Artificial Intelligence

CONSTRAINT SATISFACTION PROBLEMS

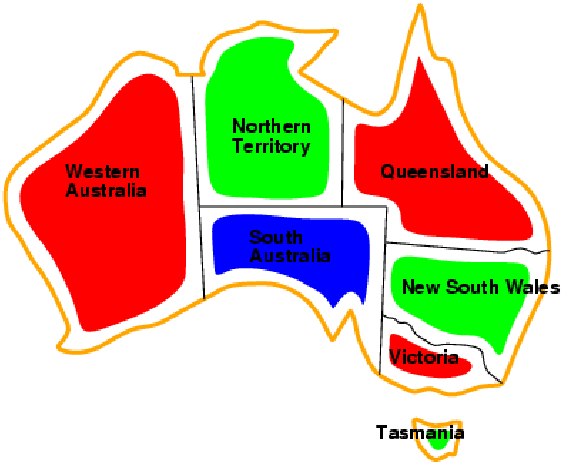
Nguyễn Ngọc Thảo – Nguyễn Hải Minh
{nnthao, nhminh}@fit.hcmus.edu.vn

Outline

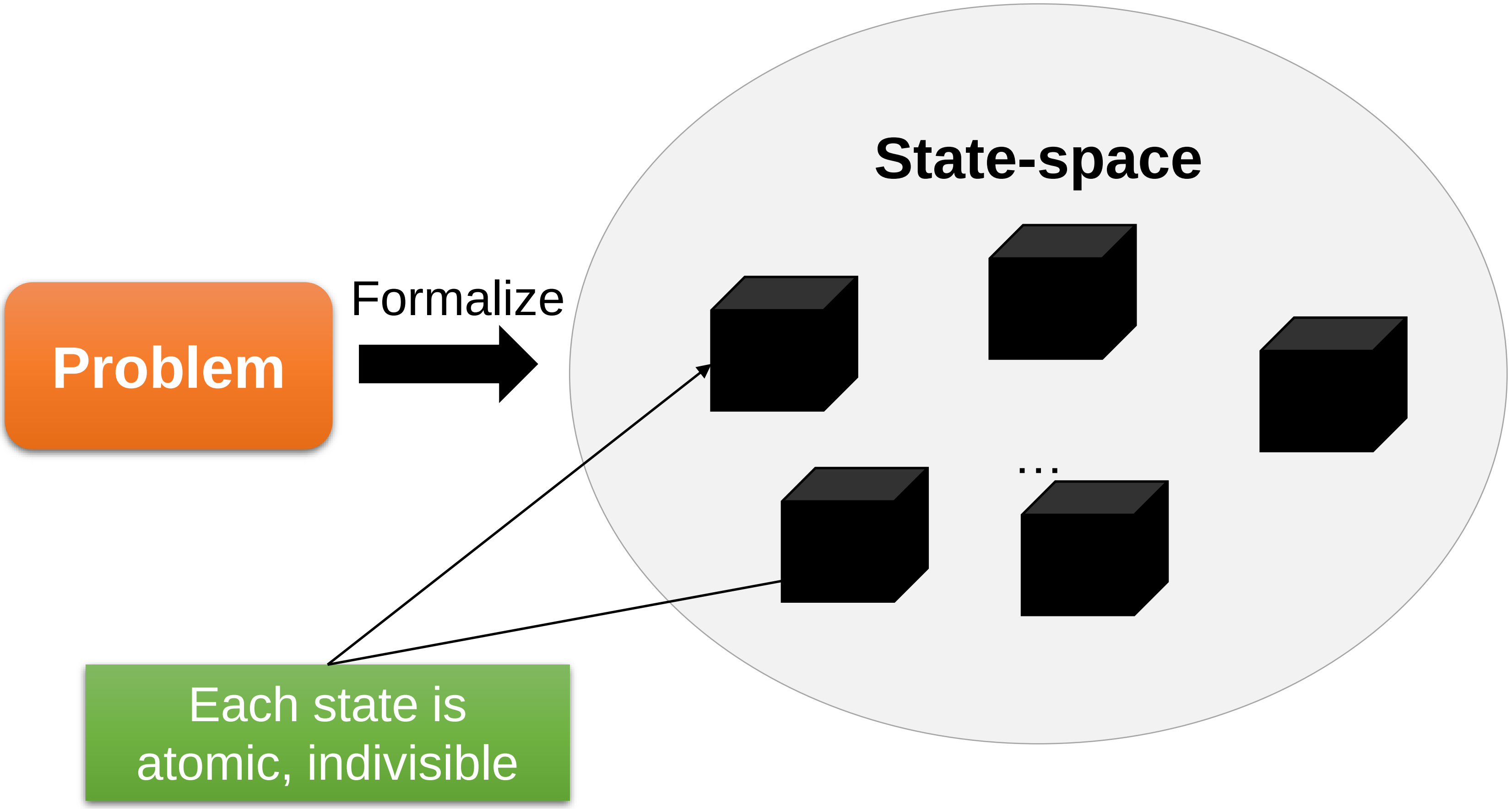
- Constraint Satisfaction Problems (CSPs)
- Constraint Propagation: Inference in CSPs
- Backtracking Search for CSPs
- Local Search for CSPs
- The Structure of Problems

Constraint Satisfaction Problem

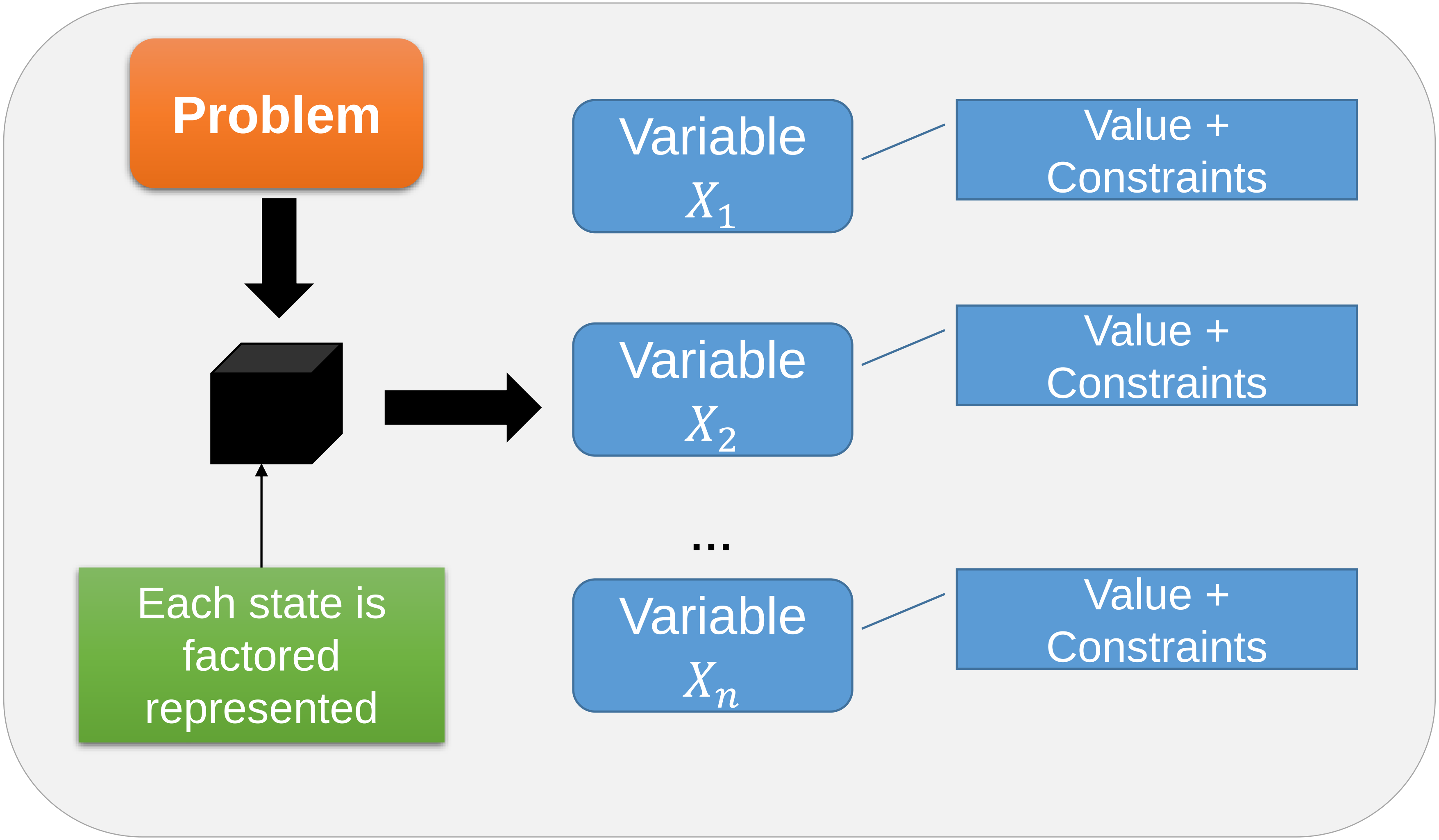
- *Defining Constraint Satisfaction Problems*
- *Example Problem: Map Coloring*
- *Example Problem: Job-shop Scheduling*
- *Variations on the CSP Formalism*



State-space search problems



Constraint Satisfaction Problem



Constraint Satisfaction Problem

- A **constraint satisfaction problem** (**CSP**) uses a factored representation for each state.
  - State = a set of variables and each of which has a value
  - Solution = each variable has a value that satisfies all constraints on that variable
- Take advantage of the structure of states
- General-purpose rather than problem-specific heuristics
  - Identify combinations of variable-value that violate the constraints

    $\rightarrow$ eliminate large portions of the search space all at once
  - Solutions to complex problems

Constraint Satisfaction Problem

- A **CSP** consists of the following three components

$\boldsymbol{X} = \{X_1, \ldots, X_n\}$: a set of variables

$\boldsymbol{D} = \{D_1, \ldots, D_n\}$: a set of domains, one for each variable.

  - $D_i = \{v_1, \ldots, v_k\}$: set of allowable values for variable X_i

$\boldsymbol{C}$**:** a set of constraints that state allowable combinations of values.

