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

→ State is a "black box": arbitrary data structure

→ Goal test can be any function over states

→ Successor function can also be anything.

Constraint Satisfaction Problem (CSP)

→ A special subset of search Problems.

→ State is defined by variable X_i with value from Domain D/D_i.

→ Goal test is a set of constraints specifying allowable combinations of values for subset of variables.

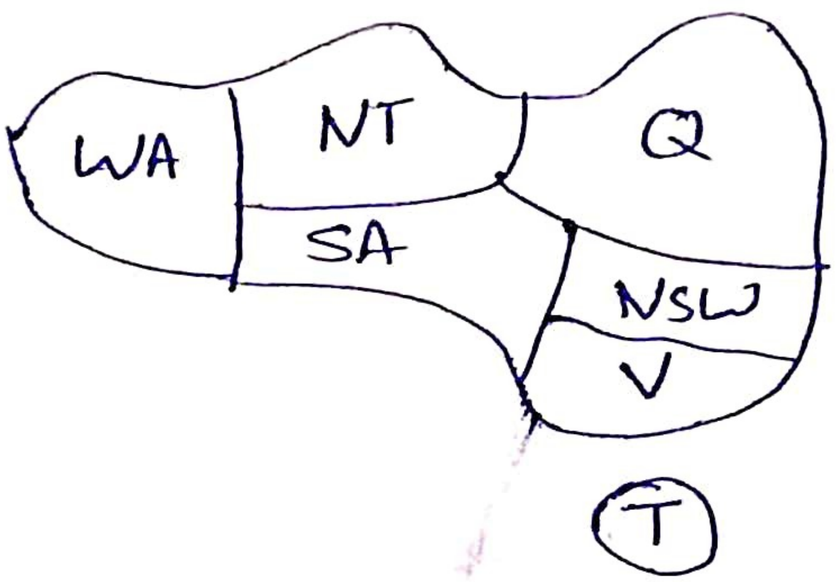
★ Example 1: Map Coloring {For Australia}

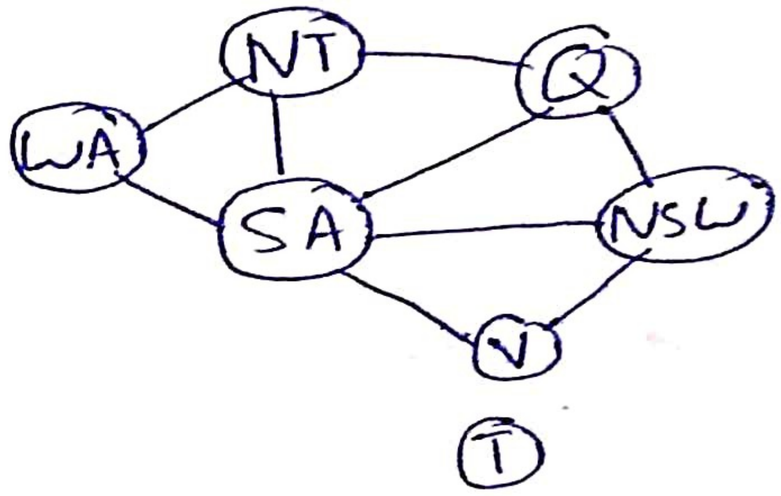
⇒ Variables: WA, NT, Q, NSW, V, SA, T
{States of Australia}

⇒ Domains: D = {red, green, blue}

⇒ Constraints: Adjacent regions must have different color.

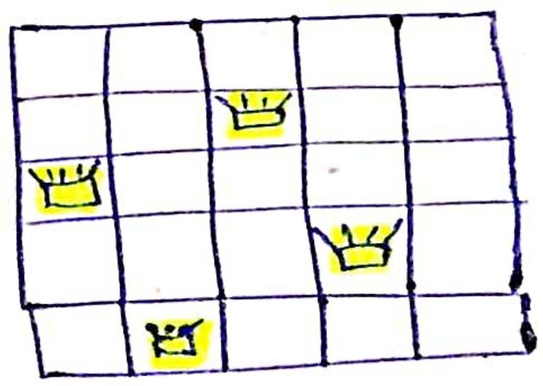
Constraint Graphs





⇒ 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-attacking

★ Constraint Graphs

⇒ **Binary CSP:** Each constraint relates (atmost) two variables.

⇒ **Binary Constraint graph**

- nodes are variables
- arcs shows constraints

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

★ Example: Sudoku

Variables: Each Open Square

Domain: {1, 2, ... 9}

Constraints:

- 9-way alldiff for each Column
- 9-way alldiff for each row
- 9-way alldiff for each region

★ Varieties of CSPs

⊕ Discrete Variables

① Finite domain
   - ↳ d means ● $O(d^n)$ Complete assignments

② Infinite domain (Integers, strings etc.)

⊕ Continuous Variables

★ Variaties 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)

Initial state: The empty assignment, { }

Succesor function: Assign a value to an unassigned variable.

Goal test: 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 RECURSIVE-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 (var, assignment, CSP) do
4. if value is consistant 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 Ordering
   ↳ 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 $X \to Y$ is consistent iff for every x in the tail there is some y in the head which could be assigned without violating a constraint.

Forward Checking: 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) returns 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 $O(n^2 d^2)$

★ 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.