3
Constraint Satisfaction Problem
(CSP)

⇒ Assumption about the world:
- → A single agent
- → Deterministic action
- → Fully observed state
- → Discrete state space

Planning : Sequence of action
- → Path to the goal is the important thing
- → Paths have various costs, depths
- → Heuristics give problem-specific guidance

Identification : Assignments to variables
- → The goal itself is important, not the path
- → All paths at the same depth

  (for some formulations)
- → CSPs are a specialized class of identification problems.

Standard Search Problem	Constraint Satisfaction (CSP) Problem
→ State is a "black box": arbitrary data structure	→ A special subset of search Problem
→ Goal test can be any function over states	→ State is defined by Variable X_i with value from Domain D/D_i.
→ Successor function can also be anything.	→ Goal test is a set of constraints specifying allowable combinations of values for subset of variables.

* Ordering

Minimum Remaining Values (MRV)

⇒ Choose the variable with the fewest legal left values in its domain.

⇒ Also called fail fast Ordering.

Least Constraining Value

⇒ Given a choice of variable, choose the least Constraining value.

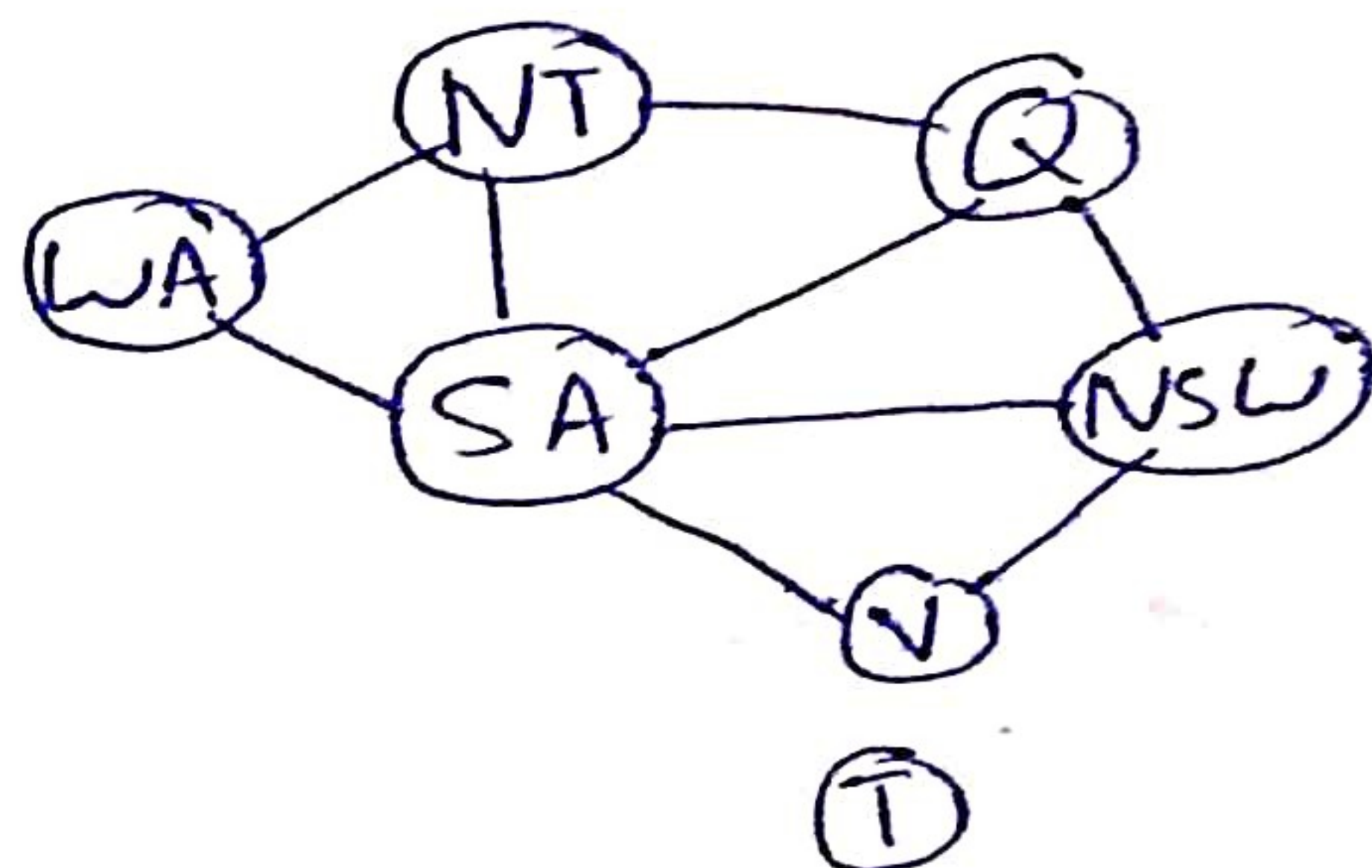   ↳ i.e. the One that rules out the fewest values in the remaining variables.
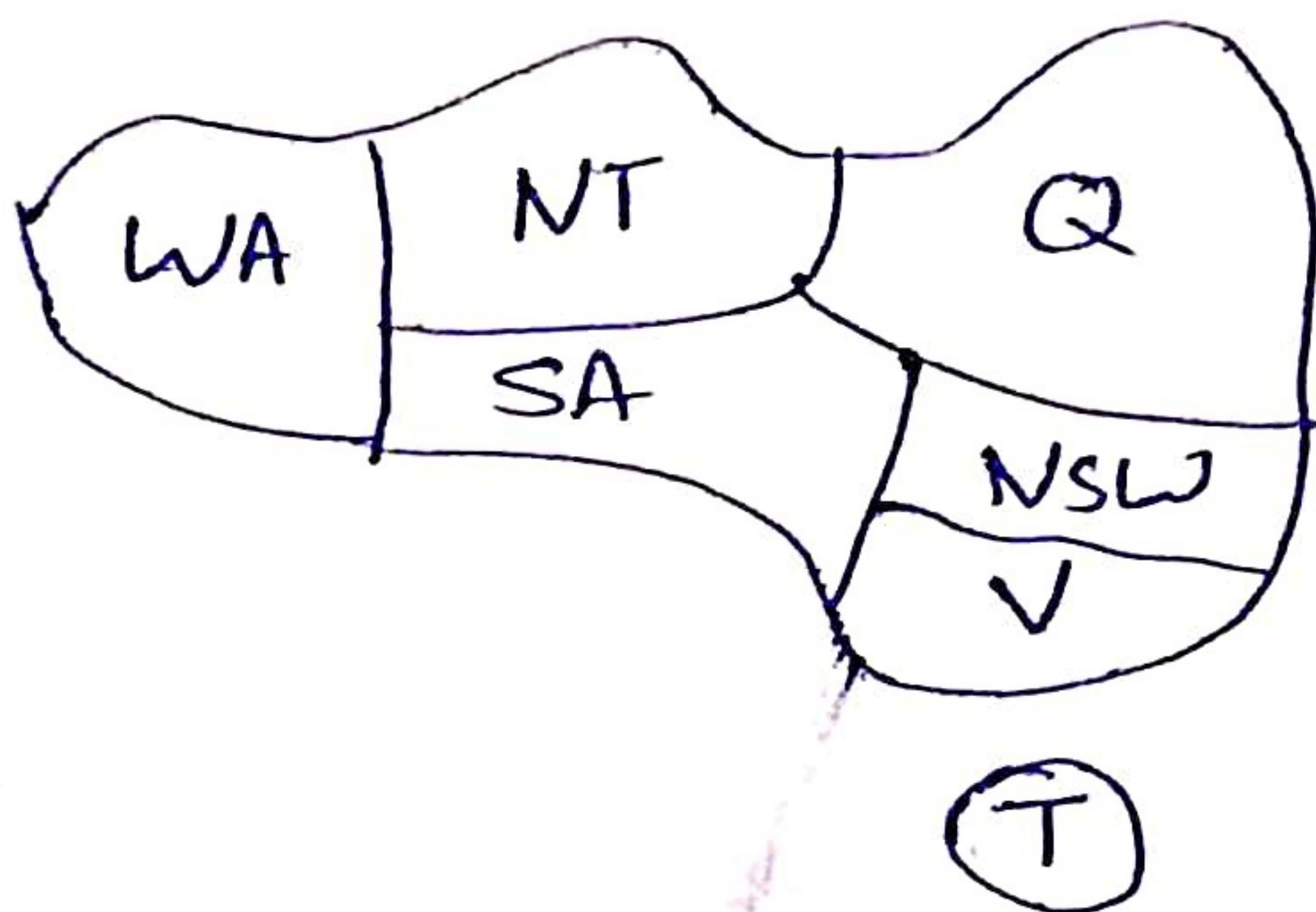
* Example 1: Map Colouring (For Australia)

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

(States of Australia)

⇒ **Domains:** $D = \{red, green, blue\}$

⇒ **Constraints:** Adjacent regions must have different colour.

