Constraint Satisfaction Problem

03/03/2025

Koustav Rudra

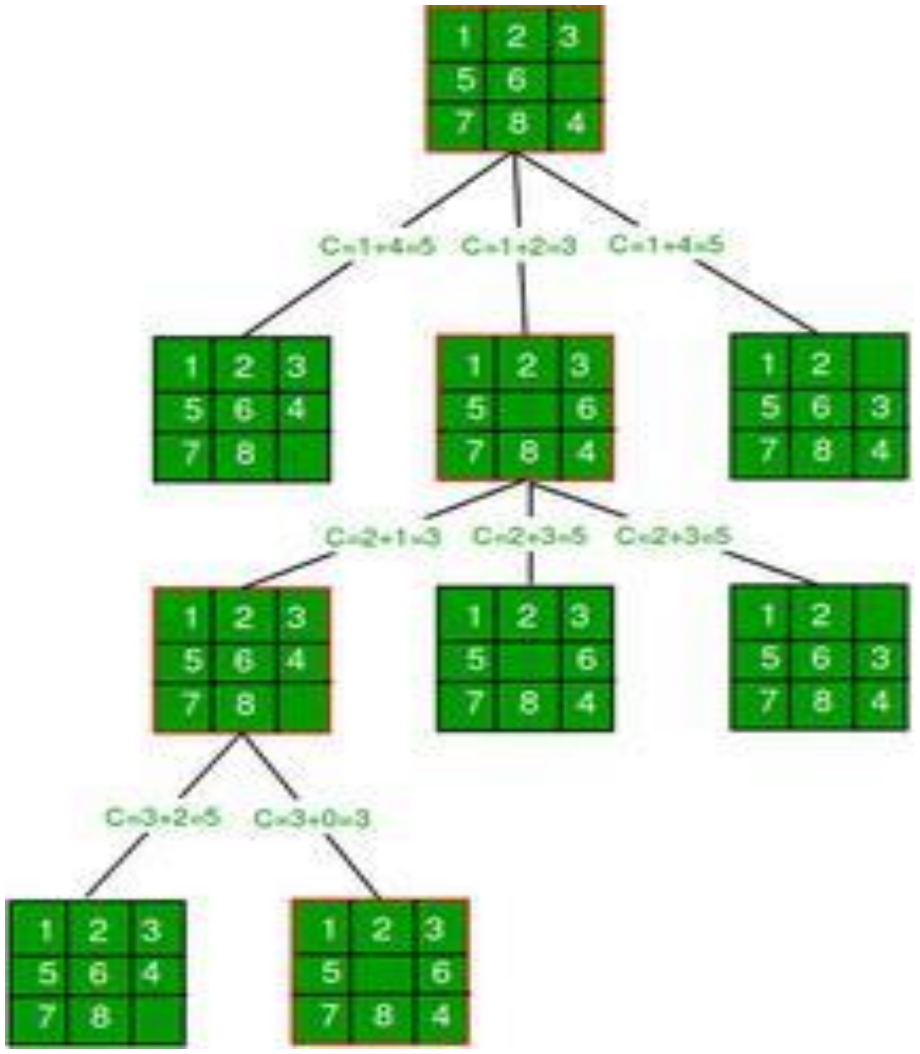
Objective

- Problem Formulation
- Problem representation
- Solvers

AI Problem Solvers: Evolution

```
def solve(State state):
        ……………………………………
        ……………………………………
        move(c1, c2)
        check(solution)
        ……………………………………
        ……………………………………
```

Brute-Force Approach



Problems:

- Very much problem specific
- Solution developed for one problem will not work for others

AI Problem Solvers: Evolution

```
def solve(State state):
        ………………………………………
        ………………………………………
        state.isGoal()
                return true
        succ = state.successor()
        ………………………………………
        ………………………………………
```

Search Algorithms

- **Overall Structure:** Problem Agnostic
- Still isGoal and successor are problem specific

- Can we have Truly Generic Problem Solvers?