- Each C_i consists of a pair $\langle scope, rel \rangle$
  - $scope$: a tuple of variables that participate in the constraint
  - A relation rel defines the values that participated variables can take

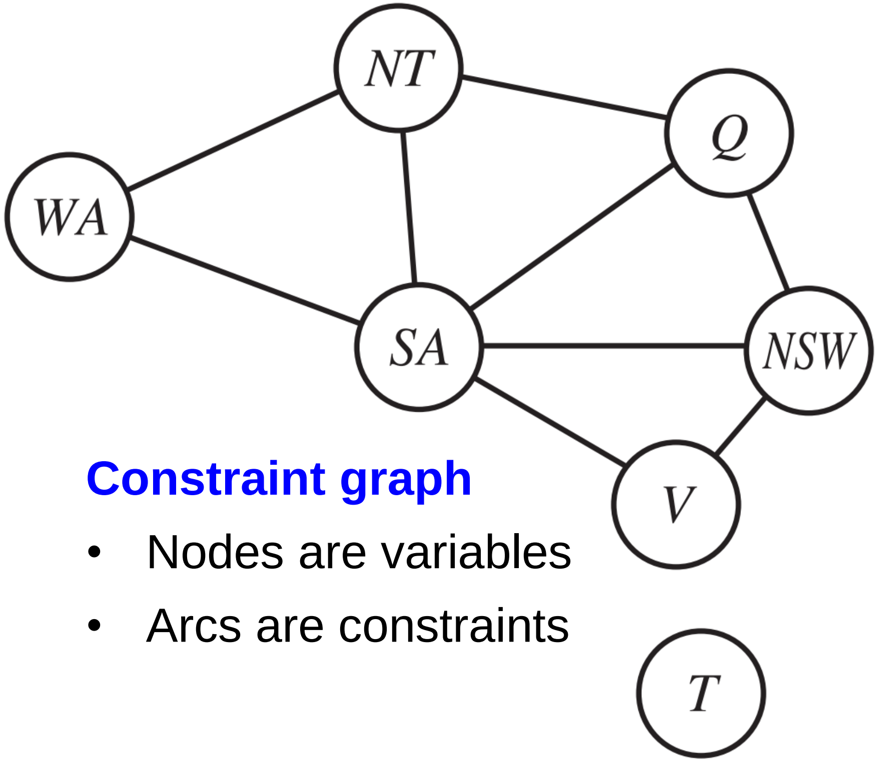
Constraints in CSPs

- Assume that both X_1 and X_2 have the domain $\{A, B\}$
- *"Two variables must have different values"*
- A relation can be an explicit list of all tuples of values that satisfy the constraint.
  - E.g., $\langle (X_1, X_2), [(A, B), (B, A)] \rangle$
- It can be an abstract relation that supports two operations
  - Test whether a tuple is a member of the relation
  - Enumerate the members of the relation
  - E.g., $\langle (X_1, X_2), X_1 \neq X_2 \rangle$

Solutions for CSPs

- Each state is defined by an assignment of values to some or all the variables, $\{X_i = v_i, X_j = v_j, \ldots\}$.
- A **solution** to a CSP is a consistent – complete assignment.
  - A consistent assignment does not violate any constraints.
  - A complete assignment has every variable assigned, while a partial assignment assigns values to only some variables.

Example problem: Map coloring



Constraint graph

- Nodes are variables
- Arcs are constraints

- Color each region either **red**, **green**, or **blue** in such a way that no neighboring regions have the same color

Example problem: Map coloring

- Variables: $X = \{WA, NT, Q, NSW, V, SA, T\}$
- Domains: $D_i = \{red, green, blue\}$
- Constraints: Adjacent regions must have different colors

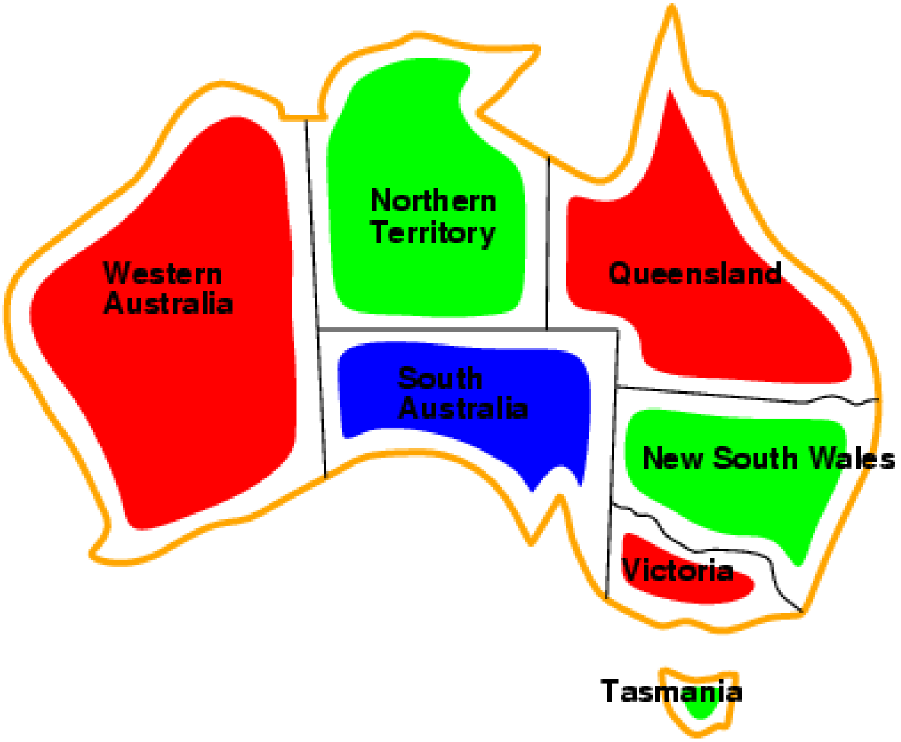
$$C = \begin{Bmatrix} SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, \\ WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V \end{Bmatrix}$$

  - where $SA \neq WA$ is a shortcut of $\langle (SA, WA), SA \neq WA \rangle$
  - $SA \neq WA$ can be fully enumerated as {(red,green), (red,blue), (green,red), (green,blue), (blue,red), (blue,green)}

Example problem: Map coloring

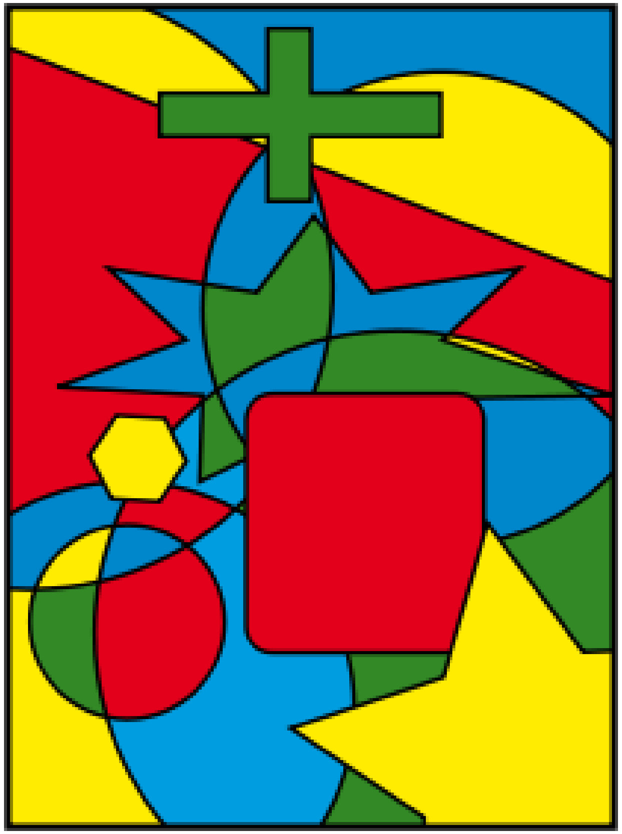
- There are many possible solutions to this problem
- For example,

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

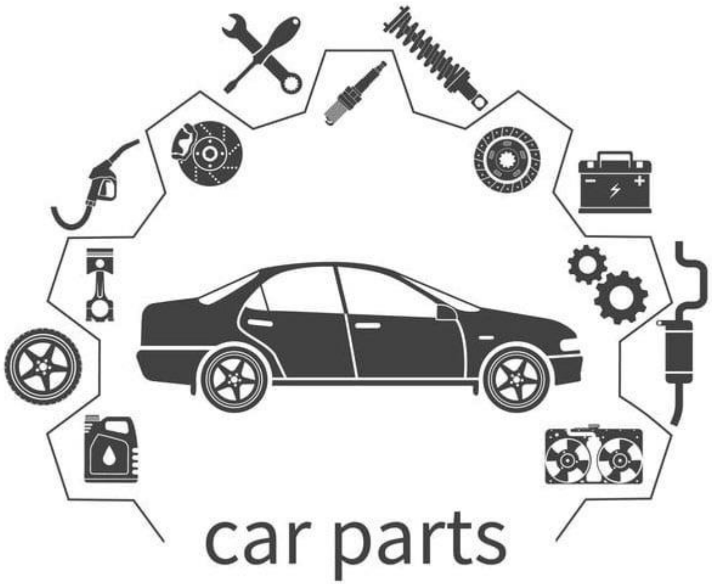


Aside: The Graph Coloring Problem

- More general problem than map coloring
- Planar graph = graph in the 2D plane with no edge crossings
- Guthrie's conjecture (1852): *Every planar graph can be colored with 4 colors or less*
  - Proved (using a computer) in 1977 (Appel and Haken)



Example problem: Job-shop scheduling



15 tasks

- Install axles (front and back)
- Affix all four wheels (right and left, front and back)
- Tighten nuts for each wheel
- Affix hubcaps, and
- Inspect the final assembly

- Some tasks must occur before another while many other tasks can go on at once.
  - E.g., a wheel must be installed before the hubcap is put on
- A task takes a certain amount of time to complete.

