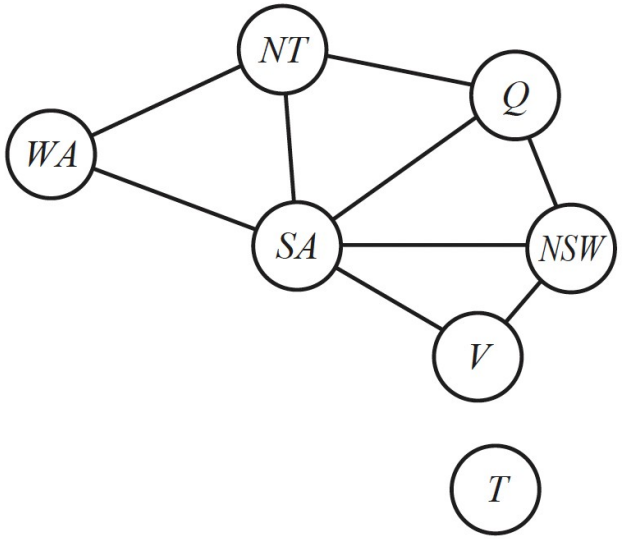
Example: 4-queens





Filtering: Constraint Propagation





- Forward checking propagates information from assigned to unassigned variables, but doesn't detect all failures
- NT and SA cannot both be blue
- Why didn't we detect this yet?

Constraint Propagation

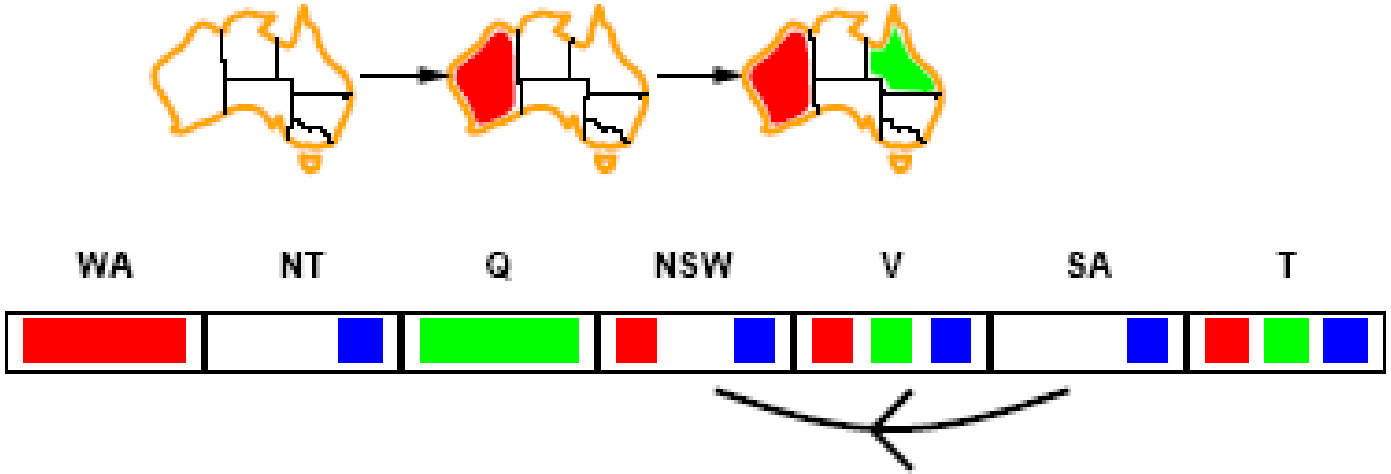
- Constraint propagation: a specific type of inference
  - Using constraints to reduce the domain for a variable, which in turn can reduce the domain for another variable, and so on…
  - Reason from constraint to constraint