- Yes, but for specific class of problems
    - Constraint Satisfaction Problems

- What are the implications?
    - Make isGoal and successor are problem agnostic
    - Design methods and heuristics: problem agnostic

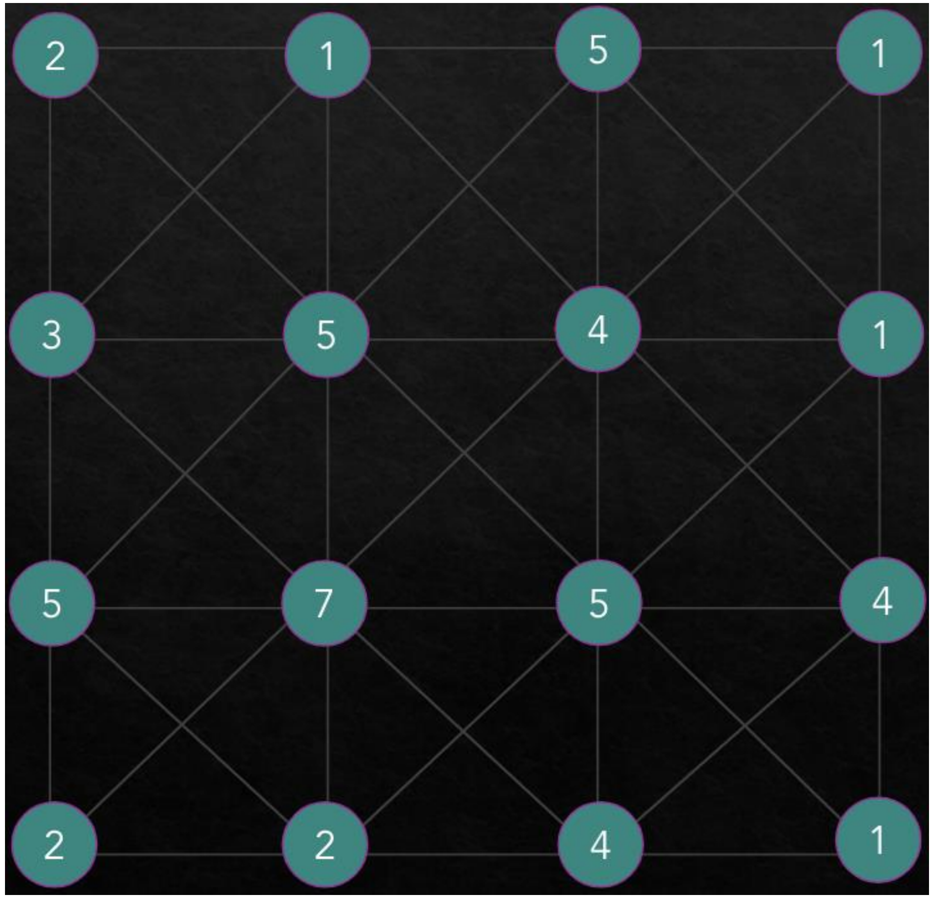
Revisiting Search Problems

- The world
  - Single agent, deterministic action, fully observable, discrete state

- Planning a sequence of actions
  - Important: Path to goal
  - Paths: varying costs and depths
  - Heuristics to reduce search space

- Identification of goal
  - Goal is important not path
  - All paths are at same depth
  - CSPs are identification problems

Example



Search Formulation:

1. Initial state: Nodes with no connection
2. Successor Function
    1. Add any one edge
    2. Next state: Resultant graph
3. Goal Test
    1. Whether each node has degree equal to the no. attached to the node

Path to Goal important?

Or

Configuration that satisfy certain criteria?