Example problem: Job-shop scheduling

- Variables: $X = \{Axle_F, Axle_B, Wheel_{RF}, Wheel_{LF}, Wheel_{RB}, Wheel_{LB}, Nuts_{RF}, Nuts_{LF}, Nuts_{RB}, Nuts_{LB}, Caps_{RF}, Caps_{LF}, Caps_{RB}, Caps_{LB}, Inpsect\}$
- Domains: The time that the task starts
- Assume that the task T_1 and T_2 take duration d_1 and d_2 to complete, respectively
- Precedence constraints: A task T_1 must occur before task T_2, i.e., $\boldsymbol{T_1 + d_1 \leq T_2}$
- Disjunctive constraints: The tasks T_1 and T_2 must not overlap in time, i.e., $\boldsymbol{T_1 + d_1 \leq T_2}$ or $\boldsymbol{T_2 + d_2 \leq T_1}$

Example problem: Job-shop scheduling

- The axles must be in place before the wheels are put on. Installing an axle takes 10 minutes.

$$Axle_F + 10 \le Wheel_{RF} \qquad Axle_F + 10 \le Wheel_{RF}$$
$$Axle_B + 10 \le Wheel_{RB} \qquad Axle_B + 10 \le Wheel_{LB}$$

- For each wheel, affix the wheel (which takes 1 minute), then tighten the nuts (2 minutes), and finally attach the hubcap (1 minute)

$$Wheel_{RF} + 1 \le Nut_{RF} \qquad Nuts_{RF} + 2 \le Cap_{RF}$$
$$Wheel_{LF} + 1 \le Nut_{LF} \qquad Nuts_{LF} + 2 \le Cap_{LF}$$
$$Wheel_{RB} + 1 \le Nut_{RB} \qquad Nuts_{RB} + 2 \le Cap_{RB}$$
$$Wheel_{LB} + 1 \le Nut_{LB} \qquad Nuts_{LB} + 2 \le Cap_{LB}$$

- Suppose we have four workers to install wheels, but they must share one tool that helps put the axle in place.

$$Axle_F + 10 \le Axle_B \quad or \quad Axle_B + 10 \le Axle_F$$

Example problem: Job-shop scheduling

- The inspection comes last and takes 3 minutes → for every variable except $Inspect$, add a constraint of the form $X + d_X \leq Inspect$.

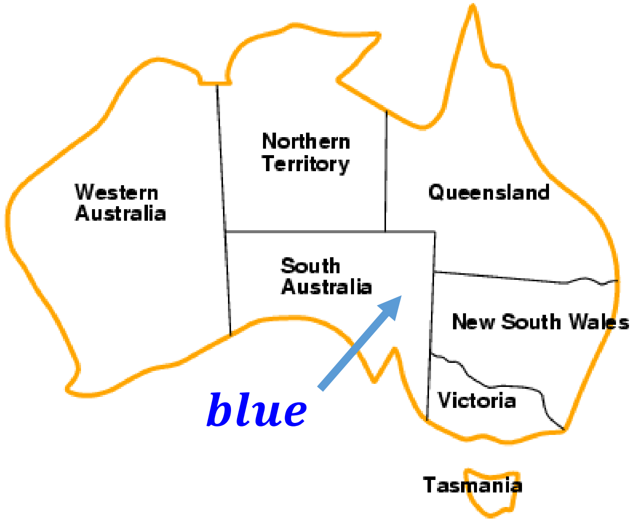
- Finally, suppose there is a requirement to get the whole assembly done in 30 minutes → limit the domain of all variables to $D_i = \{1, 2,3, \dots, 27\}$.

Why formulate a problem as a CSP?

- Provide natural representation for a wide variety of problems
- Many problems intractable in regular state-space search can be solved quickly with CSP formulation.
  - E.g., the Australian problem

    Search: $3^5 = 243$ assignments

    CSP: $2^5 = 32$ assignments $\downarrow$ 87%



- Better insights to the problem and its solution

Variations on the CSP formalism

- Discrete and finite variables
  - n variables, domain size $d \rightarrow O(d^n)$ complete assignments
  - E.g., map coloring, scheduling with time limits, 8-queens etc.
- Discrete, infinite domains
  - Sets of Integers, strings, etc.
  - E.g., job scheduling without deadlines
  - **Constraint language:** understand constraints without enumeration, e.g., StartJob1 + 5 ≤ StartJob3
- Continuous domains
  - Real-world problems often involve continuous domains and even real-valued variables.

Real-world CSPs

- Operations Research (scheduling, timetabling)
  - Scheduling the time of observations on the Hubble Space Telescope
- Linear programming
  - Constraints must be linear equalities or inequalities $\rightarrow$ solved in time polynomial in the number of variables.
- Bioinformatics (DNA sequencing)
- Electrical engineering (circuit layout-ing)
- Airline schedules
- Cryptography
- Computer vision: image interpretation
- …

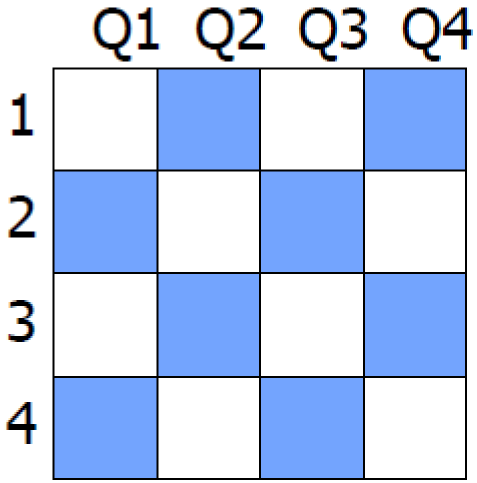
Types of constraints

- Unary constraint: restrict the value of a single variable
  - E.g., the South Australians do not like green $\rightarrow \langle (SA), SA \neq green \rangle$
- Binary constraint: relate two variables
  - E.g., $\langle (SA, WA), SA \neq WA \rangle$
- Higher-order constraints: involve three or more variables
  - E.g., Professors A, B, and C cannot be on a committee together
  - Always possible to be represented by multiple binary constraints
- Global constraints: involving an arbitrary number of variables
  - $Alldiff =$ all variables involved must have different values
  - E.g., Sudoku: all variables in a row/column must satisfy an $Alldiff$

Preference constraints

- Which solutions are preferred $\rightarrow$ soft constraints
  - E.g., red is better than $green$ $\rightarrow$ this often can be represented by a cost for each variable assignment
- **Constraint optimization problem** (COP): a combination of optimization with CSPs $\rightarrow$ linear programming

Example: 4-Queens Problem

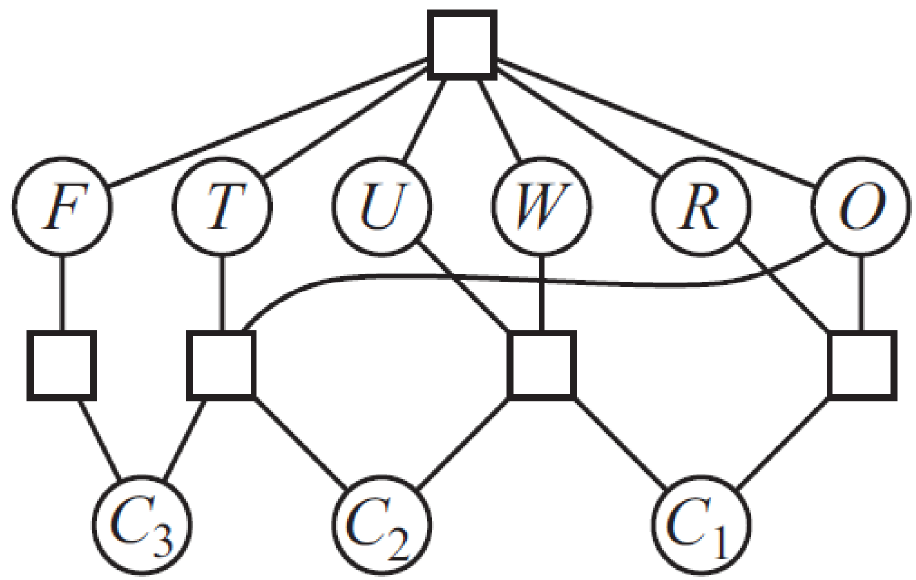


- Variables: $Q1, Q2, Q3, Q4$
- Domains: $\mathrm{D} = \{1,2,3,4\}$
- Constraints
  - $Qi \neq Qj$ (cannot be in the same row)
  - $Qi - Qj \neq i - j$ (cannot be in the same diagonal)

Example: Cryptarithmetic

- Variables: $F \; T \; U \; W \; R \; O \; C_1 \; C_2 \; C_3$
- Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints:
  - $Alldiff(F, T, U, W, R, O)$
  - $C_3 = F, T \neq 0, F \neq 0$
  - $\ldots$

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



Constraint Propagation

- *Node Consistency*
- *Arc Consistency*
- *Path Consistency*
- *K-Consistency*
- *Global Constraints*

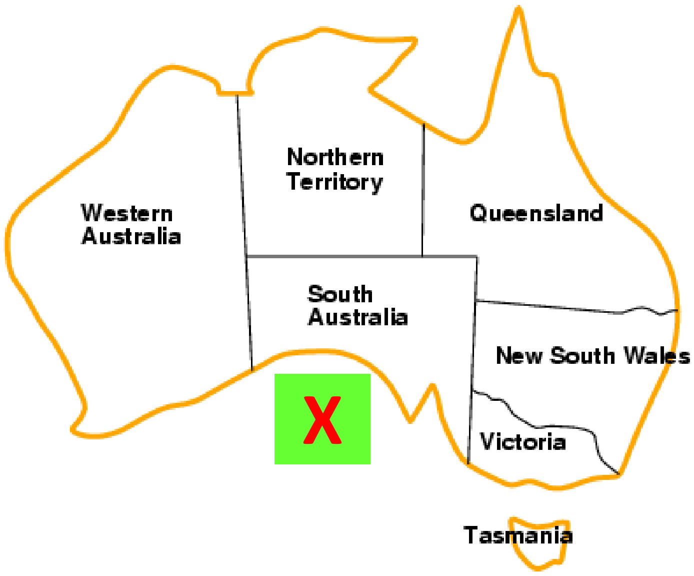
Constraint propagation

- Constraints help to reduce the number of legal values for a variable $\rightarrow$ legal values for another variable are also reduced
- Intertwined with search, or done as a preprocessing step
  - Sometimes the preprocessing can solve the whole problem!
- Enforcing **local consistency** in each part of a graph causes inconsistent values to be eliminated throughout the graph

Node consistency

- A single variable is node-consistent if all the values in the variable's domain satisfy the variable's unary constraints.

