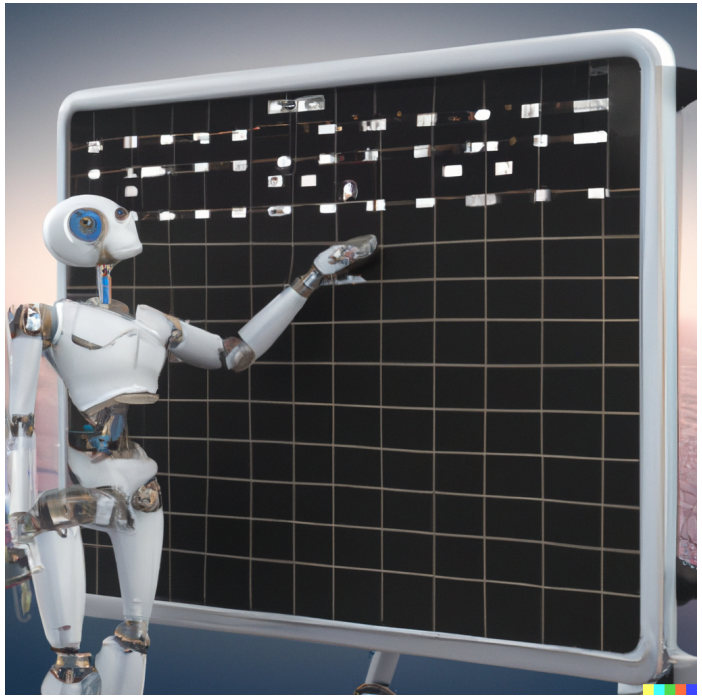
Lecture 6 - Constraint Satisfaction Problems

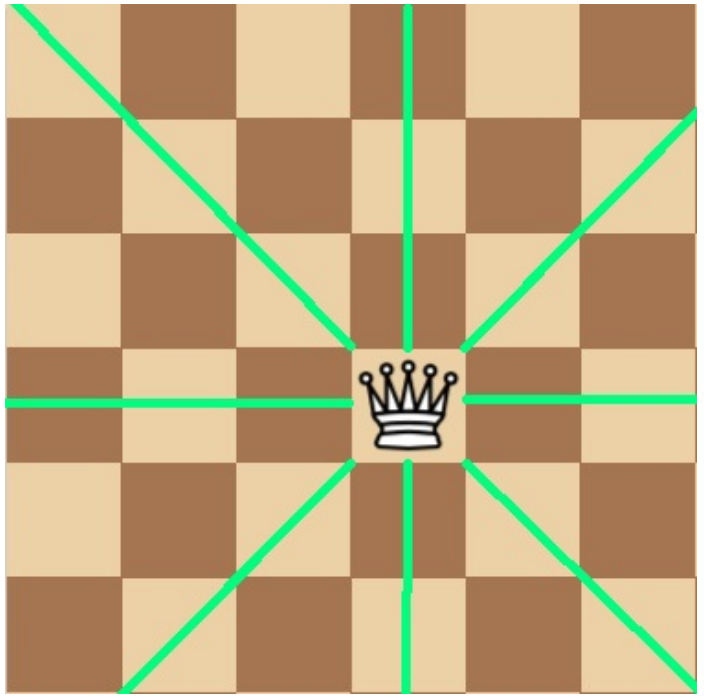
Gleb Sizov
Norwegian University of Science and Technology



What is Constraint Satisfaction Problem (CSP)?

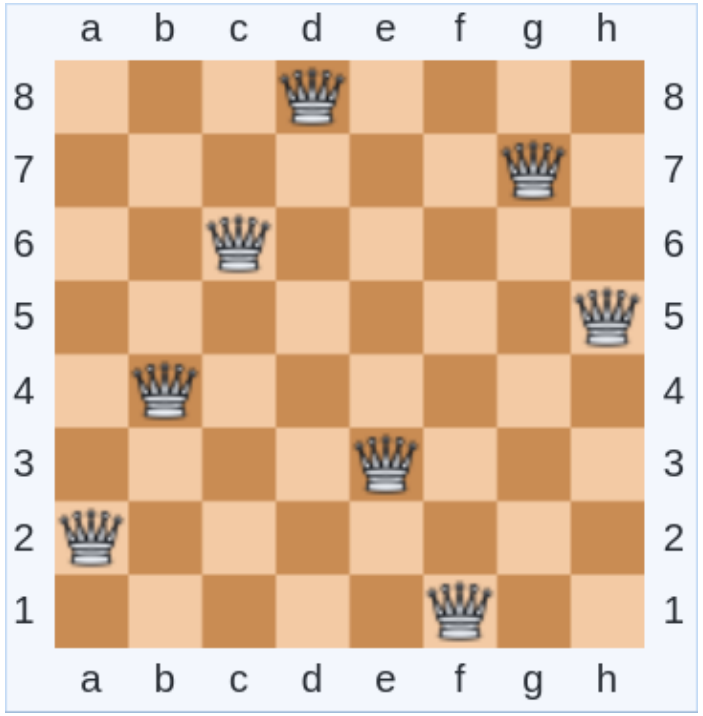
Example: N-Queens problem

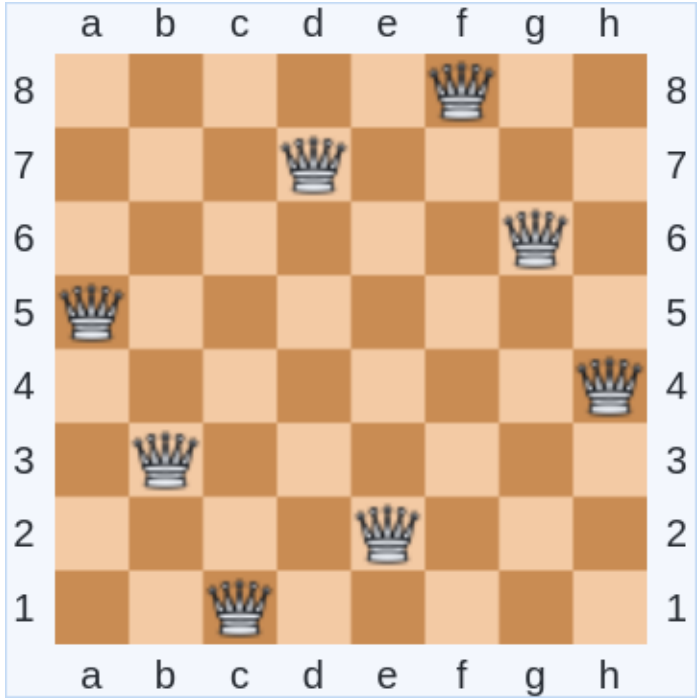
Place N queens on an NxN chess board with the **constraint** that they don’t attack each other.



N-Queen solutions

8-queens: 92 solutions





N-Queen problems
Try every configuration?

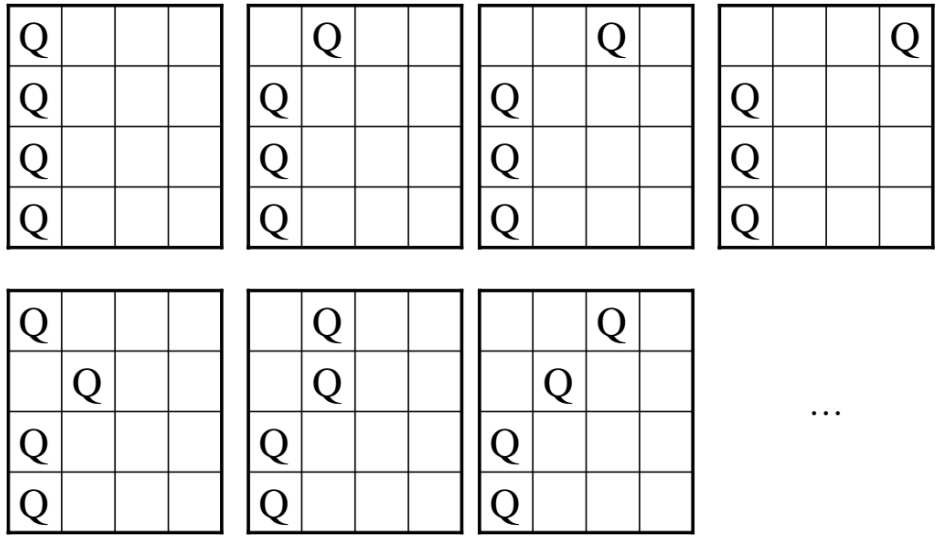
Number of configurations:

4-queens: 256

8-queens: 16,777,216

16-queens: 18,446,744,073,709,551,616

N-queens: N^N

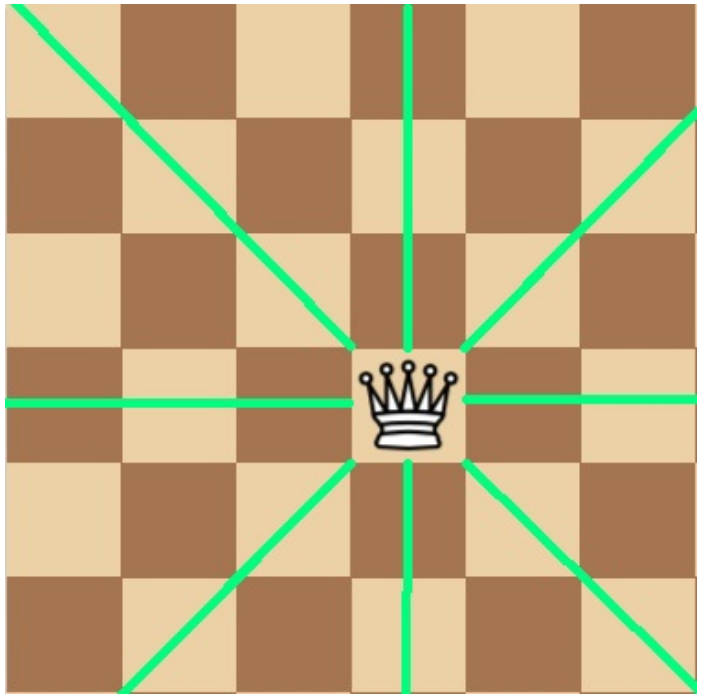


N-Queen problem - Backtracking

Idea: Placing some queens, removes possible locations for the remaining queens.