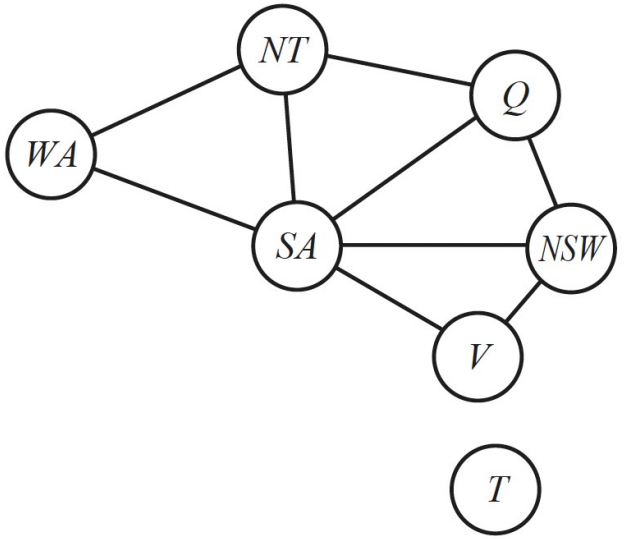
- Techniques like constraint propagation and forward checking are in effect eliminating parts of the search space
  - Somewhat complementary to search

- Constraint propagation goes further than FC by repeatedly enforcing constraints locally

- Arc-consistency (AC) is a systematic procedure for constraint propagation

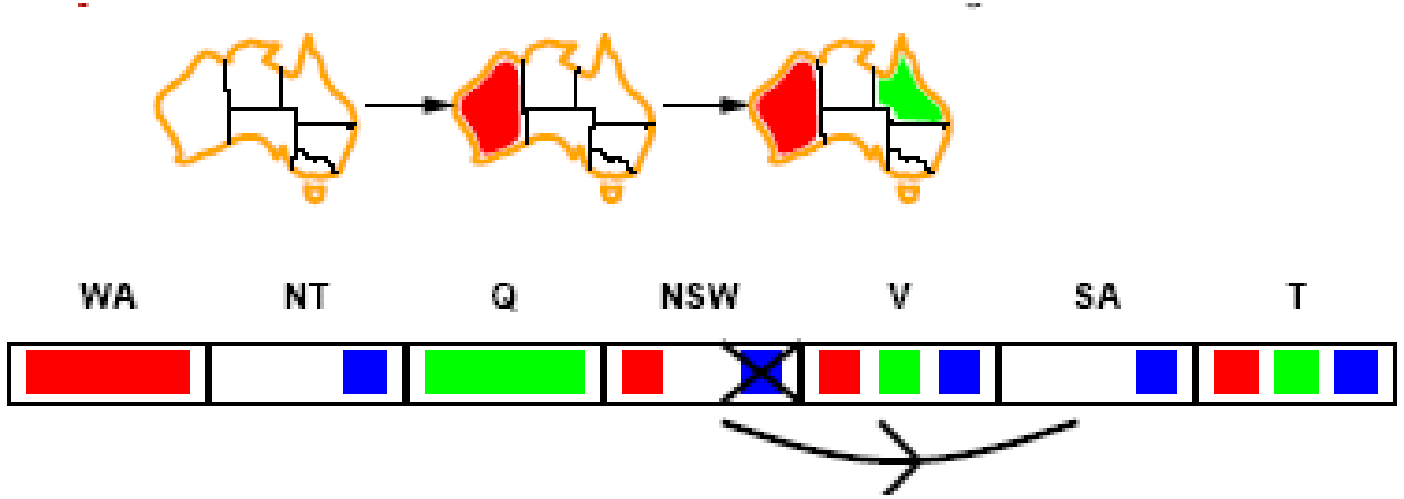
Arc Consistency

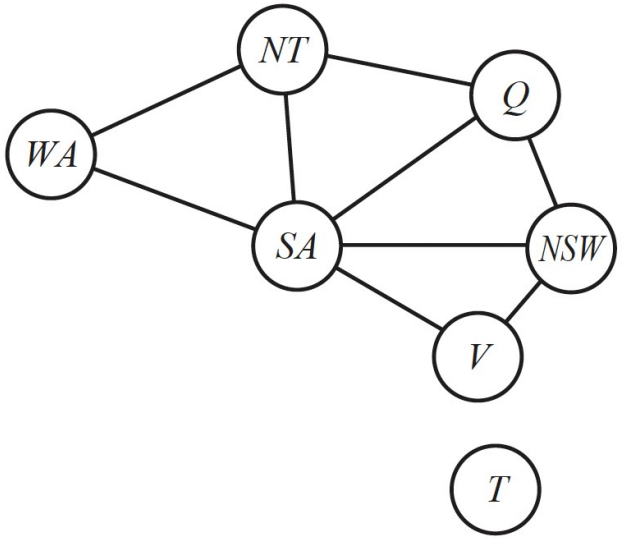




- An arc is consistent if for every value x of X there is some value y of Y consistent with x (without violating the constraint)
- Consider state of search after WA and Q are assigned
- SA NSW is consistent if
  - SA=blue and NSW=red