The South Australians dislike green, the domain of $\{SA\}$ will be $\{red, \cancel{green}, blue\}$

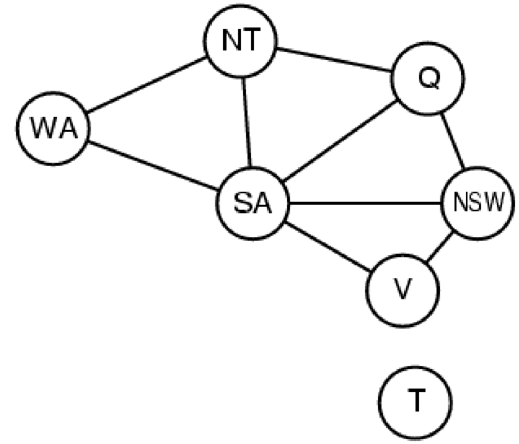


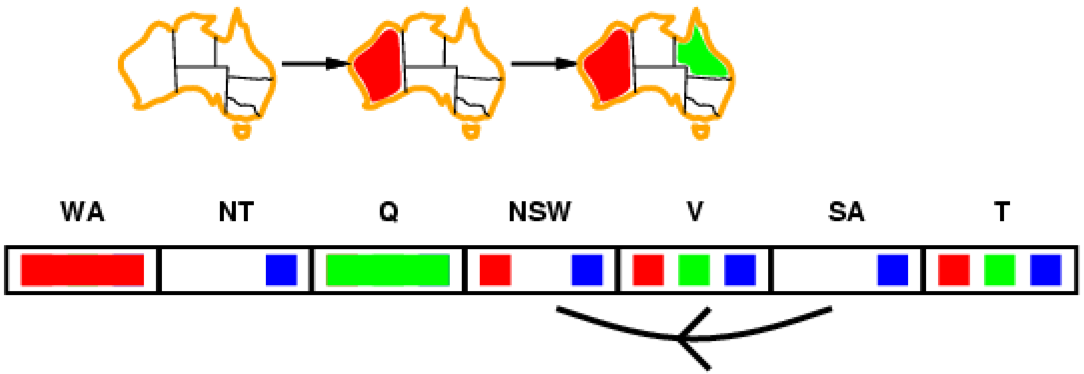
- Eliminate all the unary constraints in a CSP by running node consistency

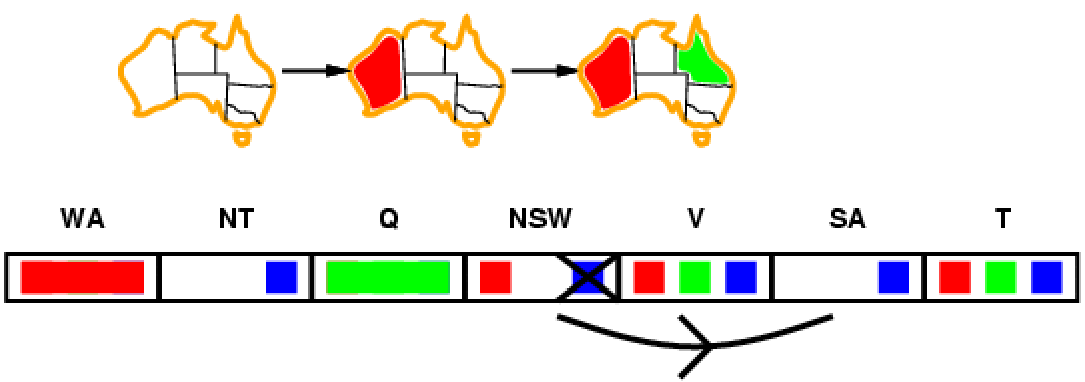
Arc consistency

- A variable in a CSP is arc-consistent if every value in its domain satisfies the variable's binary constraints.
  - E.g., $\langle (X, Y), \{(0,0), (1,1), (2,4), (3,9)\} \rangle$, where both domains are sets of digits $\rightarrow$ reduce X's domain to {0, 1, 2, 3} and Y 's domain to {0, 1, 4, 9}
- Arc consistency may have no effect in several cases.
  - E.g., the Australia map, no matter what value chosen for SA (or for WA), there is a valid value for the other variable.

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



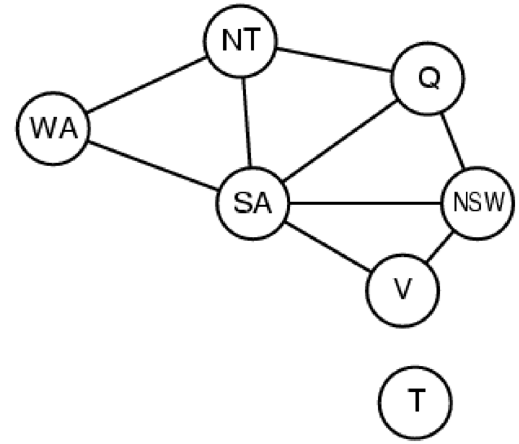


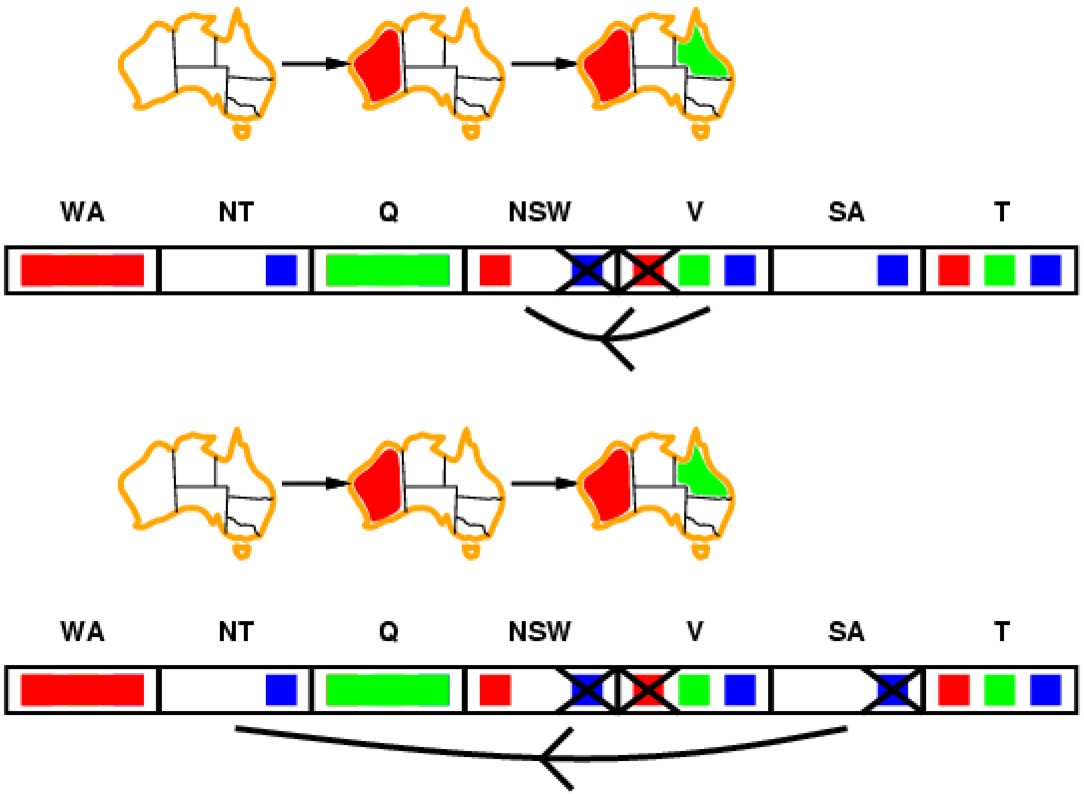


Consider state of search after WA and Q are assigned

- $SA \rightarrow NSW$ is consistent if $SA = blue$ and $NSW = red$
- $NSW \rightarrow SA$ is consistent if $NSW = red$ and $SA = blue$ $NSW = blue$ and $SA = ???$

Arc-consistency can be made by removing $blue$ from NSW





If X loses a value, neighbors of X need to be rechecked

Continue to propagate constraints

- Check $V \to NSW$
- Not consistent for $V = red \to$ remove red from V

Arc consistency detects failure earlier than forward checking

Arc consistency

- Run as a preprocessor before the search starts or after each assignment
- AC must be run repeatedly until no inconsistency remains.
- Trade-off
  - Eliminate large (inconsistent) parts of the state-space, require some overhead to do
  - Generally more effective than direct search
- Need a systematic method for arc-checking
  - If X loses a value, neighbors of X need to be rechecked $\rightarrow$ incoming arcs can become inconsistent again while outgoing arcs stay still

The 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*

The worst-case complexity is $O(cd^3)$

n: number of variables, each has domain size d, c binary constraints (arc)

The AC-3 algorithm

function REVISE(*csp*, X_i, X_j) **returns** true iff we revise the domain of X_i
  revised ← *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 ← *true*
  return *revised*

Backtracking Search

- *Backtracking Search*
- *Variable and Value Ordering*
- *Interleaving Search and Inference: Forward Checking*
- *Intelligent backtracking: Looking backward*

CSP as a Search problem

- Let's start with the straightforward approach, then fix it.
- States are defined by the values assigned so far
  - Initial state: empty assignment { }
  - Successor function: assign a value to an unassigned variable that agrees with the current assignment $\rightarrow$ fail if no legal assignments
  - Goal test: the current assignment is complete
- This is the same for all CSPs
  - Every solution appears at depth n with n variables $\rightarrow$ use depth-first (or depth-limited) search
  - Given d is the domain size, the branching factor $b = (n - l)d$ at depth l, $n! \cdot d^n$ leaves with only d^n complete assignments!