1. Row by row, place queens in a column without conflict.
2. If no column is found change the column of the last placed queen.
3. Go to 1.

Link to animation



Applications

1. Scheduling - what happens when: factories and transportation.
2. Assignment - who does what: resource management
3. Configuration - what is the optimal configuration: hardware configuration

Constraint Satisfaction Problem (CSP) definition, components

Components:

- **Variables**
- **Domains** - specify allowable **values** for each variable
- **Constraints** specify allowable **combinations of values**.

A CSP solving algorithm should assign a **value** to each **variable** that satisfies all the **constraints**

CSP definition example: Map-coloring

Task: Colour this map with 3 colours so that adjacent regions have different colours

Variables: $\{WA, NT, Q, NSW, V, SA, T\}$

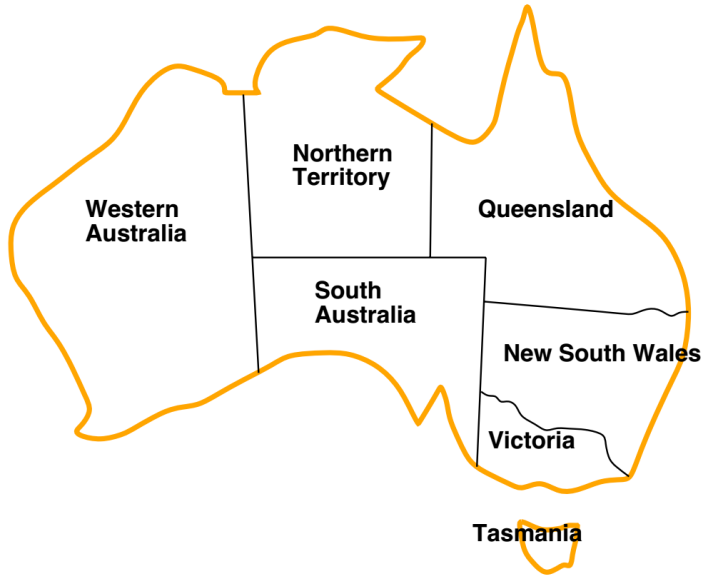
Domain: $\{red, green, blue\}$

Constraints: adjacent regions must have different colors

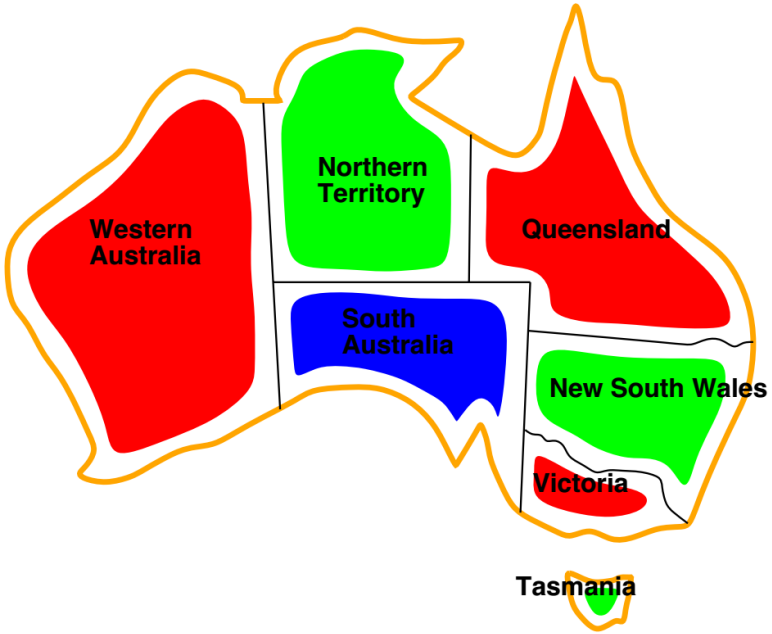
e.g. $WA \neq NT$

or $(WA, NT) \in$

$\{(red, green), (red, blue), (green, red), (green, blue)\}$



Example: Map-Coloring solution



Solutions are assignments satisfying all constraints, e.g.

$$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$$

Constraint types

Unary constraints involve a single variable,

e.g., $SA \neq green$

Binary constraints involve pairs of variables,

e.g., $SA \neq WA$

Global constraints involve 3 or more variables,

e.g., Sudoku, $Alldiff$

Preferences (soft constraints),

e.g., red is better than green

Often includes cost for variable assignment

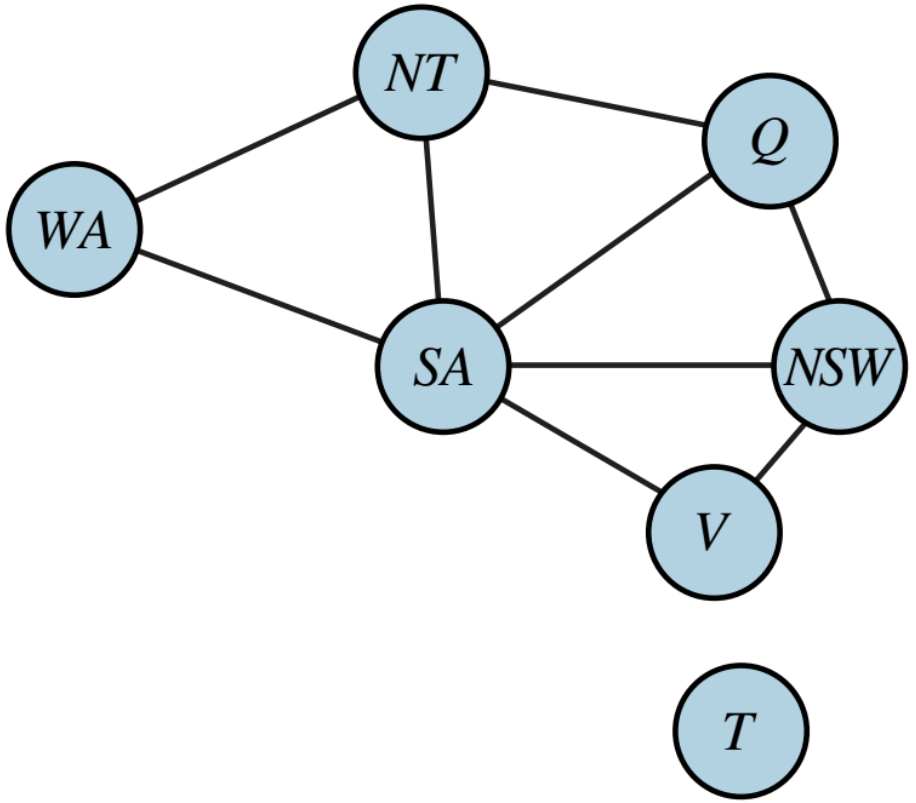
$\rightarrow$ constrained optimization problems

	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

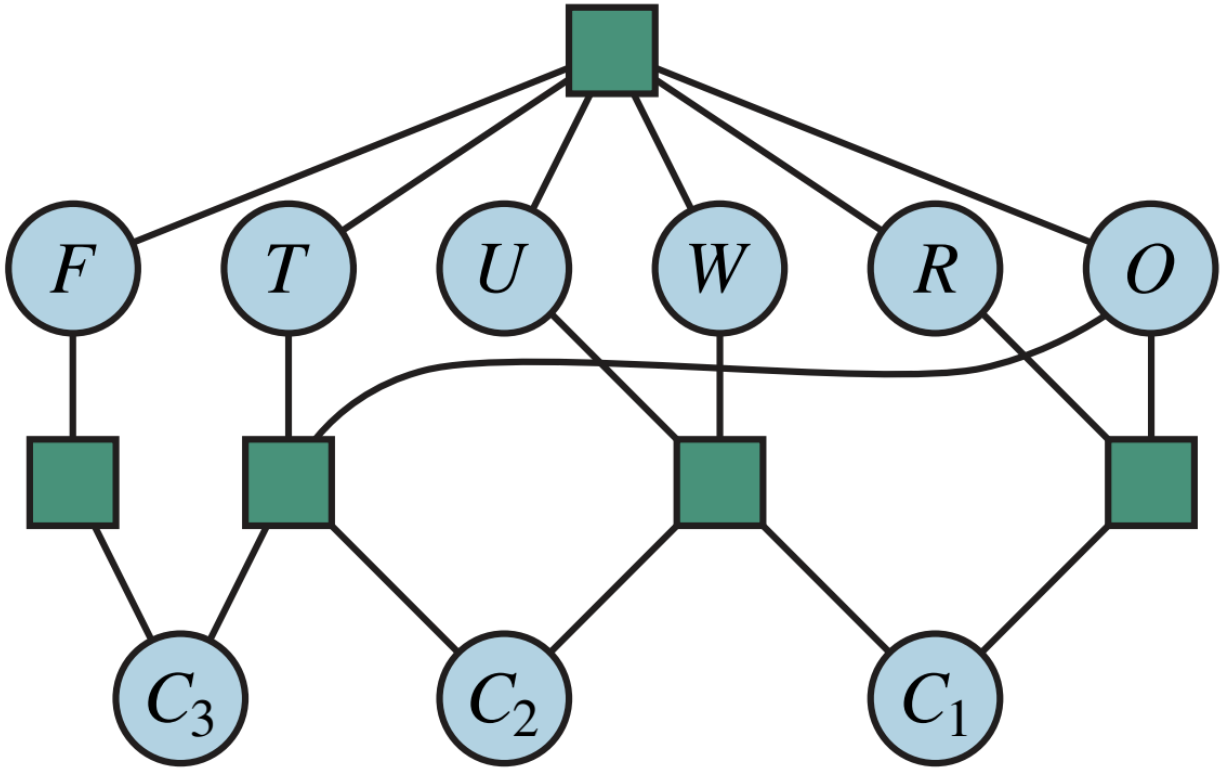
Constraint graph

Nodes - variables, arcs - constraints



Constraint hypergraph

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



Approaches to solving CSPs

1. Inference - Constraint propagation
2. Incremental search - Backtracking
3. Search with inference, e.g., Backtracking with Forward Checking
4. Local Search