Constraint: Number of outgoing edges

Assignment: On/Off

Jointly all the assignments make sense or not

Combinatorial problem

Goal Identification Problem

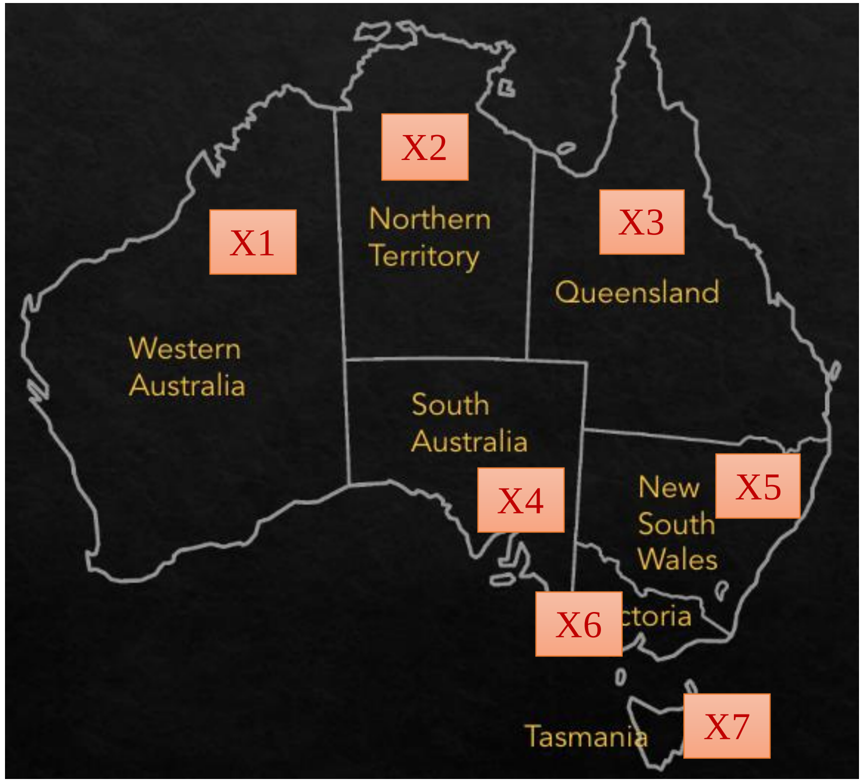
Can we define domain independent methods to solve the problem?

Constraint Satisfaction Problems

- Standard search problems:
  - State is problem independent ➔ Arbitrary data structures
  - Goal test: Function of state
    - Problem dependent
  - Successor: Function of state
    - Problem dependent

- Constraint Satisfaction Problems
  - Subset of search problems [Identification Problem]
  - State: $<Xi,Di>_N$
  - Goal Test: A set of constraints
    - $C1 \wedge C2 \ldots \wedge Cn$
    - Legal combination of values for subset of variables

Constraint Satisfaction Problems



Map Coloring Problem

- No two adjacent states have same color

CSPs: Formulation

- CSPs Problem: <X, D, C>

- State: X ➔ set of variables, Domain(Xi) = Di
  - X = $\{X_1, X_2, \ldots, X_n\}$
  - D = $\{D_1, D_2, \ldots, D_n\}$
- Goal Test: Set of constraints C
  - Ci = f(X') where $X' \subseteq$ X

- Constraint Definition
  - A pair <scope, rel>
  - Scope defines the variables
  - Relation describes interaction among variables in scope

- Example: X_1 and X_2 have domain {A, B}
  - Constraints: <(X_1,X_2), [(A,B), (B,A)]> [Explicit]
  - Constraints: <(X_1,X_2), $X_1 \neq X_2$ > [Implicit]

CSPs: Formulation

- Solution
    - Assignment: Assigning values to some or all variables
    - Consistent Assignment: Does not violate any constraint
    - Complete Assignment: Every variable is assigned a value
    - Solution: Consistent and Complete Assignment