Backtracking search

- Variable assignments are commutative.
  - E.g., [$WA = red$ then $NT = green$] = [$NT = green$ then $WA = red$]
- Only need to consider assignments to a single variable at each node $\rightarrow$ branching factor $b = d$, d^n leaves
- Depth-first search: choose values for one variable at a time and backtrack when a variable has no legal values left

Backtracking search

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

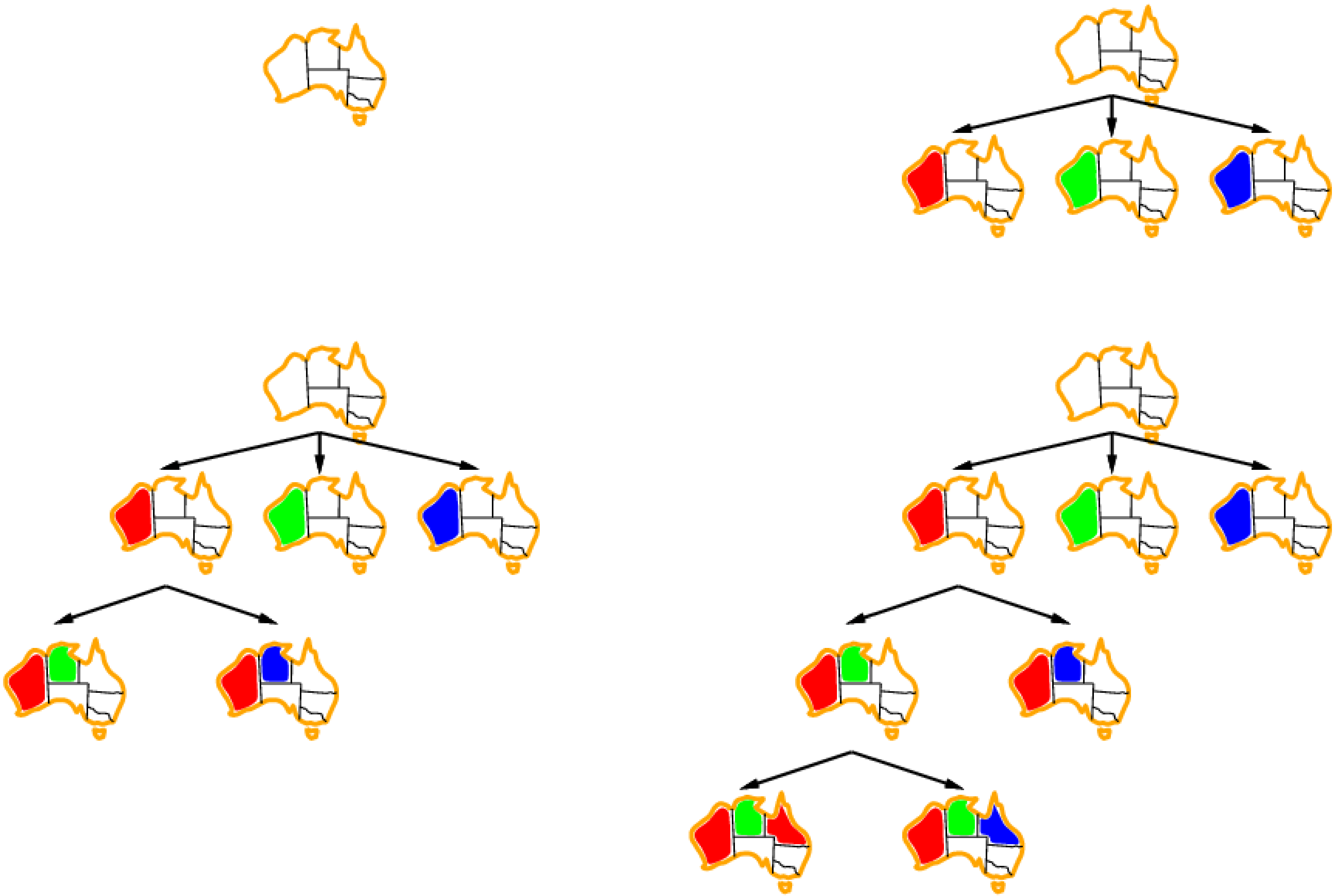
function BACKTRACK(*assignment, csp*) **returns** a solution, or failure
  if *assignment* is complete **then return** *assignment*
  var ← SELECT-UNASSIGNED-VARIABLE(*csp*)
  for each *value* **in** ORDER-DOMAIN-VALUES(*var, assignment, csp*) **do**
    if *value* is consistent with *assignment* **then**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

Which variable should be assigned next?

In what order should its values be tried?

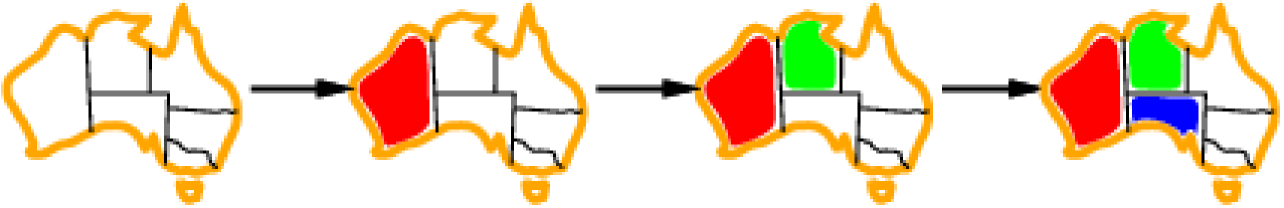
What inferences should be performed?

Backtracking search: An example



Variable and value ordering

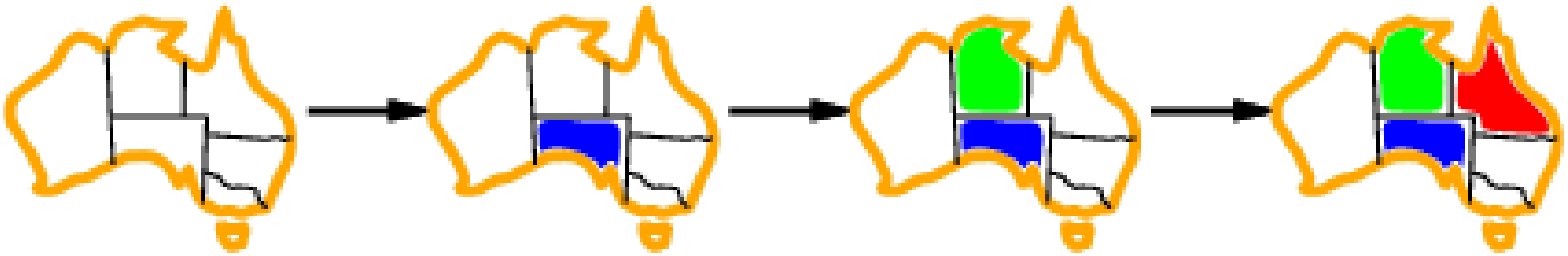
- **Minimum-remaining-values** (MRV) **heuristic:** choose the variable with the fewest legal values
  - E.g., after $[WA = red, NT = green]$ only one possible value for SA



  - Failure will be detected immediately, avoiding pointless searches
- MRV usually performs better than a random/static ordering, sometimes by a factor of 1,000 or more.

Variable and value ordering

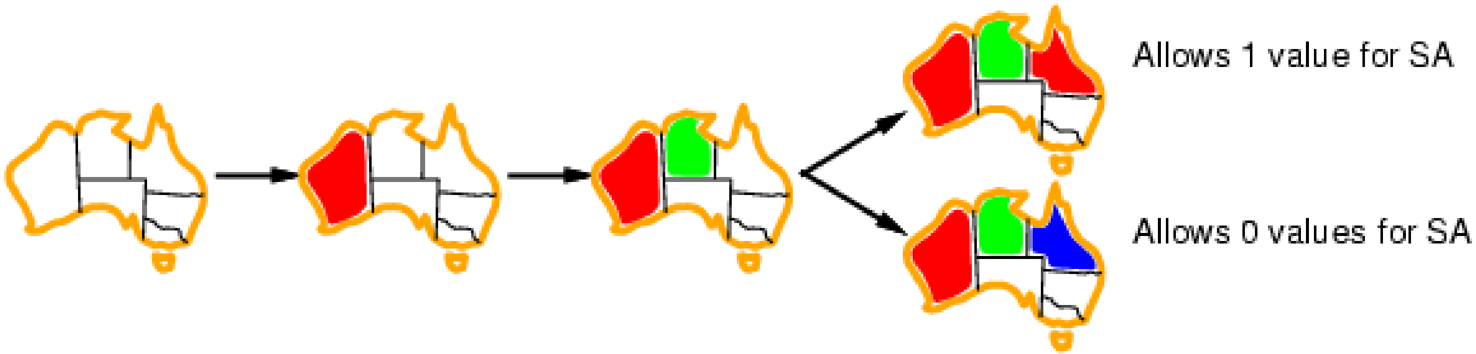
- **Degree heuristic** (DH)**:** choose the variable that involves in the largest number of constraints on other unassigned variables
  - E.g., SA has a highest degree of 5, other variables except T have degrees of 2 or 3.



- DH is the tie-breaker among most constrained variables

Variable and value ordering

- **Least constraining value** (LCV) **heuristic:** given a variable, choose the value that leaves the maximum flexibility for subsequent variable assignments

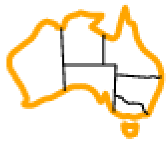


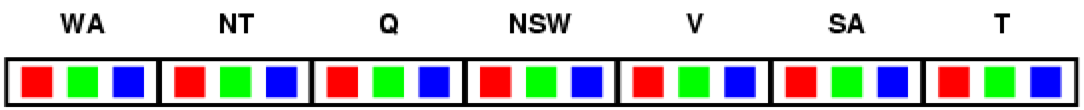
- Combining the three heuristics makes 1000 queens feasible

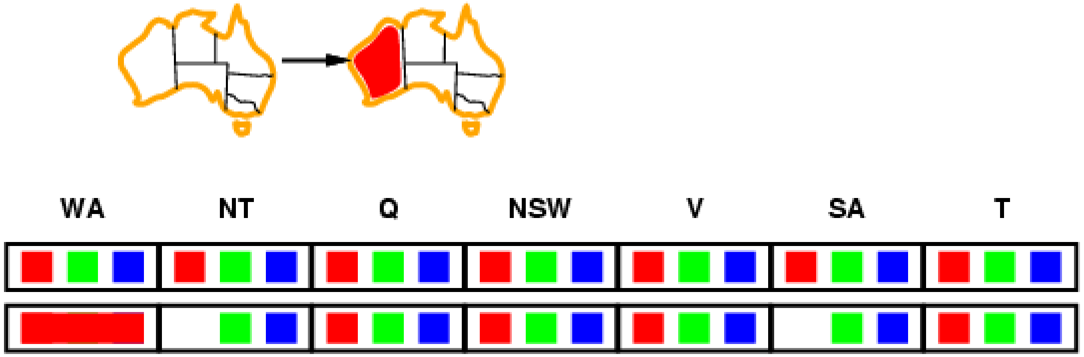
Why should variable selection be fail-first,
but value selection be fail-last?