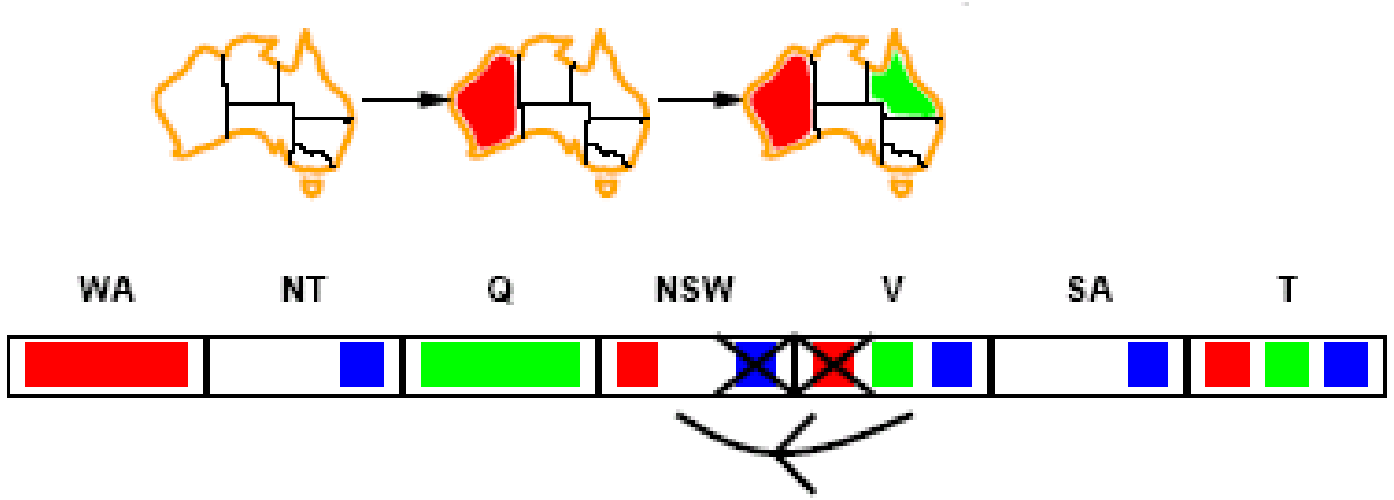
Arc Consistency

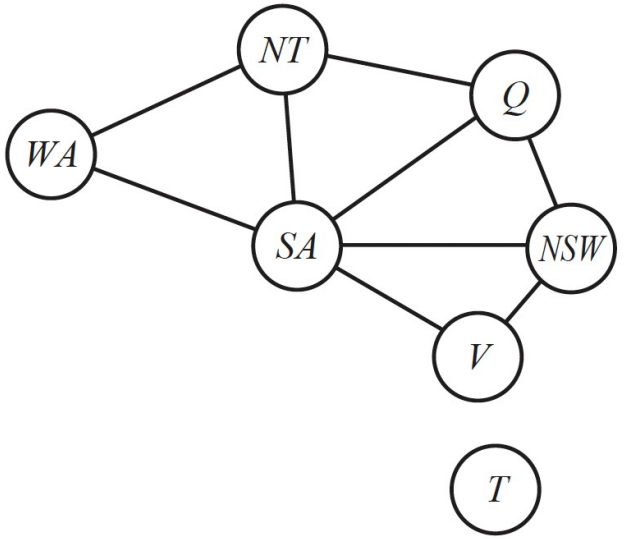




- NSW SA is consistent if
  - NSW=red and SA=blue
  - NSW=blue and SA=???
- Enforcing arc consistency removes blue from NSW