- General purpose algorithms with more power than standard search algorithms

Example: Sudoku

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

Variables: Each open square

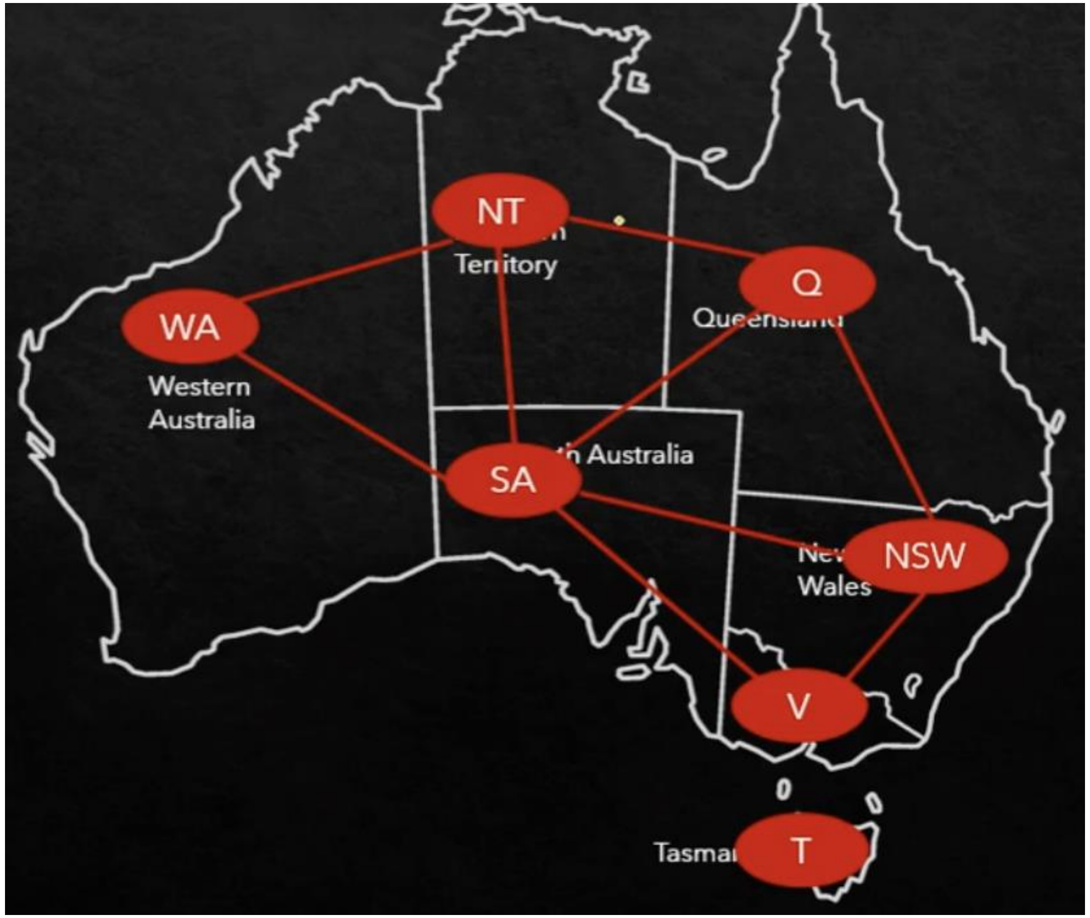
Domain: {1,2,3,4,5,6,7,8,9}

- Constraint
  - 9 ways all different for columns
  - 9 ways all different for rows
  - 9 ways all different for regions
- Constraint
  - $<A11 \neq A12, A11 \neq A13, \ldots, A11 \neq A19>$
  - $<A12 \neq A13, A12 \neq A14, \ldots, A12 \neq A19>$

Example: Map Coloring

- Variables: {WA, NT, SA, Q, NSW, V, T}
- Domain: {blue, red, green}
- Constraint: Adjacent regions have different colour
  - {WA≠NT} or
  - (WA, NT) ∈{(red, green), (red, blue), …}