Inference - Constraint propagation

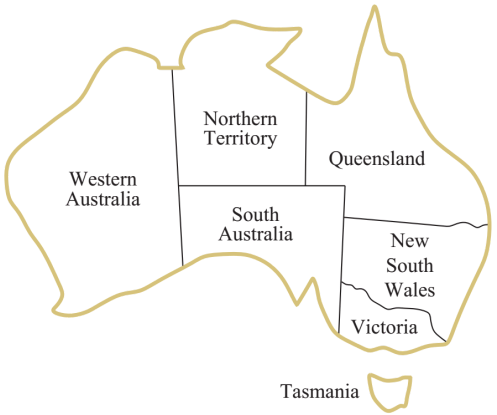
Objective: Use constraints to reduce the number of legal values for a variable (which in turn reduced legal values for other variables).

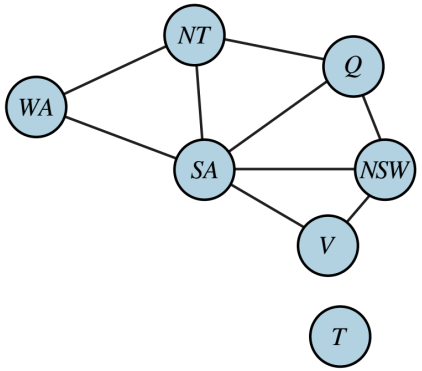
- Part of search
- Preprocessing step
- Sometimes solves the problem

Node consistency

A variable is **node-consistent** if every value in its **domain** satisfies the variables's **unary** constraints.

Example:
Domain: $D(SA) = \{red, green, blue\}$
Constraint: $SA \neq green$
Reduced domain: $R(SA) = \{red, blue\}$





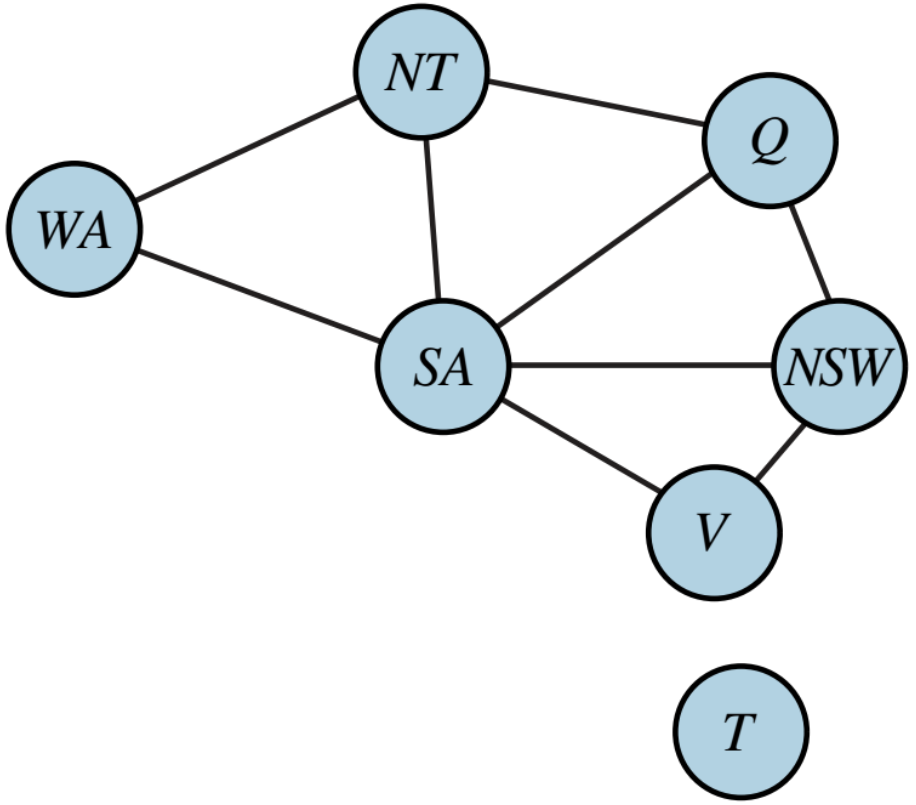
Arc consistency

A variable is **arc-consistent** if every value in its **domain** satisfies the variable's **binary constraints**.

An arc $X \to Y$ is consistent if
For every value in D_x , there exist a value in D_y that satisfies the binary constraint of the arc (X, Y).

Example:
Domain: $D(X) = D(Y) = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Constraint: $Y = X^2$
Reduced domains: $R(X) = \{0, 1, 2, 3\}$, $R(Y) = \{0, 1, 4, 9\}$

Arc consistency - Map coloring



Does arc consistency help to reduce domains?

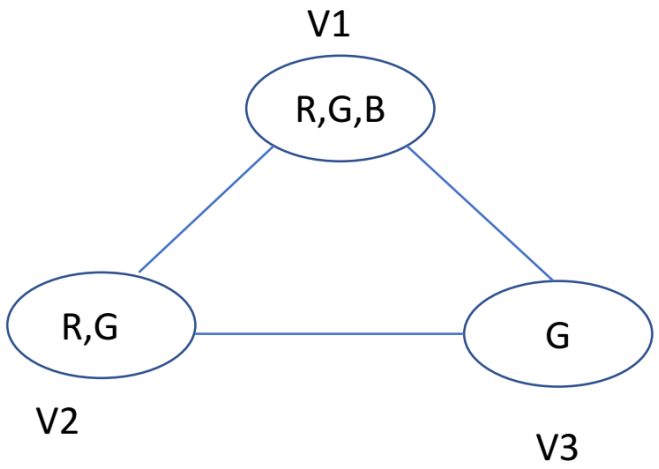
Arc consistency algorithm AC-3

function AC-3(csp) **returns** false if an inconsistency is found and true otherwise
  $queue \leftarrow$ a queue of arcs, initially all the arcs in csp

  while $queue$ is not empty **do**
    $(X_i,\ X_j) \leftarrow$ POP($queue$)
    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 $revised$

AC-3 example



R: Red, G: Green, B: Blue

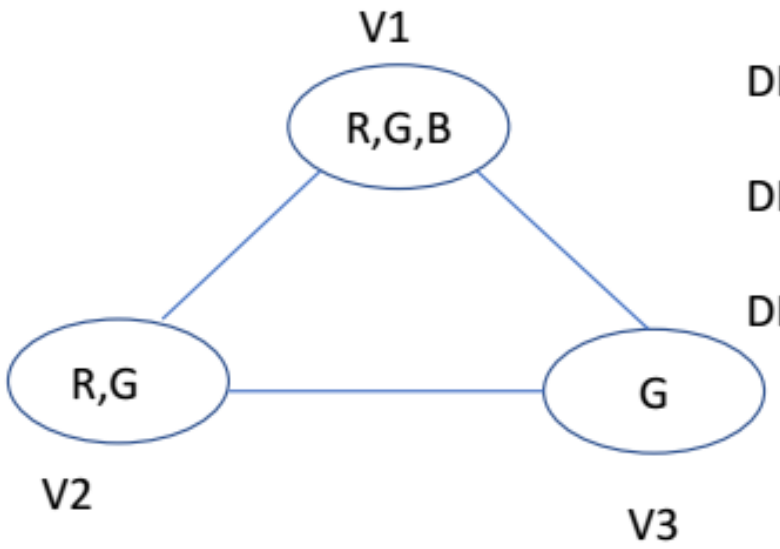
Constraints:

DIFF-color(V1,V3)

DIFF-color(V1,V2)

DIFF-color(V2,V3)

AC-3 example - Step 1



Constraints:

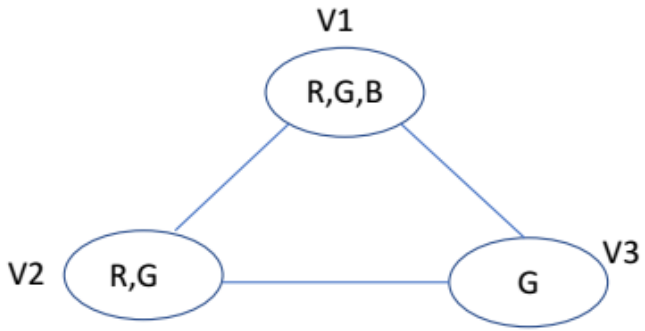
DIFF-color(V1,V3)