Inference: Forward checking

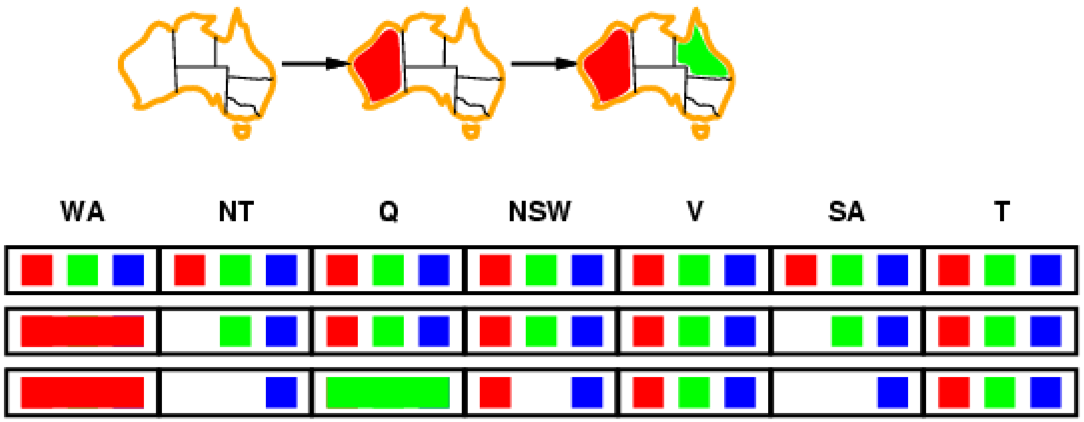
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values





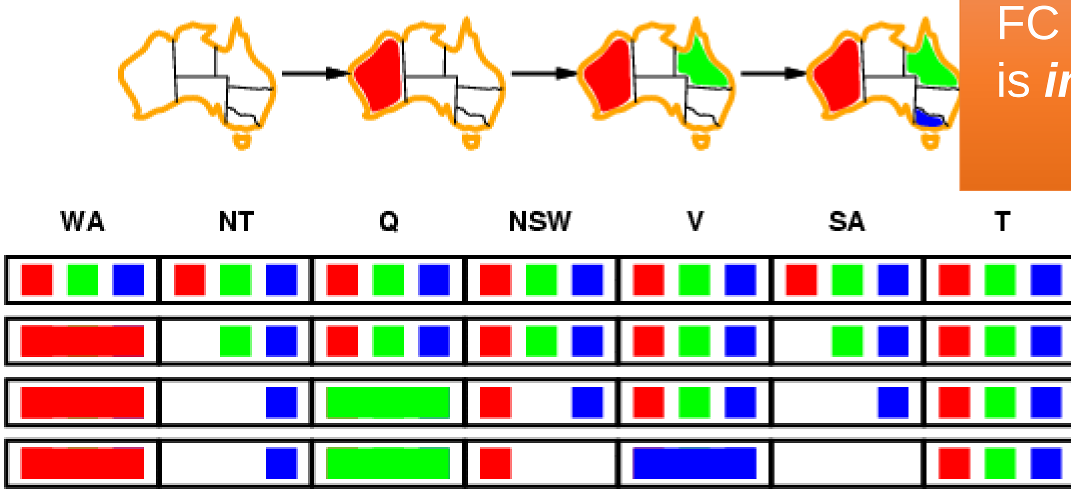


- ✓ Assign $\{WA = red\}$
- ✓ Effects on other variables connected by constraints to WA
  - – NT *can no longer be red*
  - – SA *can no longer be red*



- ✓ Assign $\{Q = green\}$
- ✓ Effects on other variables connected by constraints to Q
  - NT *can no longer be green*
  - SA *can no longer be green*
  - NSW *can no longer be green*

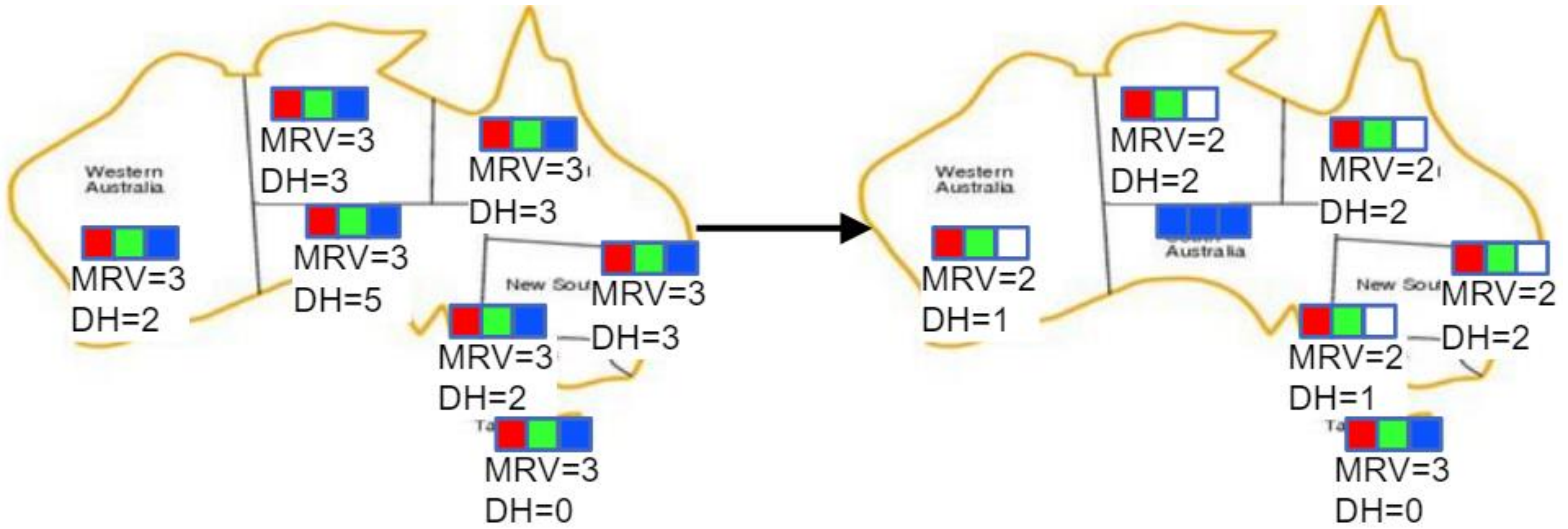
FC has detected that partial assignment is ***inconsistent*** with the constraints and backtracking can occur.



- ✓ Assign $\{V = blue\}$
- ✓ Effects on other variables connected by constraints to V
  - NSW *can no longer be blue*
  - SA *is empty*

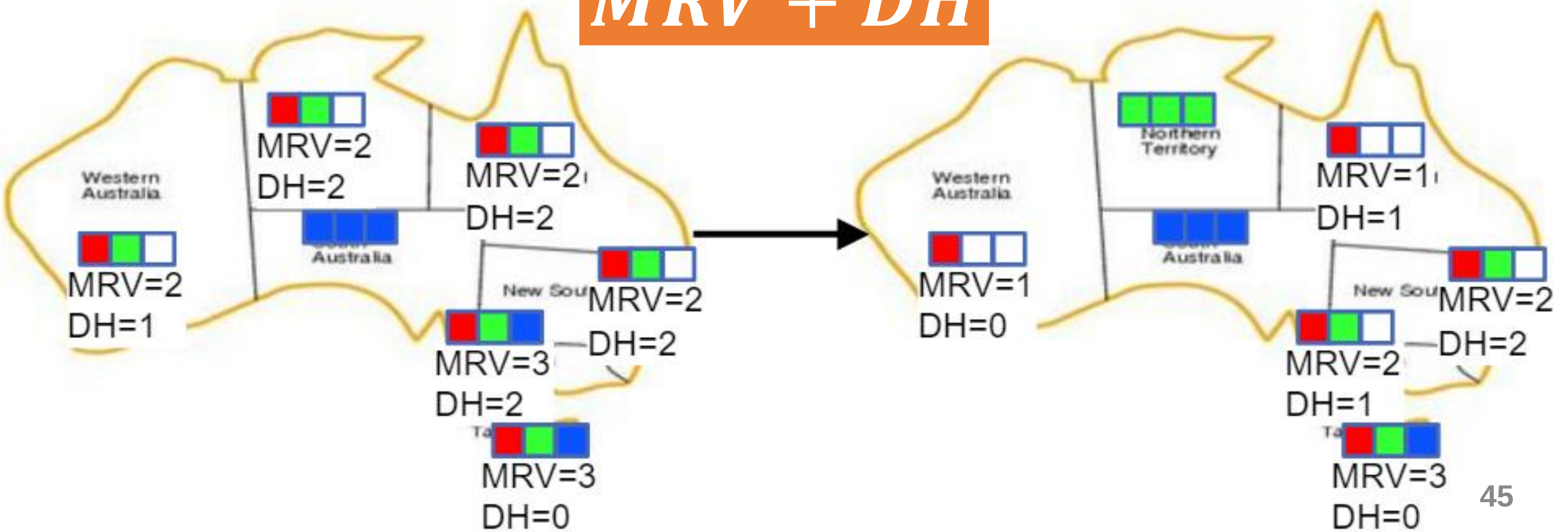
Inference: Forward checking

- MRV heuristic + forward checking $\rightarrow$ more effective search
- It can detect many inconsistencies but not all of them.
  - Make only the current variable arc-consistent, but do not look ahead and make all the other variables arc-consistent
- Forward checking vs. Arc consistency
  - Given a constraint C_{XY} between two variables X and Y, for any value of X, there is a consistent value that can be chosen for Y such that C_{XY} is satisfied, and vice versa.
  - Arc consistency is **directed** and is checked in both directions for two connected variables $\rightarrow$ stronger than forward checking