Graphs as Abstraction Tool



Constraint Graph

Binary CSP:
- constraints involve atmost two variables

Binary Constraint Graph:
- Nodes → Variables
- Arcs → Constraints

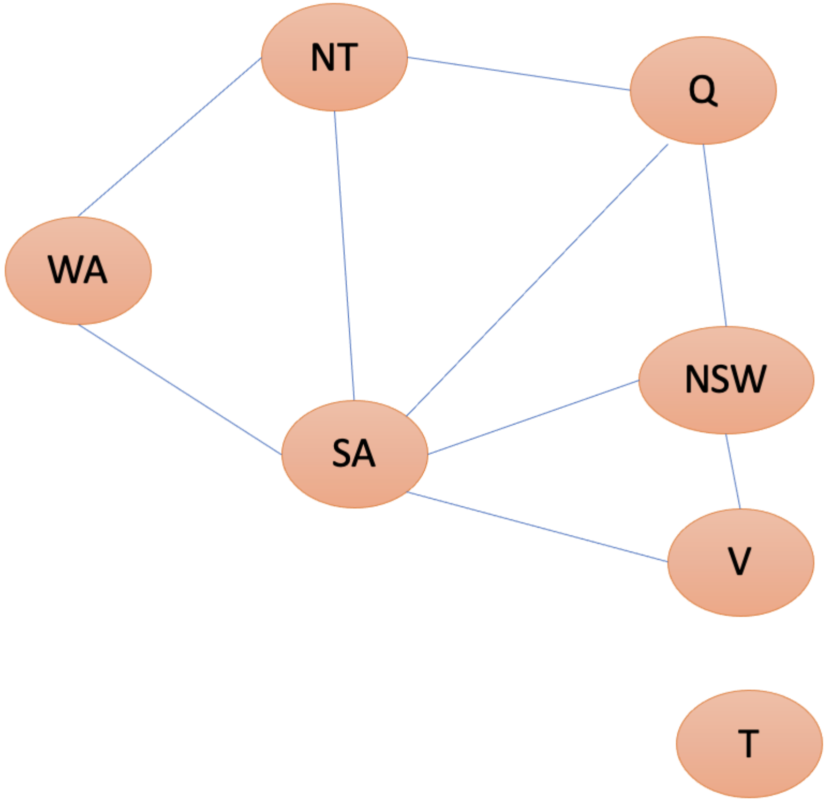
- Claim: CSP algorithms with graph to speed up search
- Generic solvers
  - Abstraction through constraints

CSP Variations: Variables

- Discrete variables
  - **Finite domains**
    - n variables, domain size d ➔O(d^n) complete assignments
    - Example: Boolean CSP, 3-SAT
    - Worst case: Exponential size
  - **Infinite domains**
    - Integer, string
    - Example: Job scheduling [start/end days for job]
    - Constraint Language: start job1 + 10 < start job2
- Continuous variables
  - Start/End times of Hubble Space Telescope observations
  - Linear programming problems

CSP Variations: Constraints

- Unary constraints – single variables
  - SA ≠ green
- Binary constraints
  - SA ≠ WA
- Higher order constraints – 3 or more variables
  - Cryptarithmetic
- Soft Constraints
  - Prof. A prefers to have classes in second half
  - Optimization + CSP
  - Every solution has some values [greater if preferences are kept]



CSP as Search Problem

- Initial State
  - Empty assignment {}
- Successor Function
  - Assign a value to any unassigned variable without conflict w.r.t previously assigned variables
- Goal Test
  - Current assignment complete?
- Path Cost
  - Constant cost for every step

- Incremental Formulation
- Every solution appears at depth n if there are n variables
- Search tree extends upto depth n
- Depth first search algorithms for CSP