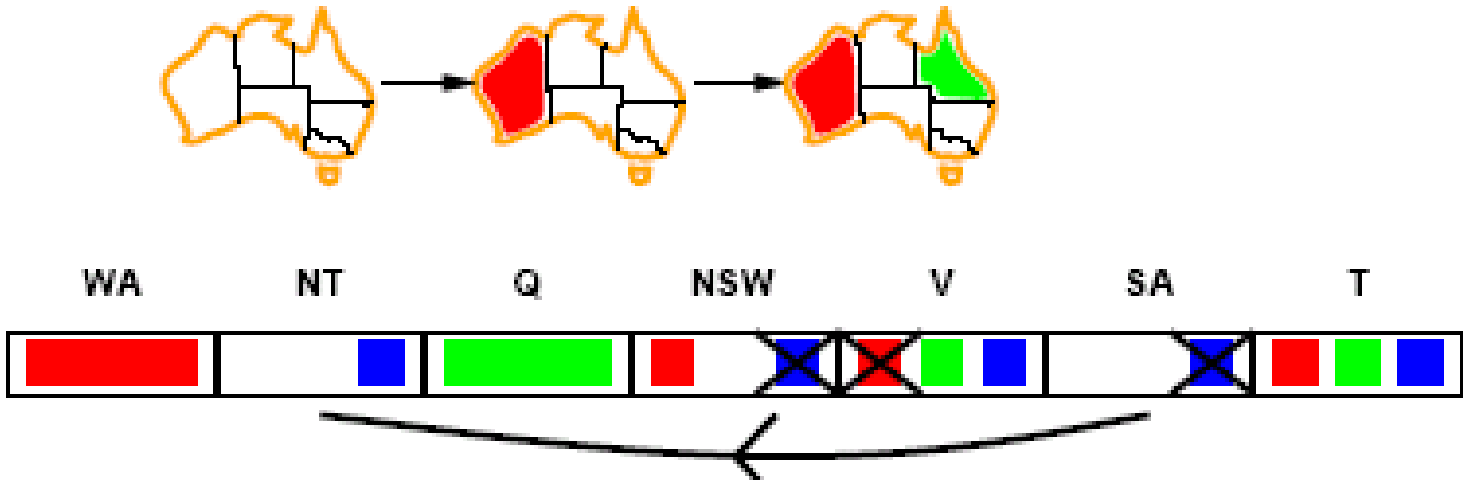
Arc Consistency

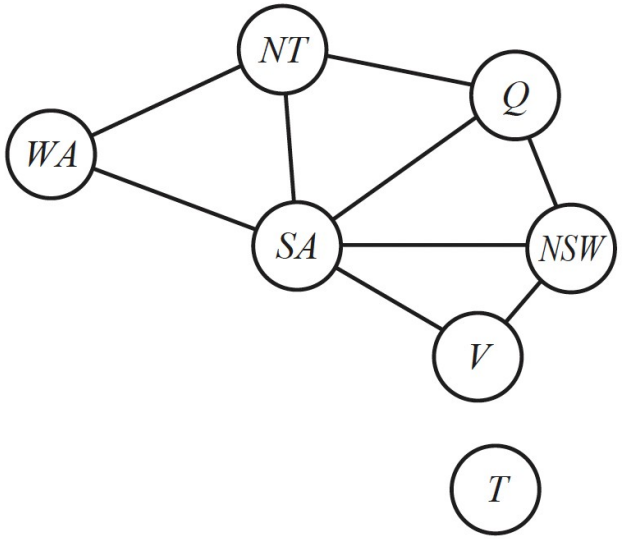




- Continue to propagate constraints
  - Check V NSW
  - Not consistent for V=red
  - Remove red from V

Arc Consistency





- Continue to propagate constraints
  - Check SA NT
  - Not consistent for any value of SA
  - Detect failure
- Arc consistency detects failure earlier than forward checking

Arc Consistency Checking

- Can be run as a preprocessor or after each assignment
- AC must be run repeatedly until no inconsistency remains
- Trade-off
  - Requires some overhead to do, but generally more effective than direct search
  - Eliminates large inconsistent parts of the state space more effectively than search can
- Need a systematic method for arc-checking
  - If X loses a value, neighbors of X need to be rechecked
  - Incoming arcs can become inconsistent again (outgoing arcs stay the same)

Arc Consistency Checking

- Can be run as a preprocessor or after each assignment
- AC must be run repeatedly until no inconsistency remains
- Trade-off
  - Requires some overhead to do, but generally more effective than direct search
  - Eliminates large inconsistent parts of the state space more effectively than search can
- Need a systematic method for arc-checking
  - If X loses a value, neighbors of X need to be rechecked
  - Incoming arcs can become inconsistent again (outgoing arcs stay the same)