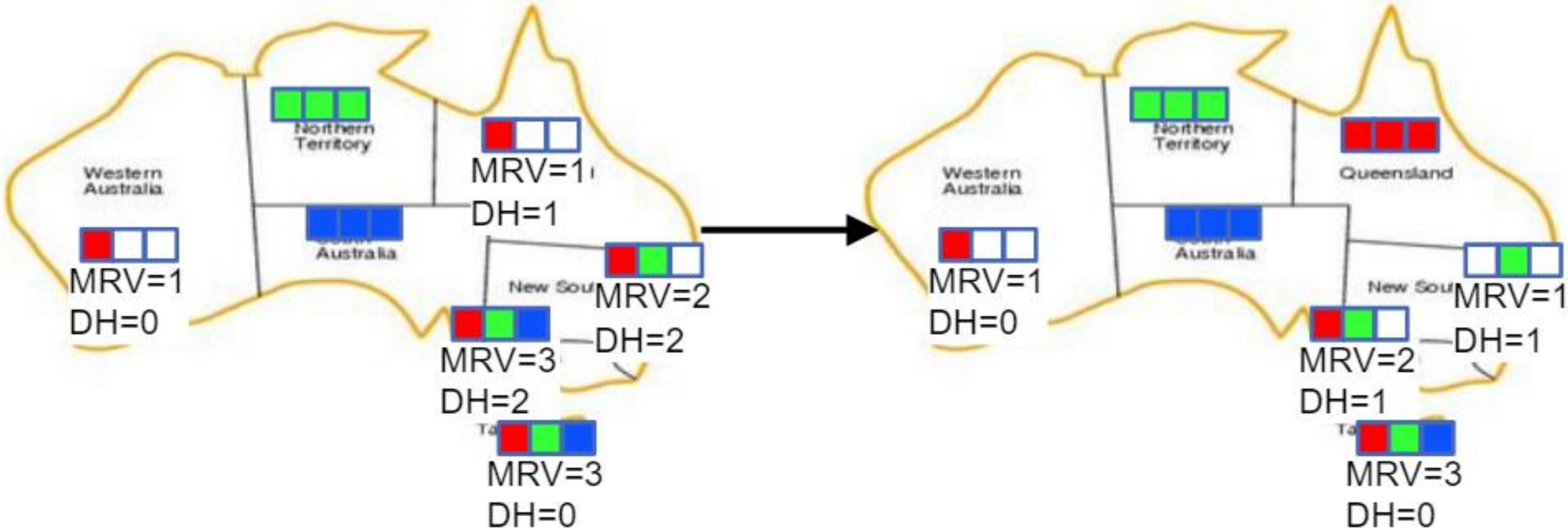


Western Australia
MRV=3
DH=2
MRV=3
DH=3
MRV=3
DH=5
MRV=3
DH=3
New Sou
MRV=3
DH=3
MRV=3
DH=2
MRV=3
DH=0
Western Australia
MRV=2
DH=1
MRV=2
DH=2
Australia
MRV=2
DH=2
New Sou
MRV=2
DH=2
MRV=2
DH=1
MRV=3
DH=0

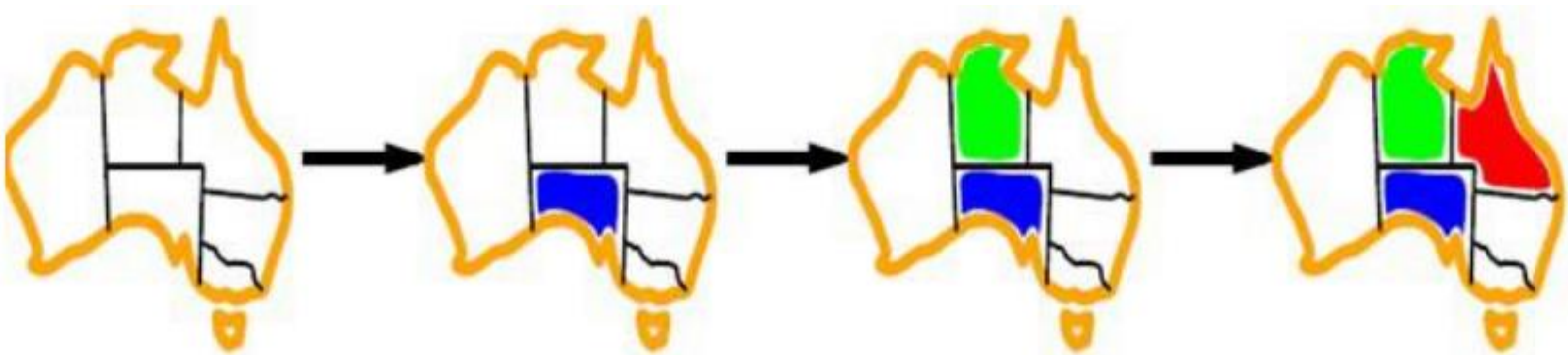
$MRV + DH$



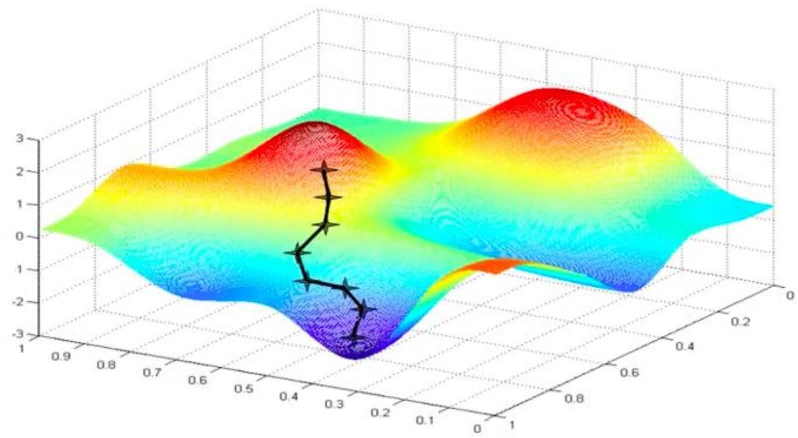
Western Australia
MRV=2
DH=1
MRV=2
DH=2
Australia
MRV=2
DH=2
New Sou
MRV=2
DH=2
MRV=3
DH=2
MRV=3
DH=0
Western Australia
MRV=1
DH=0
Northern Territory
Australia
MRV=1
DH=1
New Sou
MRV=2
DH=2
MRV=2
DH=1
MRV=3
DH=0


Northern Territory
Western Australia
MRV=1
DH=1
MRV=1
DH=0
Australia
New Sou
MRV=2
DH=2
MRV=3
DH=2
MRV=3
DH=0
Northern Territory
Queensland
Western Australia
MRV=1
DH=0
Australia
New Sou
MRV=1
DH=1
MRV=2
DH=1
MRV=3
DH=0

$MRV + DH$



Local Search for CSPs



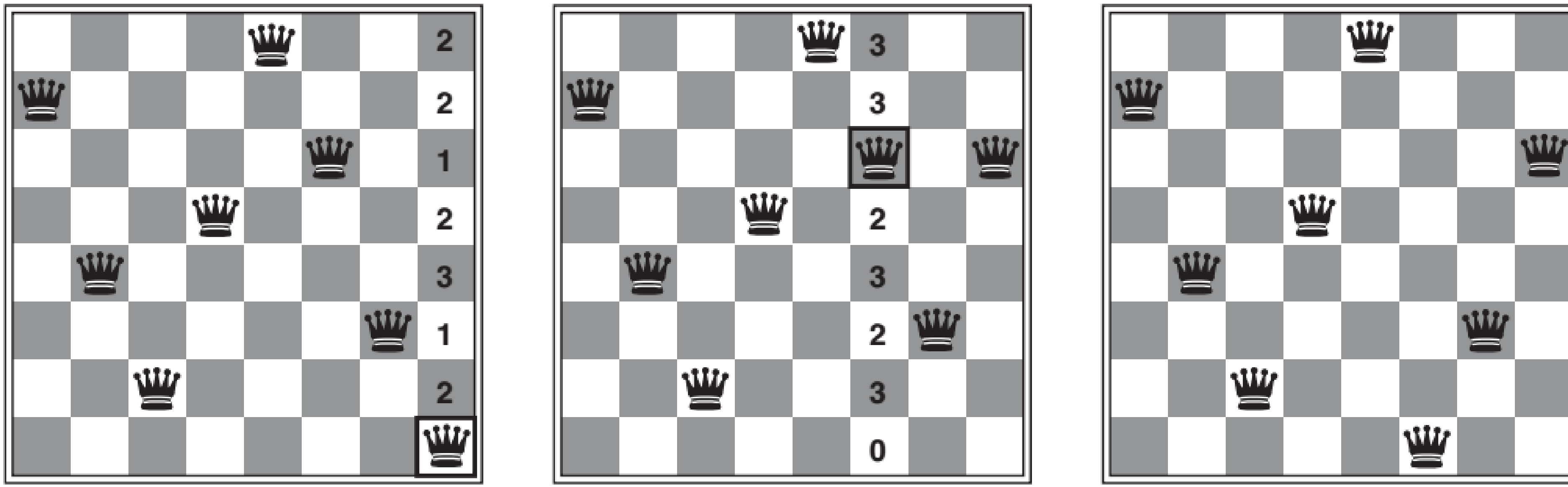
Local search for CSPs

- Complete-state formulation
  - The initial state assigns a value to every variable $\rightarrow$ violation
  - The search changes the value of one variable at a time $\rightarrow$ eliminate the violated constraints
- ***Min-conflicts heuristic:*** *the minimum number of conflicts with other variables*
- Min-conflicts is surprisingly effective for many CSPs.
  - Million-queens problem can be solved ~ 50 steps
  - Hubble Space Telescope: the time taken to schedule a week of observations down from 3 weeks (!) to ~10 minutes

MIN-CONFLICTS algorithm

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

MIN-CONFLICTS: 8-queens



A two-step solution using min-conflicts for an 8-queens problem.
At each stage, a queen is chosen for reassignment in its column.
The number attacking queens (i.e. conflicts) is shown in each square.
The algorithm moves the queen to the min-conflicts square, breaking ties randomly.

Local search for CSPs

- The landscape of a CSP under the min-conflicts heuristic usually has a series of plateaux.
  - Millions of variable assignments that are only one conflict away from a solution
- **Plateau search:** allow sideways moves to another state with the same score
- **Tabu search:** keep a small list of recently visited states and forbid the algorithm to return to those states
- **Simulated annealing** can also be used

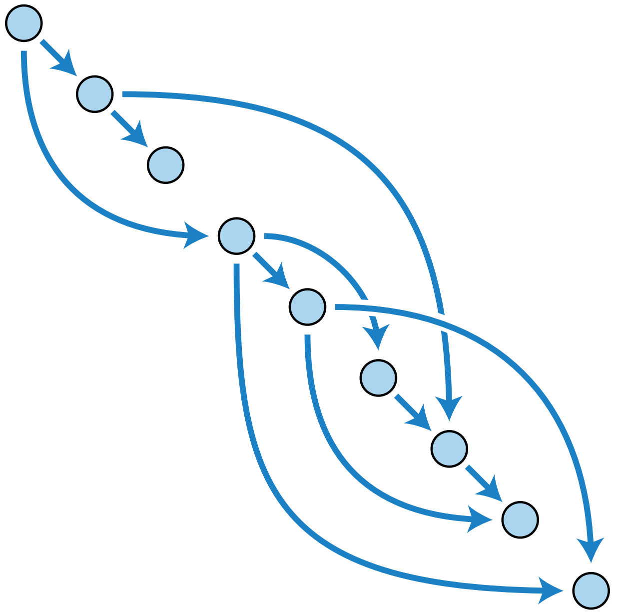
Constraint weighting

- Concentrate the search on the important constraints
- Each constraint is given a numeric weight, W_i, initially all 1.
- At each step, choose a variable/value pair to change that has the lowest total weight of all violated constraints
- Increase the weight of each constraint that is violated by the current assignment