Backtracking Search

- Do not proceed down if constraint is violated

- Backtracking search: Uninformed algorithm for CSP

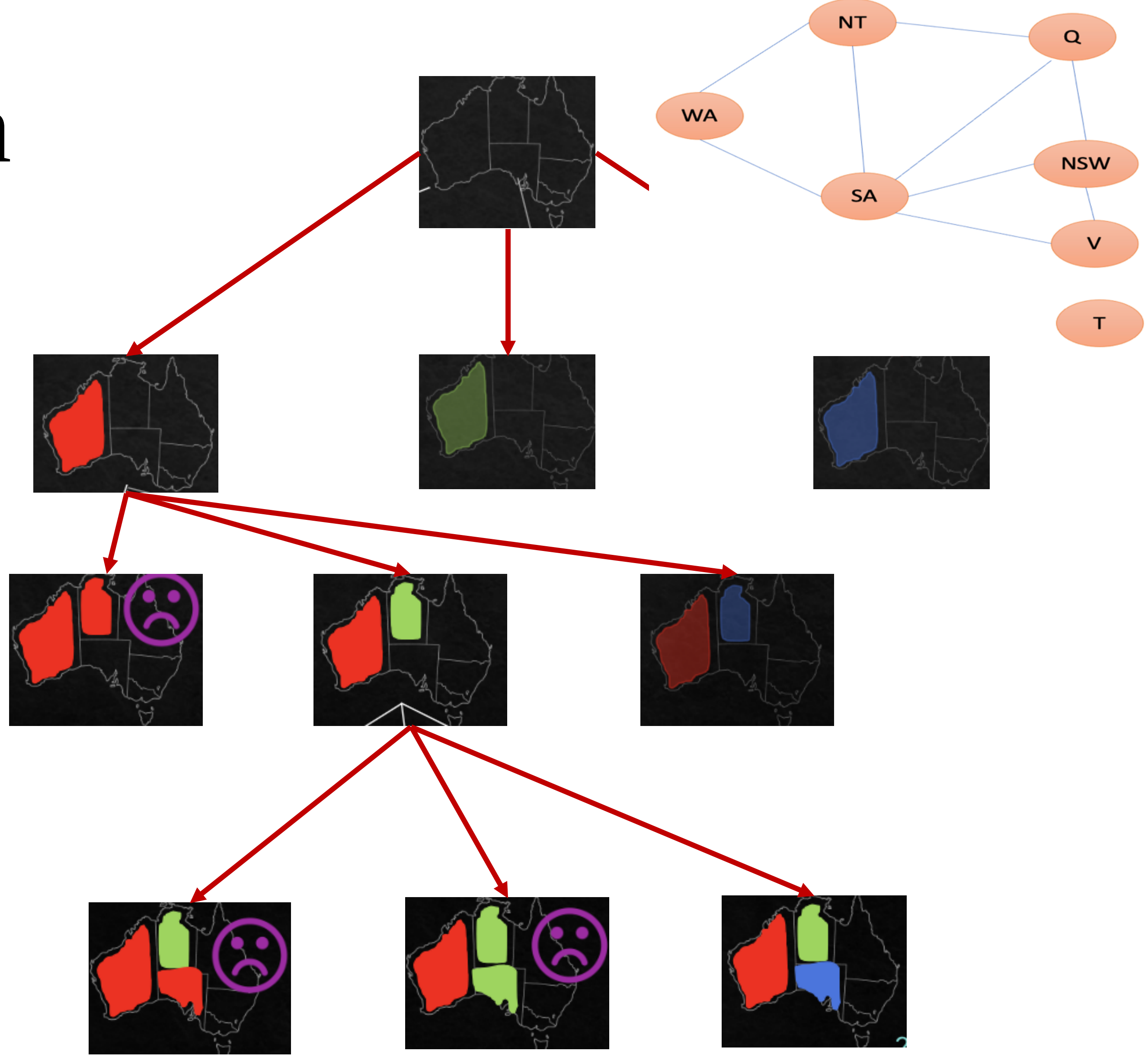
- CSP is commutative
  - Order of actions does not affect the outcome
  - [SA=red then Q=green] same as [Q=green then SA=red]

- CSP can also generate successors by considering assignment for a single variable (Independence)
  - d^n unique values

- Check constraints on the go

Backtracking Search

- Expand
  - Pick a single variable to expand
  - Iterate over domain value
- Process one children
  - One children per value
- Backtrack
  - Conflicting assignment