AC-3 Algorithm

function AC-3(csp) **returns** false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components $(X,\ D,\ C)$
  local variables: $queue$, a queue of arcs, initially all the arcs in csp

  while $queue$ is not empty **do**
    $(X_i,\ X_j) \leftarrow$ REMOVE-FIRST($queue$)
    if REVISE(csp, X_i, X_j) **then**
      if size of $D_i\ =\ 0$ **then return** $false$
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add $(X_k,\ X_i)$ to $queue$
  return $true$

---

function REVISE(csp, X_i, X_j) **returns** true iff we revise the domain of X_i
  $revised \leftarrow false$
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      $revised \leftarrow true$
  return $revised$

Complexity of AC-3

- A binary CSP with n variables has at most arcs
- Each arc can be inserted in the queue d times (worst case)
  - Only d values to delete
- Consistency of an arc can be check in time
  - pairs of values to check
- Complexity is
- Although more expensive than forward checking, AC is usually worthwhile

Local Search for CSPs

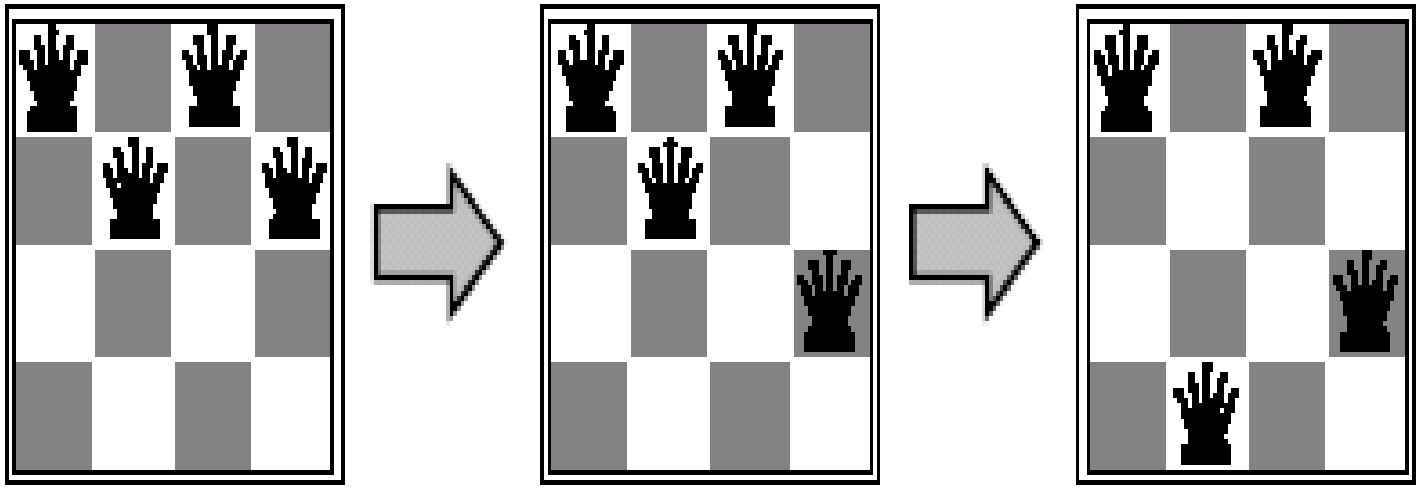
- Use complete-state representation
  - Initial state: assign a value to every variable
  - Successor function: change the value of one variable at a time
- Allow states with unsatisfied constraints (unlike backtracking)
- The point of local search is to eliminate the violated constraints
- Variable selection: randomly select any conflicted variable
- Value selection: min-conflicts heuristic
  - Select new value that results in the minimum number of conflicts with other variables

Local Search for CSPs

function MIN-CONFLICTS(csp, max_steps) **returns** a solution or failure
  inputs: csp, a constraint satisfaction problem
    max_steps, the number of steps allowed before giving up