DIFF-color(V1,V2)

DIFF-color(V2,V3)

Constraint on the arc	Value deleted
V1-V2	
V2-V1	
V1-V3	
V3-V1	
V2-V3	
V3-V2	

AC-3 example - Step 2



Constraints:

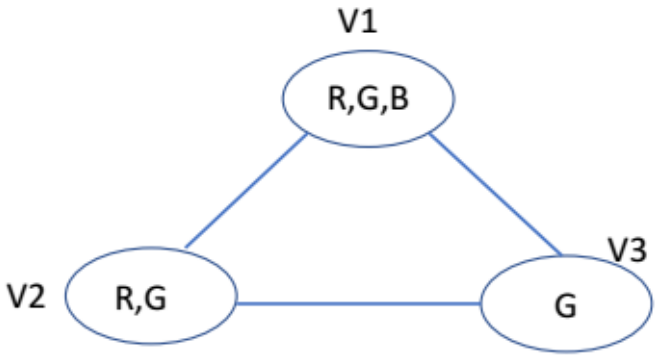
DIFF-color(V1,V2)

DIFF-color(V1,V3)

DIFF-color(V2,V3)

Constraint on the arc	Value deleted	
V1-V2	none	(V1={R,G,B})
V2-V1	none	(V2={R,G})
V1-V3		
V3-V1		
V2-V3		
V3-V2		

AC-3 example - Step 3



Constraints:

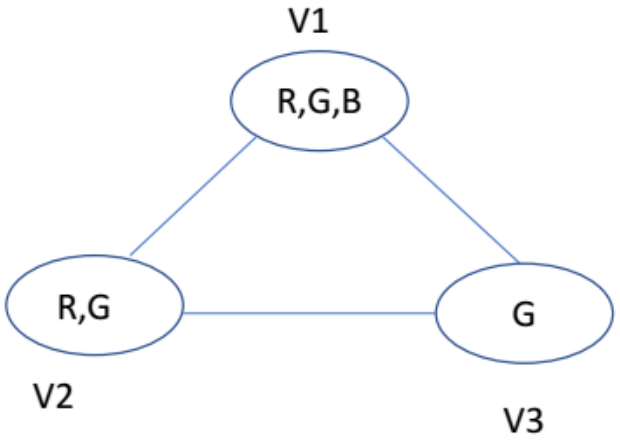
DIFF-color(V1,V2)

DIFF-color(V1,V3)

DIFF-color(V2,V3)

Constraint on the arc	Value deleted	
V1-V2	none	(V1={R,G,B})
V2-V1	none	(V2={R,G})
V1-V3	G	(V1={R,B})
V3-V1		
V2-V3		
V3-V2		
V2-V1		

AC-3 example - Step 4



Constraints:

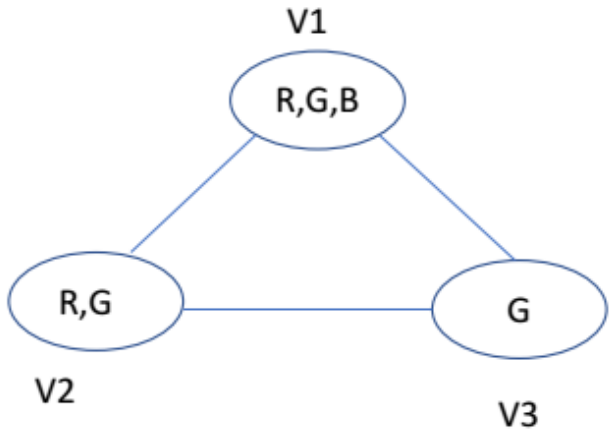
DIFF-color(V1,V2)

DIFF-color(V1,V3)

DIFF-color(V2,V3)

Constraint on the arc	Value deleted	
V1-V2	none	(V1={R,G,B})
V2-V1	none	(V2={R,G})
V1-V3	G	(V1={R,B})
V3-V1	none	(V3={G})
V2-V3	G	(V2={R})
V3-V2		
V2-V1		
V1-V2		

AC-3 example - Step 5



Constraints:

DIFF-color(V1,V2)

DIFF-color(V1,V3)

DIFF-color(V2,V3)

Constraint on the arc	Value deleted	
V1-V2	none	(V1={R,G,B})
V2-V1	none	(V2={R,G})
V1-V3	G	(V1={R,B})
V3-V1	none	(V3={G})
V2-V3	G	(V2={R})
V3-V2	none	
V2-V1	none	
V1-V2	R	(V1={B})
V2-V1	none	
V3-V1	none	

AC-3 solution

> Solutions are assignments satisfying all constraints.

IF one of the domains becomes empty

THEN no solution

ELSE solution exists - *all variables are arc-consistent and no empty domains*

Path-consistency

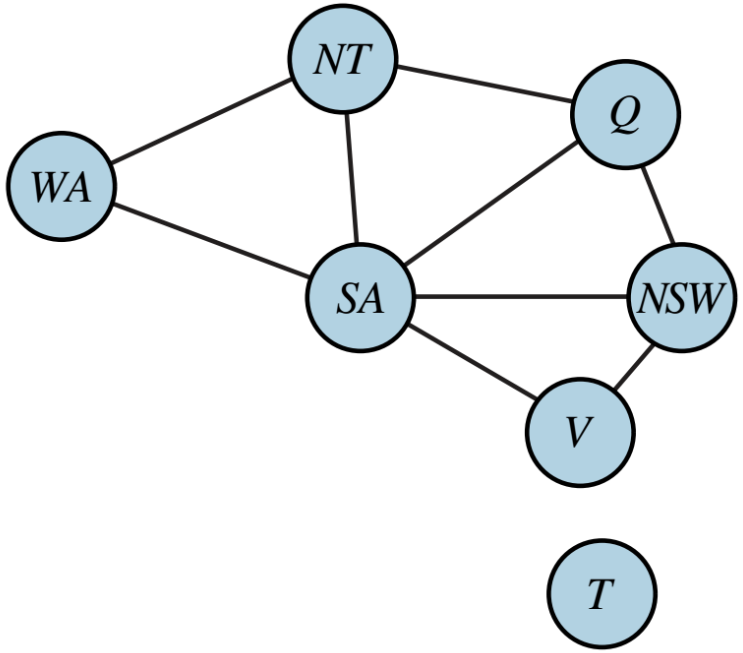
Australia coloring with two colors - arc consistency doesn't help.

Make $\{WA, SA\}$ **path-consistent** with respect to NT:

1. Enumerate assignments for WA and SA: $\{WA = red, SA = blue\}, \{WA = blue, SA = red\}$.
2. Remove assignments that are in conflict with NT.

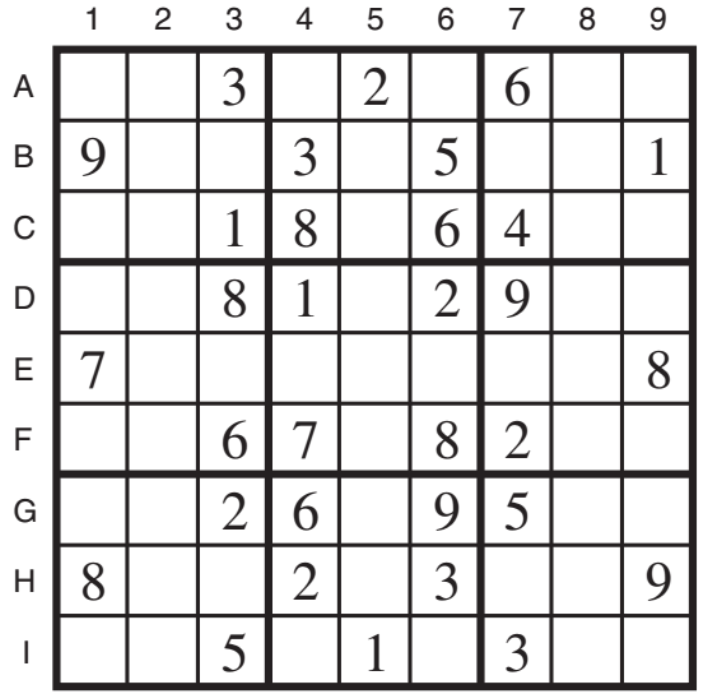
No vald assignments!

Higher-order consistency - **k-consistency**.