Making Backtracking more efficient

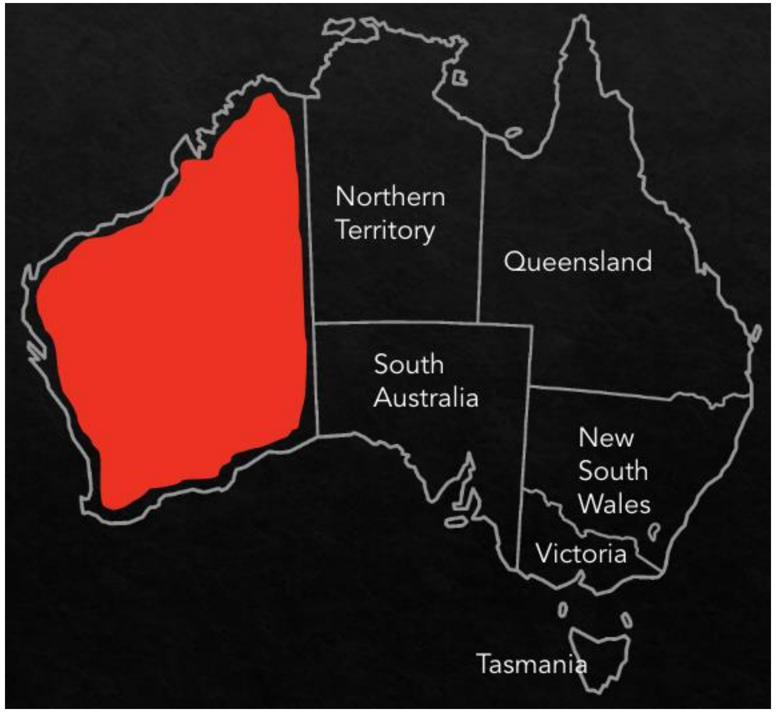
- General uninformed search facilitates huge speed gain

- Ordering
  - Which variable to assigned next?
  - What would be the order of values?

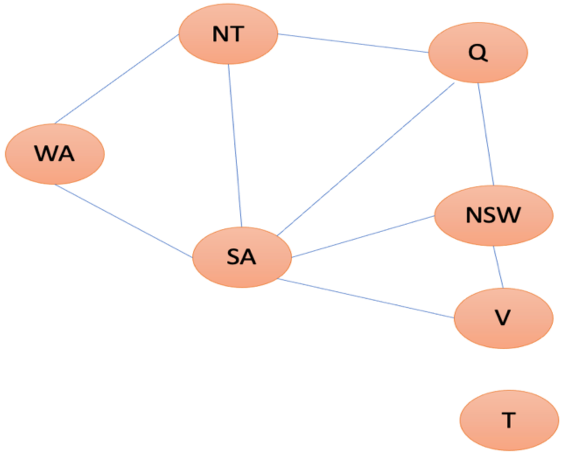
- Filter
  - Can we detect failures early?

- Can we exploit problem structure?

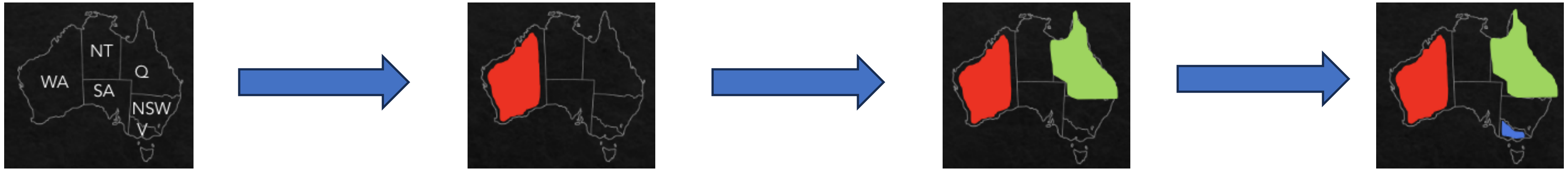
Domain Independent



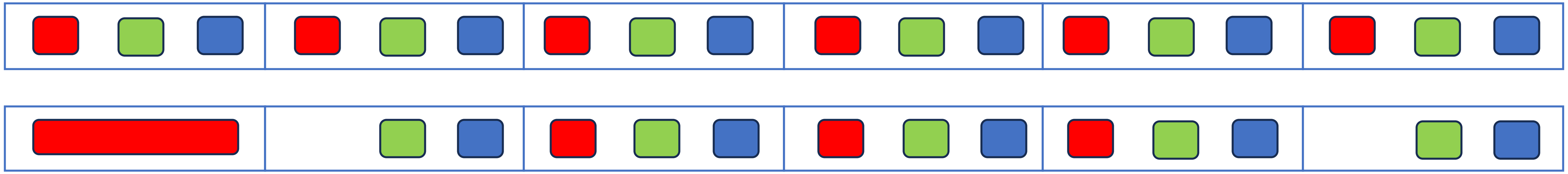
Backtracking Search: Filtering



- Filtering: Take stock of the unassigned variables and filter out the bad options
- Forward checking: Cross off values that violate a constraint when added to existing assignment