Constraint Graphs

⇒ **Solution:** An assignment satisfying all constraints.
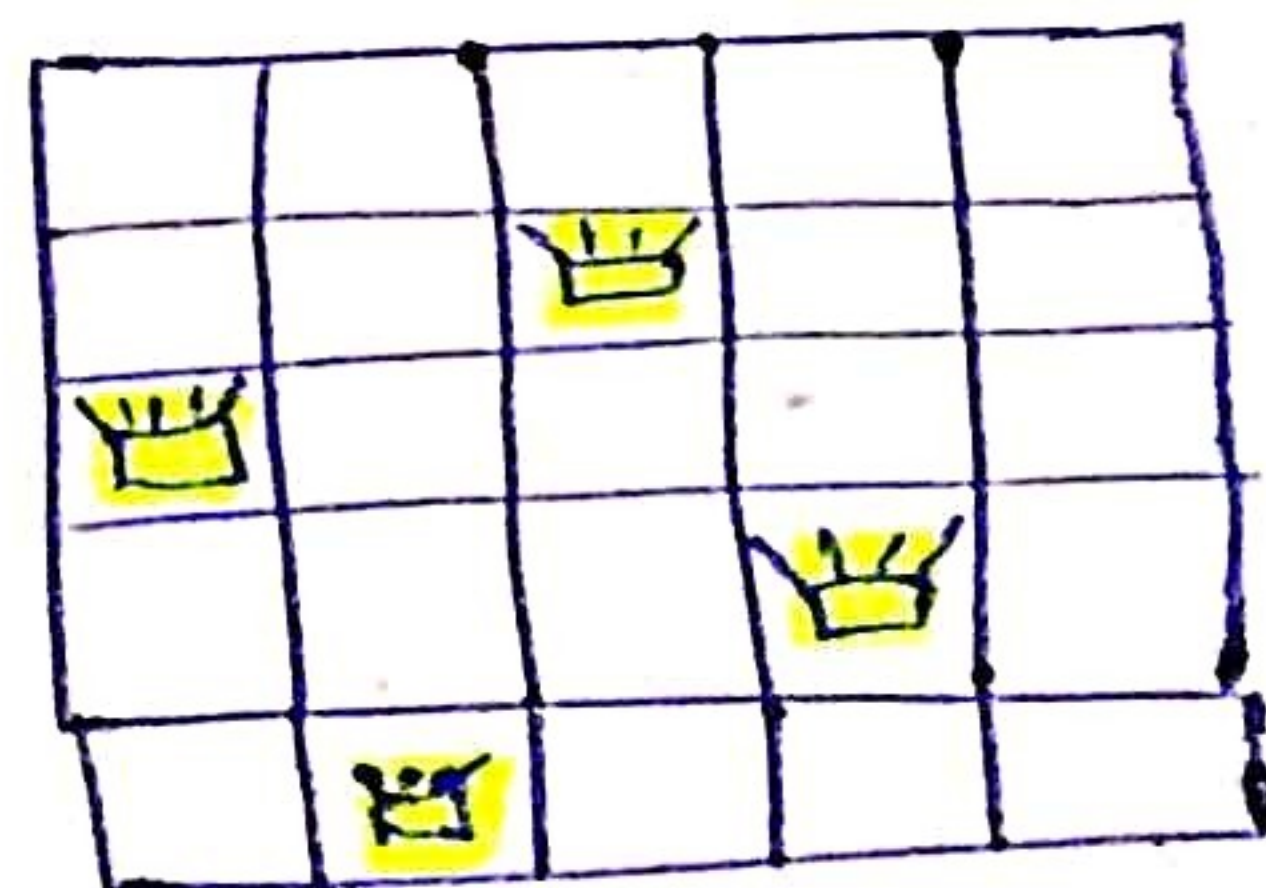
* Example 2: N-Queens

Variables: X_{ij}

Domains: $\{0, 1\}$

Constraints:

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

(2) Queens are non threading

✱ Constraint Graphs

⟹ <u>Binary CSP</u>: Each constraint relates (at most) two variables.