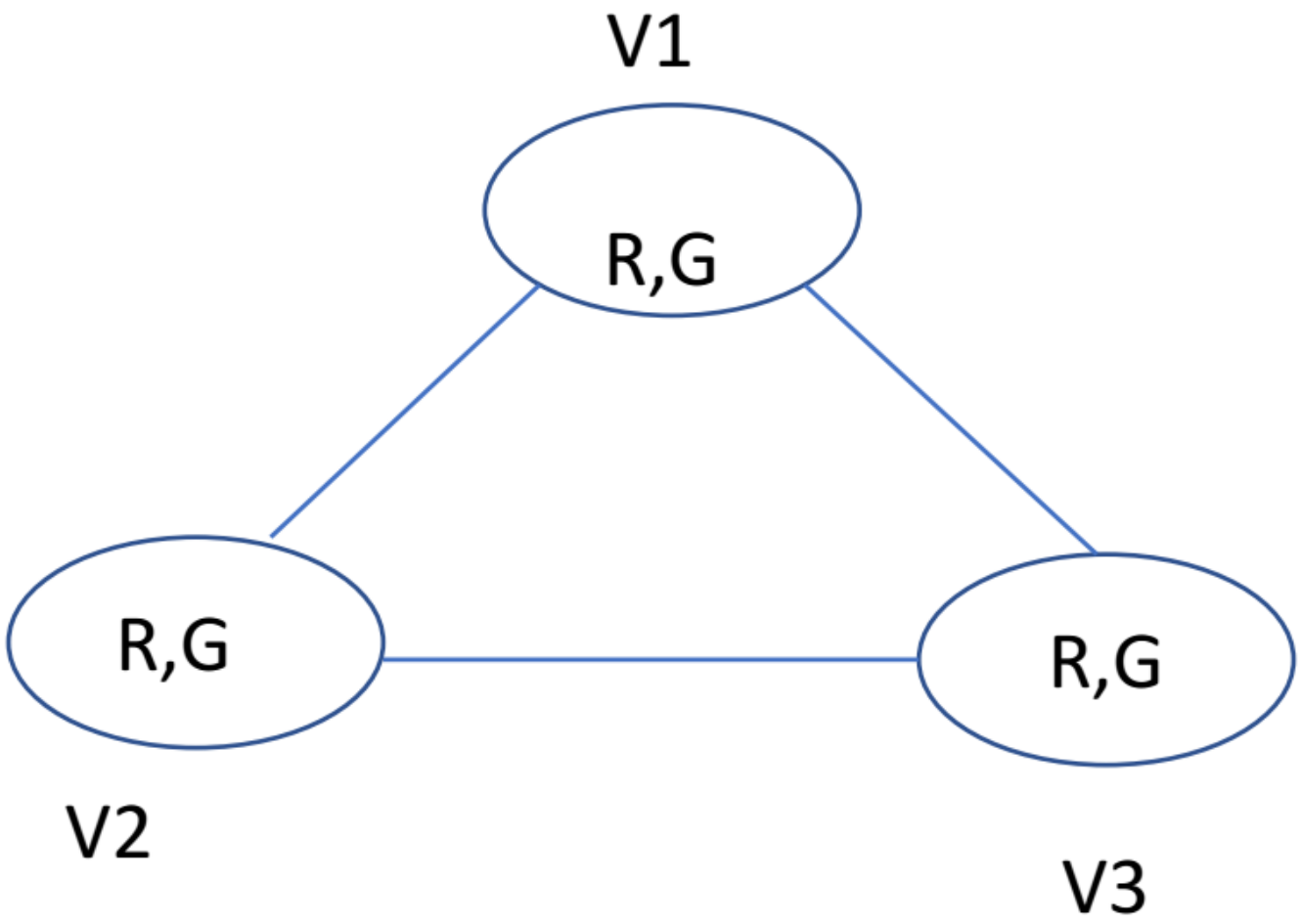
Global constraints

- $Alldiff$
- Resource constraint: $Atmost(10, P_1, P_2)$
- Bounds propagation - $Between(10, 40, P_1, P_2)$



	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		

Is arc consistency sufficient to find a solution?



Constraints:

DIFF-color(V1,V3)

DIFF-color(V1,V2)

DIFF-color(V2,V3)

CSP as search

State-space: Variables assignments so far $\{WA = red, NT = green\}$

Initial state: Empty assignment $\{\}$

Goal states: Complete and consistent assignment

Actions: Variable assignment, e.g. $SA = blue$

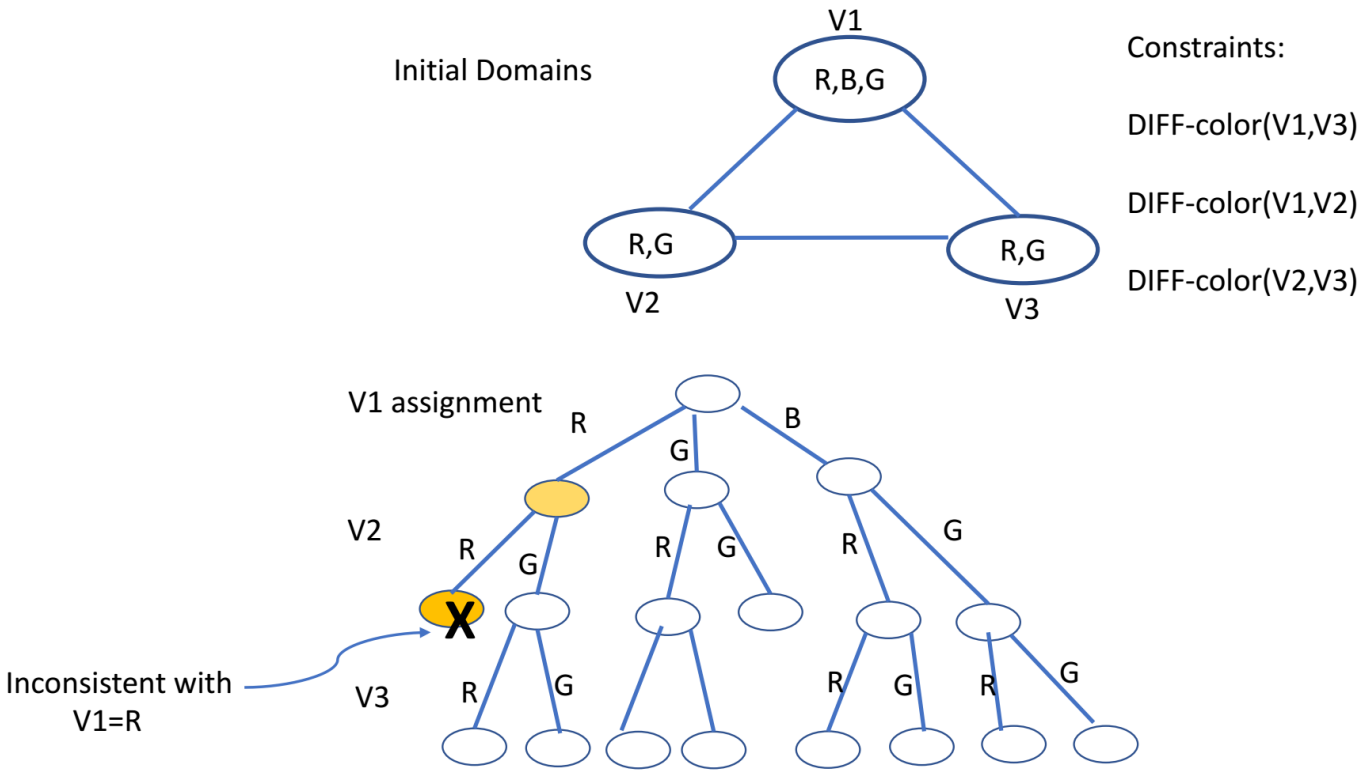
Transition model: $Result(\{WA = red, NT = green\}, SA = blue) = \{WA = red, NT = green, SA = blue\}$

Action cost: 1

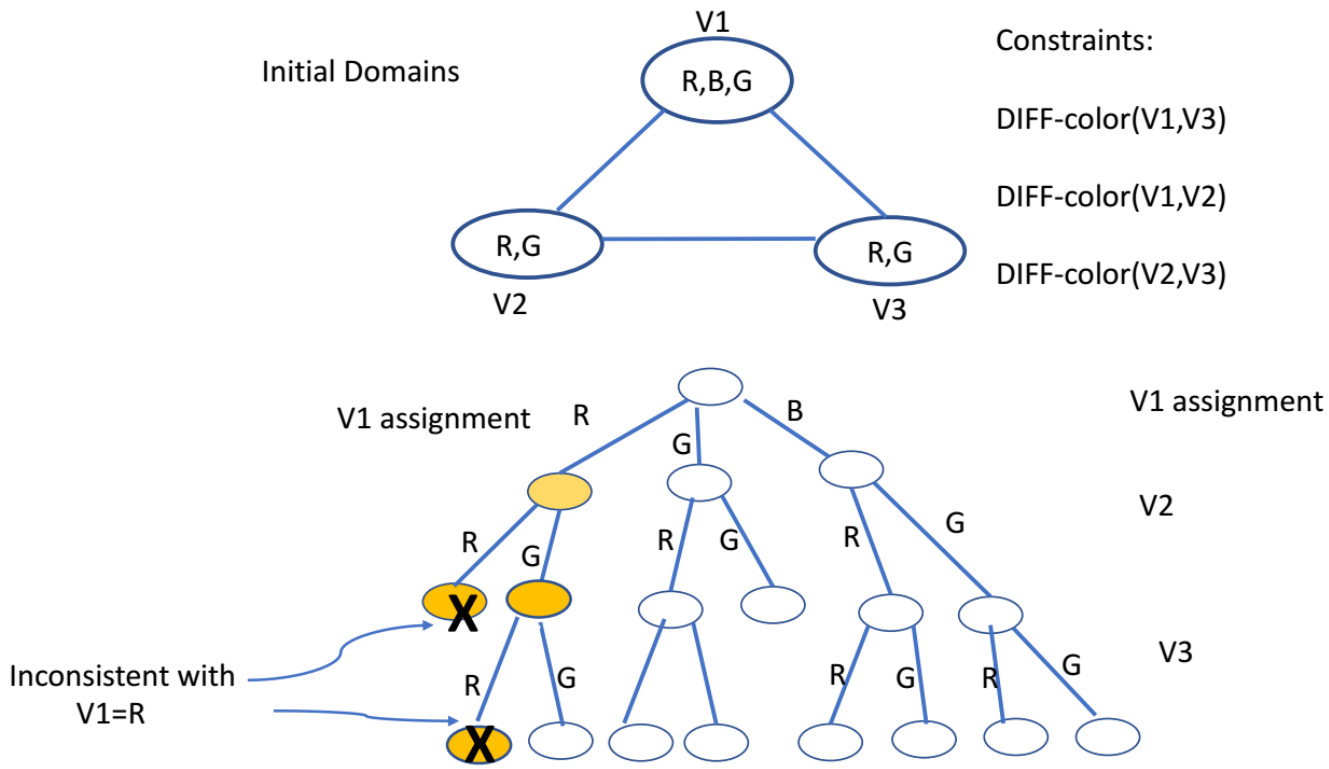
Backtracking search

- **Uninformed**, **depth-first** search for CSP.
- Variable assignment are **commutative**:
  $\{WA = red, NT = green\} = \{NT = green, WA = red\}$