Local search in online setting

- Scheduling problems: online setting
  - A week's airline schedule may involve thousands of flights and tens of thousands of personnel assignments
  - The bad weather at one airport can render the schedule infeasible.
- The schedule should be repaired with a minimum number of changes.
  - Done easily with a local search starting from the current schedule
  - A backtracking search with the new set of constraints usually requires much more time and might find a solution with many changes from the current schedule

The Structure of Problems



Independent subproblems

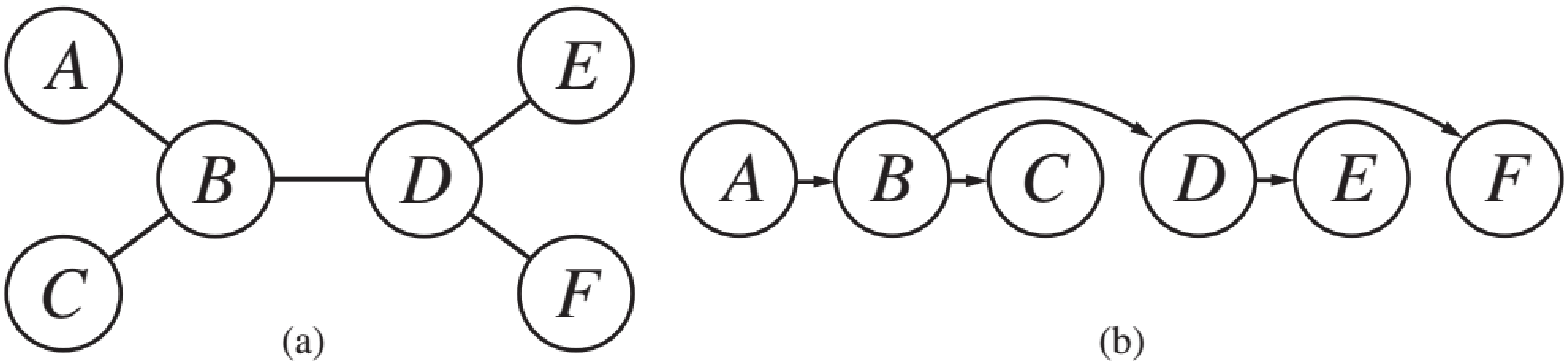
- *If assignment S_i is a solution of CSP_i, then $\bigcup_i S_i$ is a solution of $\bigcup_i CSP_i$.*
- Suppose each CSP_i has c variables from the total of n variables. Then there are n/c subproblems, each of which takes at most d^c work to solve.
  - where c is a constant and d is the size of the domain.
- Hence, the total work is $O(d^c n/c)$, which is linear in n.
  - Without the decomposition, the total work is $O(d^n)$.
- For example, the Australia map coloring: Tasmania and the mainland

Tree-structured CSP

- A constraint graph is a tree when any two variables are connected by only one path.
- *Any tree-structured CSP can be solved in time linear in the number of variables*
- **Directed arc consistency** (DAC): A CSP is directed arc-consistent under an ordering of variables $X_1, X_2, \dots, X_n$ iff every X_i is arc-consistent with each X_j for $j > i$.

Tree-structured CSP

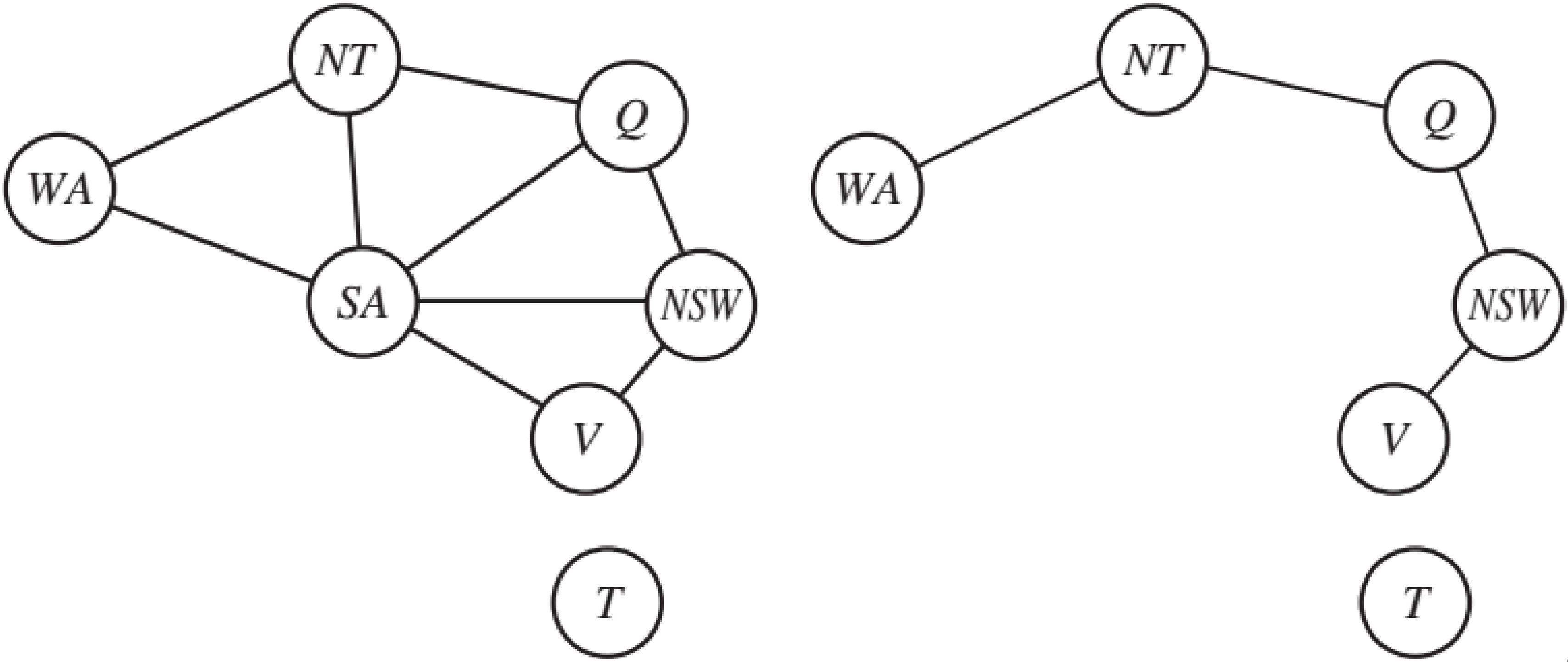
- **Topological sort:** first pick any variable to be the root of the tree and choose an ordering of the variables such that each variable appears after its parent in the tree.



(a) The constraint graph of a tree-structured CSP.
(b) A linear ordering of the variables consistent with the tree with A as the root.

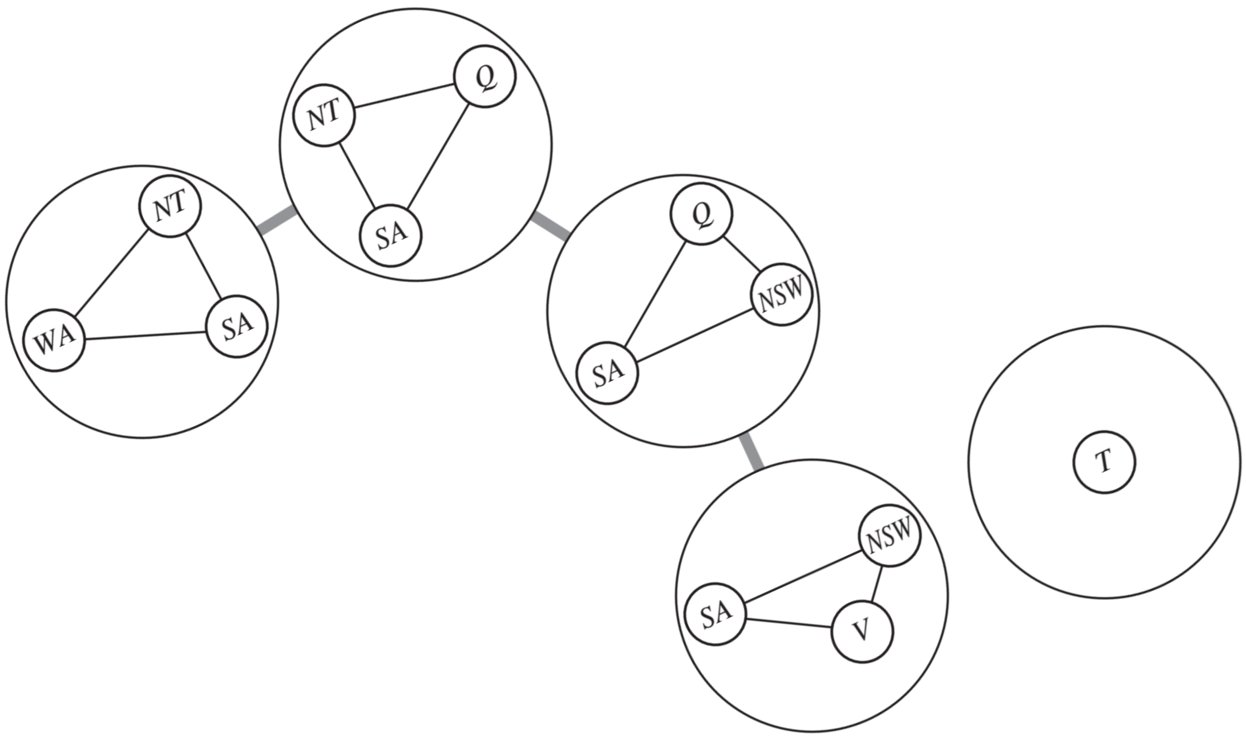
Reducing graphs to trees

- Assign values to some variables so that the remaining variables form a tree
  - E.g., fix a value for SA and delete from other variables' domains any values that are inconsistent with the value chosen for SA



Reducing graphs to trees

- Tree decomposition of the constraint graph into a set of connected subproblems
- Each subproblem is solved independently and the resulting solutions are then combined.



The structure of values

- Consider the map-coloring problem with n colors.
- For every consistent solution, there is a set of $n!$ solutions formed by permuting the color names.
  - E.g., WA, NT, and SA must all have different colors, but there are 3! ways to assign the three colors to these three regions.
- Symmetry-breaking constraint: Impose an arbitrary ordering constraint that requires the values to be in alphabetical order
  - E.g., $NT < SA < WA \rightarrow$ only one solution possible: $\{NT = blue, SA = green, WA = red\}$

THE END