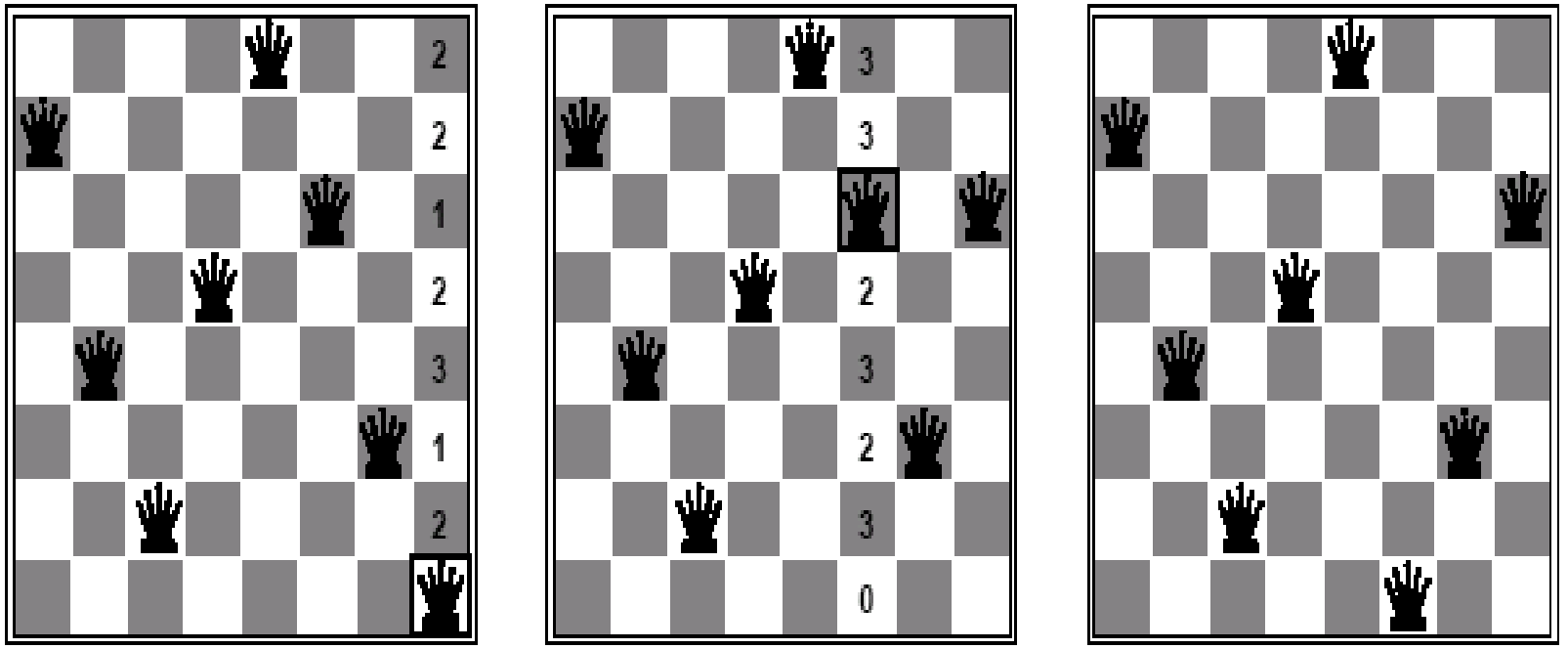
  $current \leftarrow$ an initial complete assignment for csp
  for i = 1 to max_steps **do**
    if $current$ is a solution for csp **then return** $current$
    $var \leftarrow$ a randomly chosen conflicted variable from csp.VARIABLES
    $value \leftarrow$ the value v for var that minimizes CONFLICTS(var, v, $current$, csp)
    set $var = value$ in $current$
  return *failure*

Example 1



- Use min-conflicts heuristic
  - Randomly select the 4th variable (position of the 4th queen)
  - X=1, two constraints violated
  - X=3, no constraints violated
  - X=4, two constraints violated

Example 2



- A two-step solution for the 8-queens problem
- At each stage, a queen is chosen for reassignment in its column
- The algorithm moves the queen to the min-conflicts square, breaking ties randomly

Advantages of Local Search

- The runtime of min-conflicts is roughly independent of problem size
  - Can solve the millions-queen problem in roughly 50 steps
  - Why?
    - N-queens is easy for local search because of the relatively high density of solutions in state-space

- Local search can be particularly useful in an online setting
  - Airline schedule problem
    - E.g., mechanical problems require that 1 plane is taken out of service
    - Can locally search for another “close” solution in state-space
    - Much better and faster in practice than finding an entirely new schedule

Summary

- CSPs:
  - Special kind of problem: states defined by values of a fixed set of variables, goal test defined by constraints on variable values
- Backtracking = depth-first search with one variable assigned per node
- Heuristics:
  - Variable ordering and value selection help significantly
- Constraint propagation does additional work to constrain values and detect inconsistencies
  - Works effectively when combined with heuristics
- Local search like iterative min-conflicts is often effective in practice