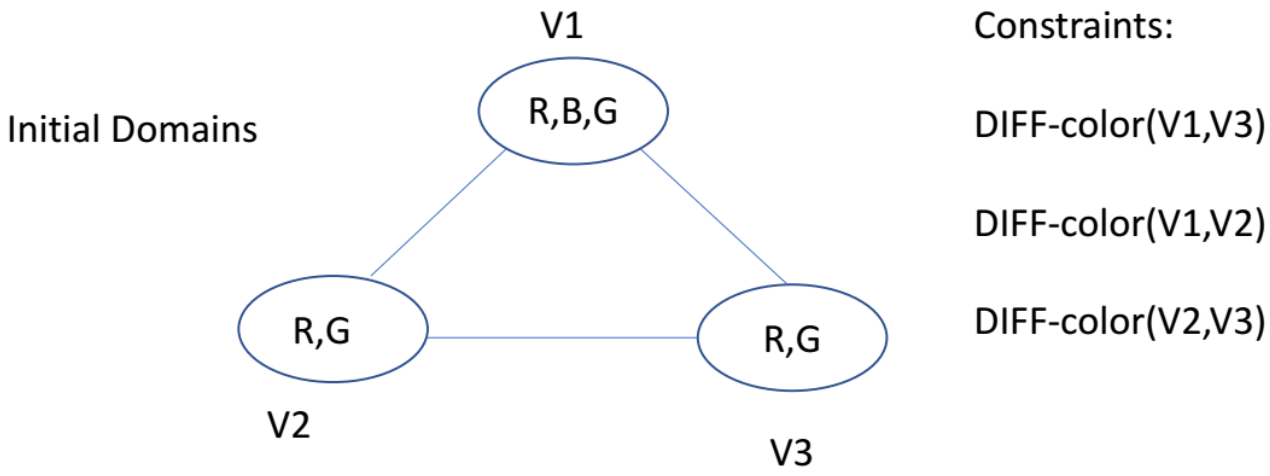
Backtracking example - Step 1

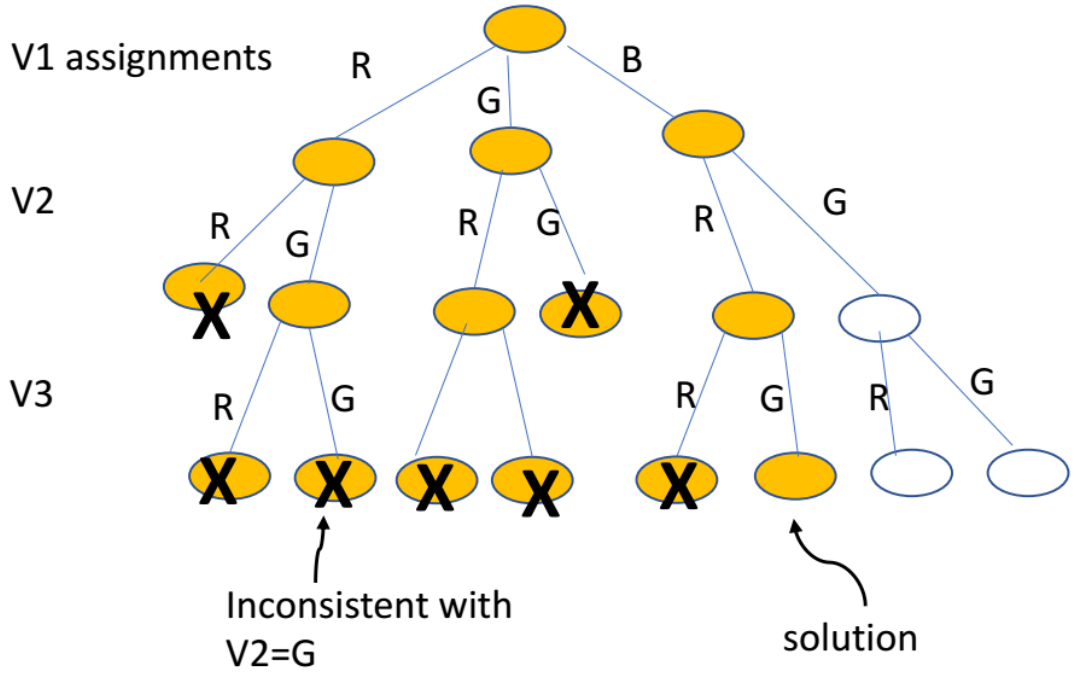


Backtracking example - Step 2



Backtracking example - Step 3





Backtracking algorithm

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
    return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, assignment)
            if inferences ≠ failure then
                add inferences to csp
                result ← BACKTRACK(csp, assignment)
                if result ≠ failure then return result
                remove inferences from csp
            remove {var = value} from assignment
    return failure
```

Improving backtracking using heuristics

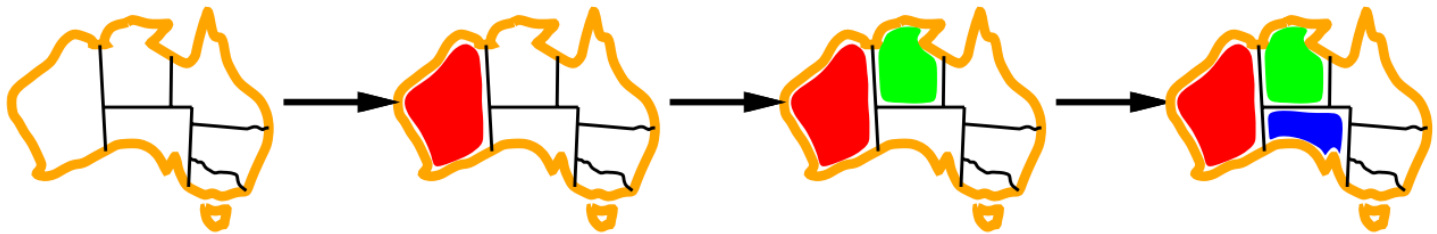
Backtracking can be improved using domain-independent heuristics:

1. Which variable should be assigned next?
   $var \leftarrow$ Select-Unassigned-Variable$(csp, assignment)$
2. In what order should its values be tried?
   for each $value$ **in** Order-Domain-Values$(csp.var.assignment)$ **do**
3. Can we detect unpromissing nodes early?
   $inferences \leftarrow$ Inference$(csp, var, assignment)$
   e.g. use arc consistency
4. Can we take advantage of problem structure?

Minimum-remaining-values (MRV) heuristic

> Which variable should be assigned next?

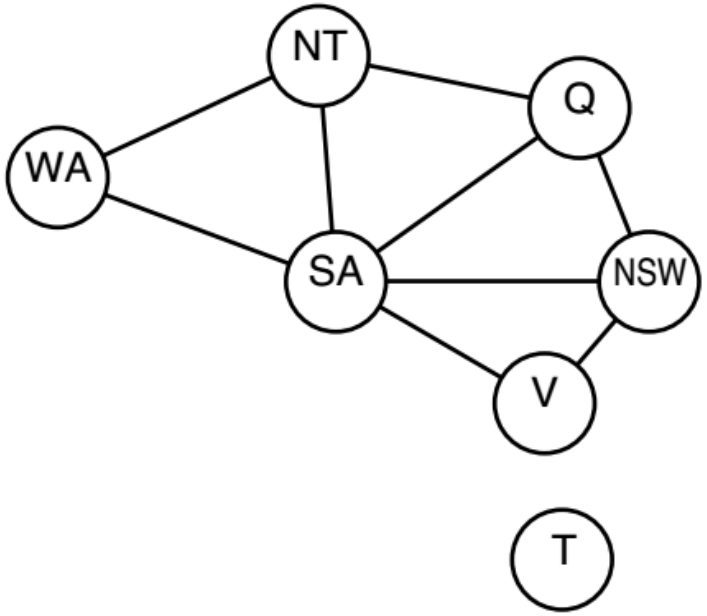
- Choose the variable with the fewest legal values.
- Also called "most constrained variable" or "fail first" heuristic.
- Objective: Detect immediately if variable has no legal values(left) or most likely to cause a failure soon.

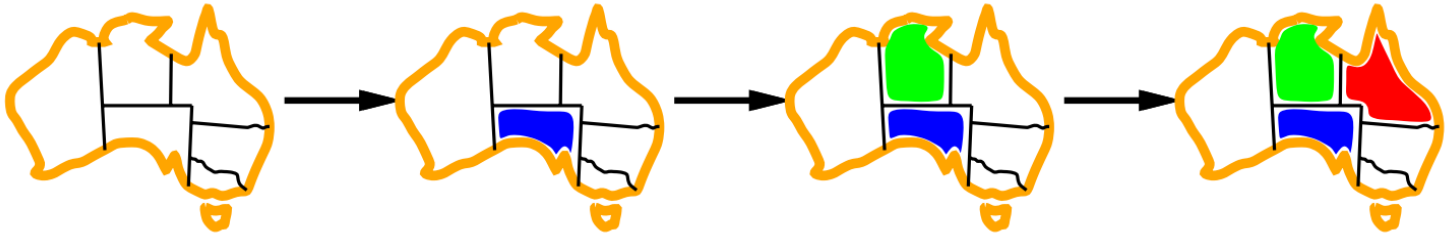


Degree heuristic

> Which variable should be assigned next?

- Heuristic: Choose the variable with the most constraints on remaining variables
- Objective: Reduce the future branching on future choices.