⟹ <u>Binary Constraint graph</u>
  - nodes are variables
  - arcs shows constraints

⟹ General-purpose CSP algorithms use the graph structure to speed up search.

✱ Examples: Sudoku

<u>Variables</u>: Each open square

<u>Domain</u>: $\{1, 2, \ldots, 9\}$

<u>Constraints</u>:
  - 9-way alldiff for each Column
  - 9-way alldiff for each row
  - 9-way alldiff for each region

★ Varieties of CSPs

⊞ <u>Discrete Variables</u>

  ① Finite domain
       - d means ⓐ $O(d^n)$ Complete assignments
  ② Infinite domain (Integers, strings etc.)

⊞ <u>Continuous Variables</u>

* Varieties of Constraints

- → Unary constraints involve a single variable
- → Binary constraints involve pairs of variables
- → Higher-order constraints involve 3 or more variables.

* Preference (Soft Constraints)

- → Example: red is better than green
- → Often represented by a cost for each variable assignment
- → Gives constrained optimization problem.

Solving CSPs

★ Standard Search Formulation

⇒ States defined by the values assigned so far (Partial assignments)

<mark>Initial state</mark> : The empty assignment, { }

<mark>Successor function</mark>: Assign a value to an unassigned variable.

<mark>Goal test</mark>: the current assignment is Complete & Satisfies all Constraints.

★ Backtracking Search

⇒ It is the basic uninformed algorithm for Solving CSPs

Idea 1: One variable at a time

Idea 2: Check Constraints as you go

↳ Consider only values which do not conflict with the previous assignments

⇒ Depth-first Search with these two improvement is called backtracking Search.

function BACKTRACKING-SEARCH (CSP) return sol/fail

1. return RERCURSIVE-BACKTRACKING ({}, CSP)

function RECURSIVE-BACKTRACKING (assignment, CSP)
return sol/fail

1. if assignment is complete then return assignment

2. var ← SELECT-UNASSIGNED-VARIABLE (VARIABLES [CSP]
, assignment
, CSP
)

3. for each value in ORDER-DOMAIN-VALUE do
(var, assignment, CSP)

4. if value is consistent with assignment
given CONSTRAINTS [csp] then

5. add {var = value} to assignment

6. result ← RECURSIVE-BACKTRACKING
(assignment, CSP)

7. if result ≠ failure then return result

8. remove {var = value} from assignment

9. return failure

★ Improve Backtracking

⇒ General-purpose Idea give huge gains in speed.

① Odering

    ↳ Which variable should be assigned next

② Filtering

    ↳ Can we detect inevitable failure early

③ Structure

    ↳ Can we exploite the problem structure

✱ Filtering | Keep track of domains of unassigned Variables and Cross off bad options

⇒ Filtering is about ruling out candidates.

Filtering: Forward Checking

⇒ Cross off values that violate a Constraint when added to the existing assignment.

Filtering: Constraint Propagation

⇒ Reason from Constraint to Constraint.

✱ Consistency of A Single Arc

⇒ An arc $X \rightarrow Y$ is Consistent iff for every X in the tail there is Some y in the head which could be assigned without violating a Constraint.

<u>Forward Checking</u>: Enforcing Consistency of arcs Pointing to each new assignment.

⇒ After enforcing arc Consistency:

① ➝ Can have One Solution left

② ➝ Can have multiple solution left

③ ➝ Can have no solution left

* Enforcing Arc Consistency in a CSP

function AC-3 (csp) return the CSP, possibly with reduced domains

inputs: csp, a binary CSP with variables $\{X_1, X_2 \dots X_n\}$

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 REMOVE-INCONSISTENT-VALUE (X_i, X_j) then

        for each X_k in NEIGHBORS$[X_i]$ do

        add (X_k, X_i) to queue

---

function REMOVE-INCONSISTENT-VALUES (X_i, X_j)
    returns true iff succeeds.

removed $\leftarrow$ false

for each x in DOMAIN$[X_i]$ do

    if no value y in DOMAIN$[X_j]$ allows (x, y) to
        satisfy the constraint $X_i \leftrightarrow X_j$

    then delete x from DOMAIN $[X_i]$; removed $\leftarrow$ true

return removed

Runtime: $O(n^2 d^3)$, can be reduced to $O(n^2 d^2)$