- Helps in the beginning, tie-breaker among MRV variables.

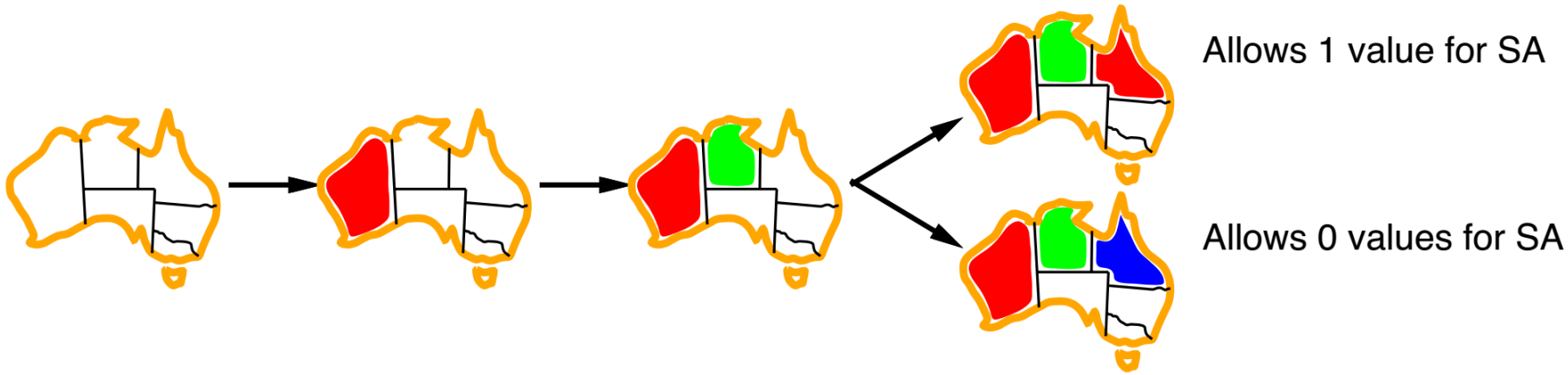




Least-constrained-value heuristic

> In what order should its values be tried?

- Heuristic: Given a variable, choose the value that rules out the fewest values in the remaining variables.
- Objective: Leave the maximum flexibility for subsequent variable assignments.



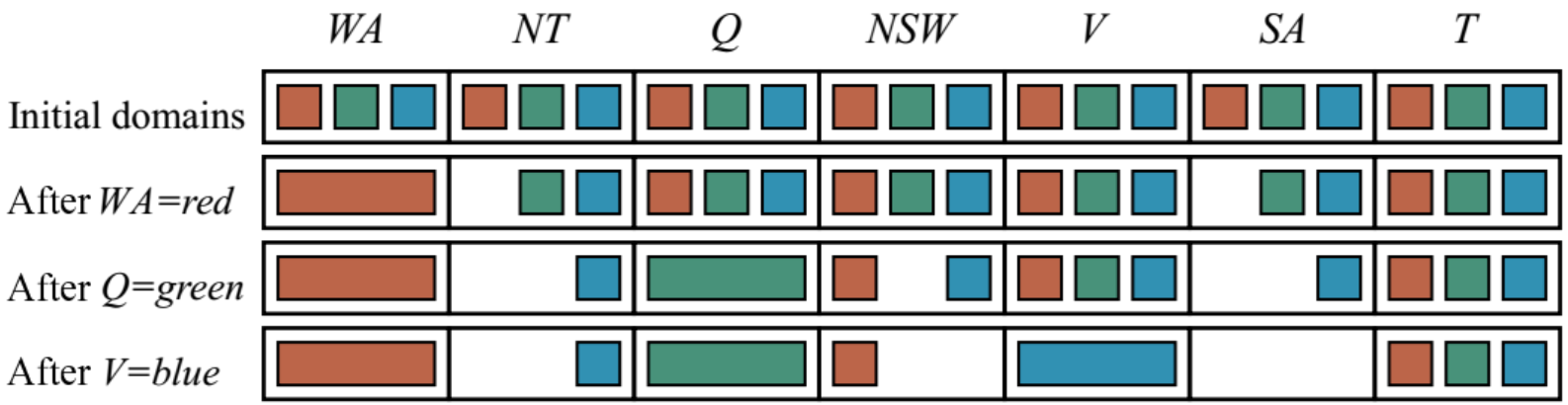
- Backtracking + MRV + degree + least-constrained-value can solve 1000-queens problem vs 25-queens without heuristics.

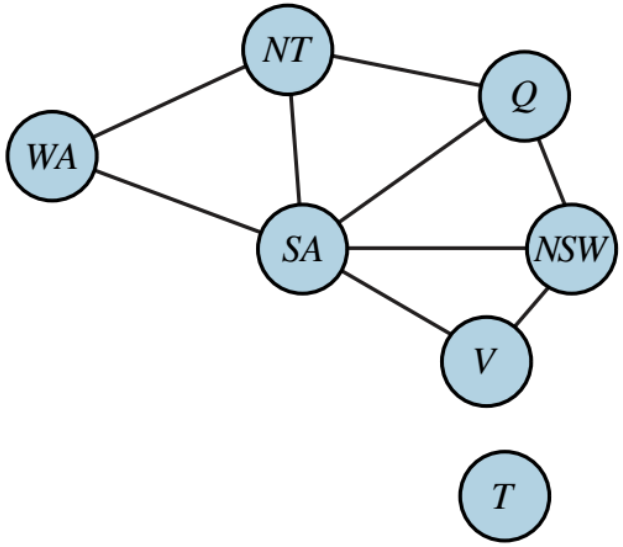
Improving backtracking using inference

> Can we detect unpromissing nodes early?

- Use AC-3 before or during search - time consuming
- **Forward checking**
    i. Arc consistency between unassigned neighbour nodes and current node
    ii. Keep track of remaining legal values of unassigned variables.
    iii. Terminate when any variable has no legal values

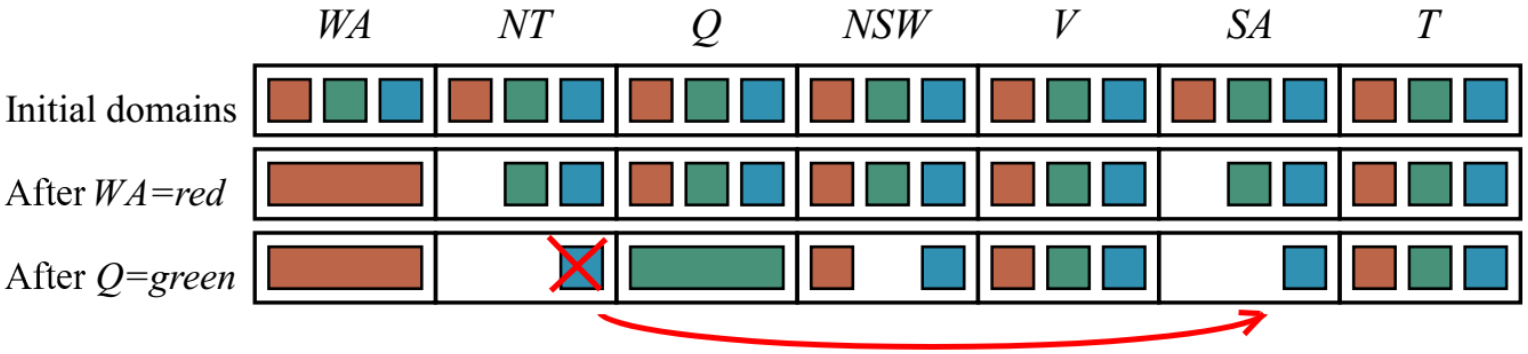
Forward checking example

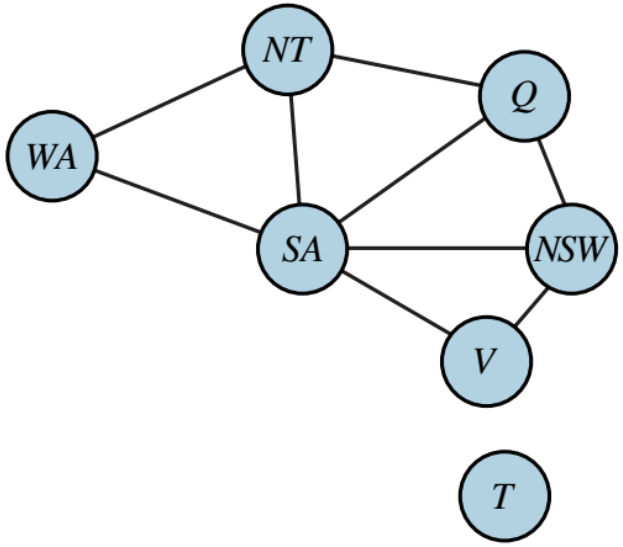




Maintaining arc consistency (MAC)

- Inference with AC-3 starting with arcs to unassigned neighbour nodes.
- Similar to forward checking but with propagation.





Intelligent backtracking

Chronological backtracking - change value for the most recent assignment.

Can we do better?

- **Backjumping** - change value for the most recent assignment in the **conflict set**. Redundant with **forward checking** and **MAC**.
- **Conflict-directed backjumping** - updates **conflict set** to include conflicts with subsequent variables.
- **Constraint learning** - find and avoid **no-good** partial assignments sets that cause the problem.

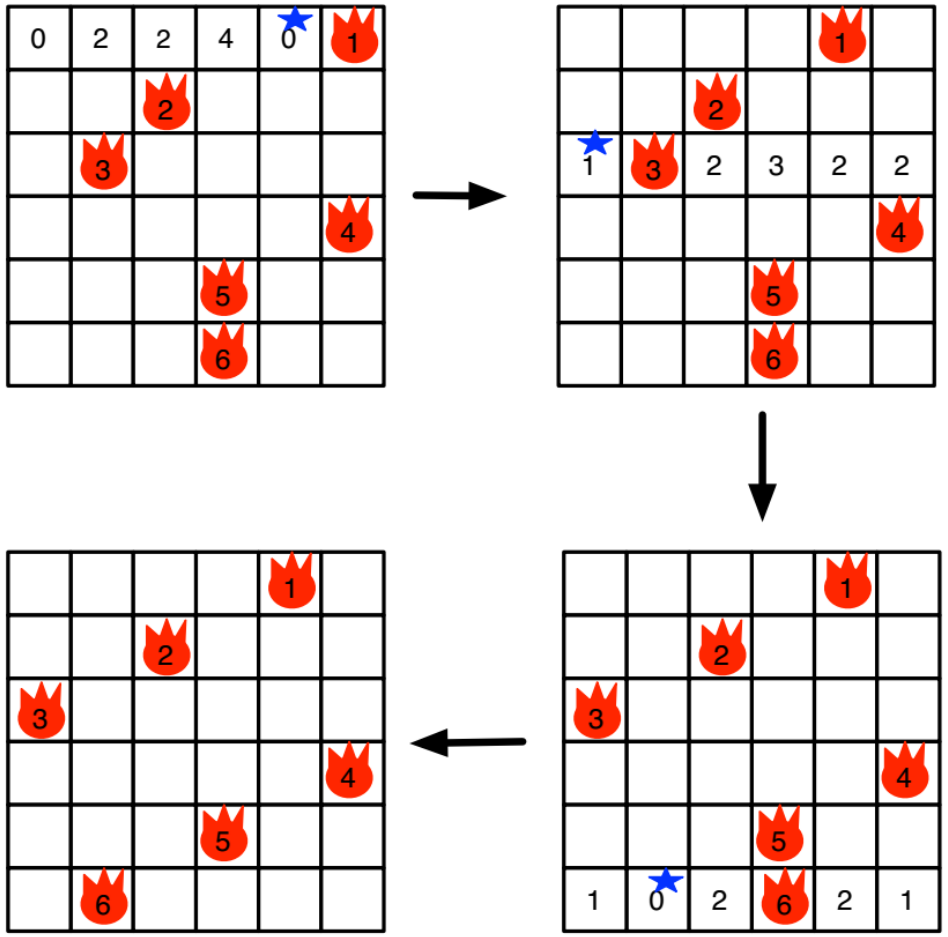
Local search for CSP

Steps:

1. Start with complete assignment, typically violating several constraints.
2. Randomly choose a conflicted variable.
3. Change its value so it brings us closer to a solution,

Min-conflicts heuristic - select a value with the minimum number of conflicts.

Animation



Local search for CSP modifications

- **Tabu search** - helps with plateaus, remembers recent previous states
- **Constaint weighting** - focus on important (often violated) constraints

Local search for CSP performance

Min-conflicts is very effective for many CSPs:

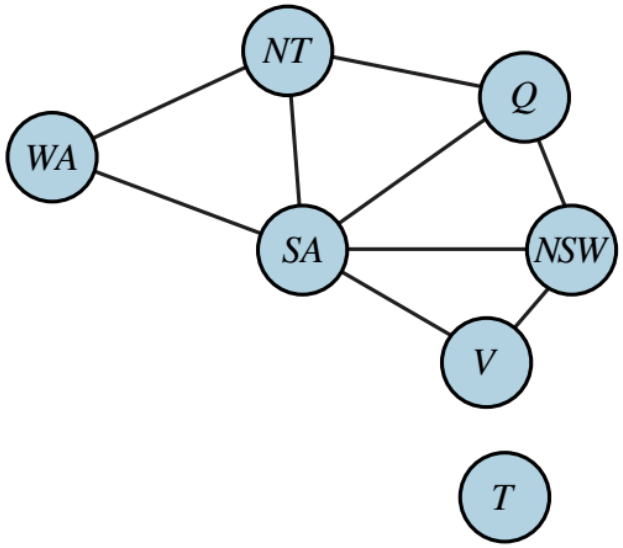
- Million-queen problem in 50 steps (on average)
- Hubble Space Telescope scheduling from weeks to 10 minutes
- Works well in online setting - repairing schedule when flights change.

Problem decomposition

Decompose a problem into **independant subproblems**. Identifiable as **connected components** of the constraint graph.

Reduces time complexity, e.g.
80 boolean variables, 20 in each subproblem
1 million nodes/sec

- Without decomposition: $2^{80} = 4$ billion years
- With decomposition: $4 * 2^{20} = 0.4$ seconds



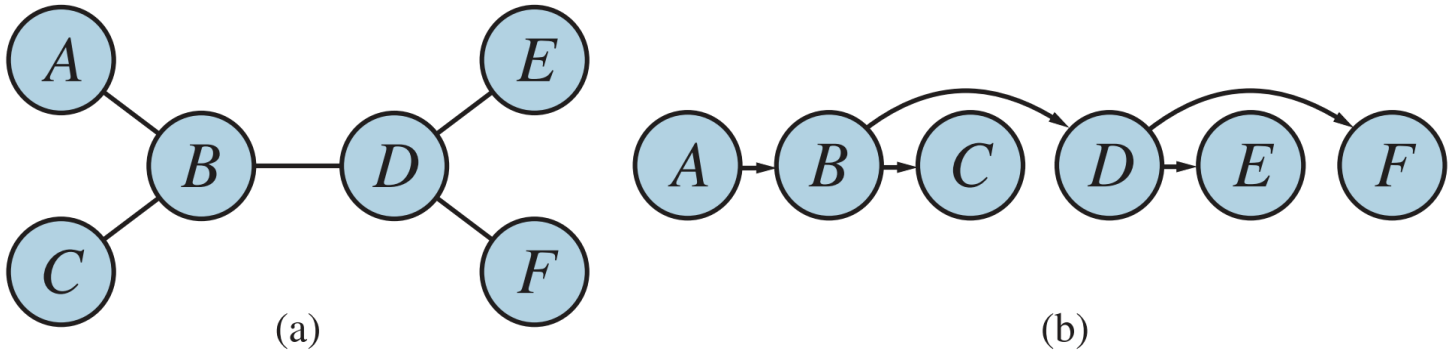
Tree-structured CSP

> Any two variables are connected by only one path.

Can be solved in time linear in the number of variables.

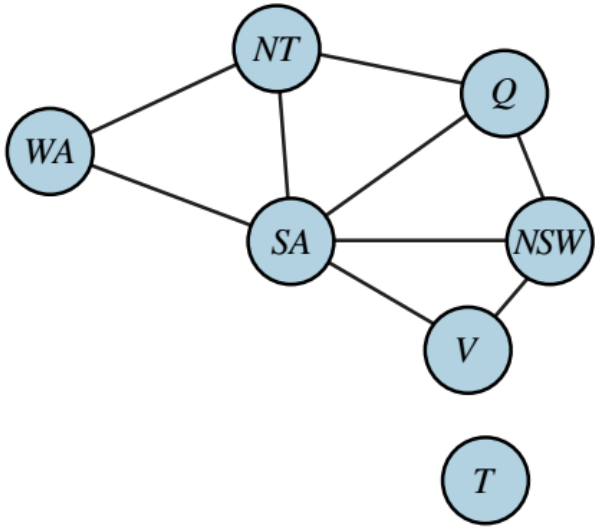
Steps:

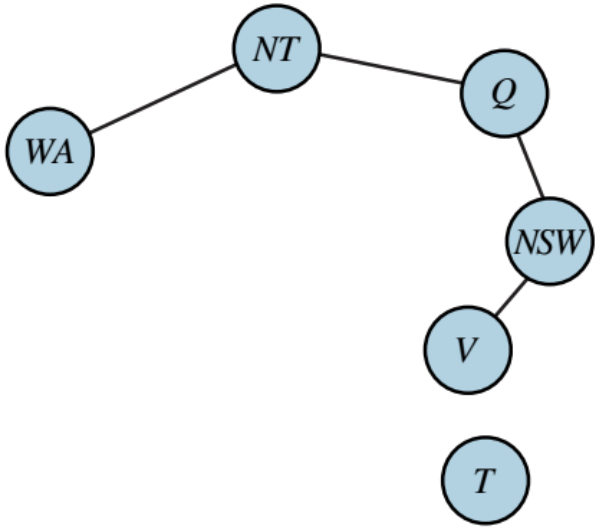
1. **Topological sort** constraint graph.
2. Make all **directed arcs** consistent.
3. Use any remaining values.



Reducing graph to trees - Cutset conditioning

1. Remove subset of variables, called **cycle cutset**, so that the rest is a tree
2. Solve tree CSP for each possible assignment of **cycle cutset**.





Reducing graph to trees - Tree decomposition

1. Transform the original graph into a tree where each node consists of a set of variables.
2. Solve tree CSP