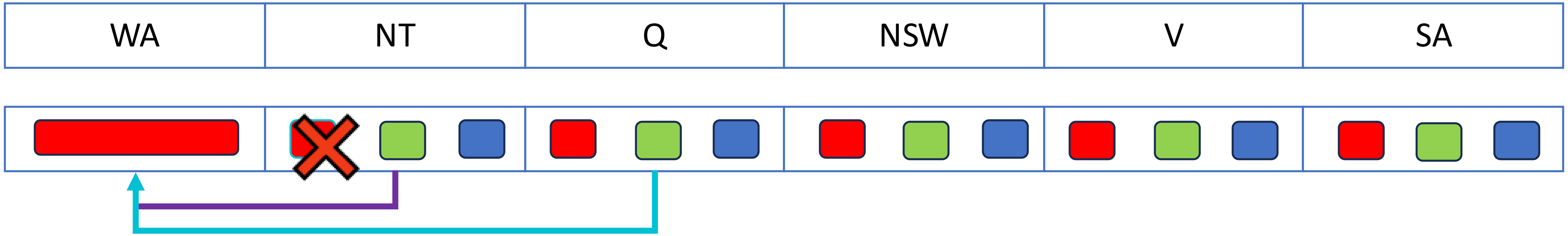


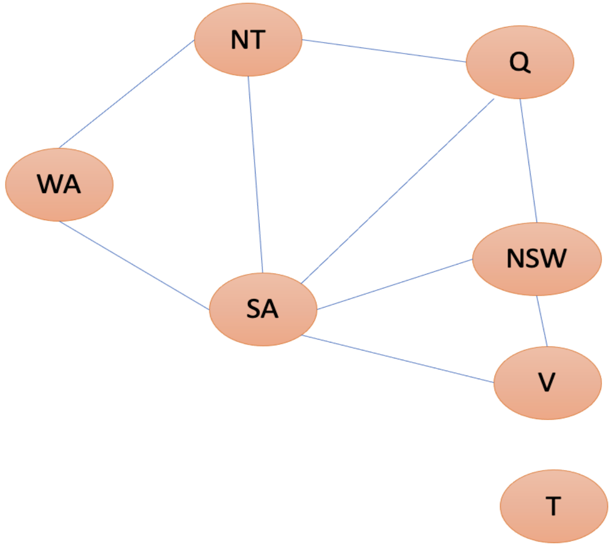
WA	NT	Q	NSW	V	SA



Constraint Propagation: Arc Consistency

- An arc X→Y is consistent iff ∀x in the tail ∃y in the head which could be assigned without violating any constraint





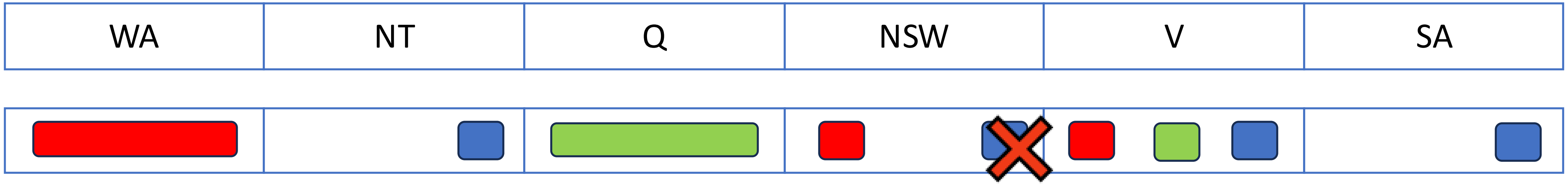
- Is NT →WA consistent?
  - No
  - For NT (RED) there is no options for WA
- How to make it consistent?
  - Delete RED from NT set

- Is Q →WA consistent?
  - Yes

Forward checking: Enforcing consistency of arcs pointing to each new assignment

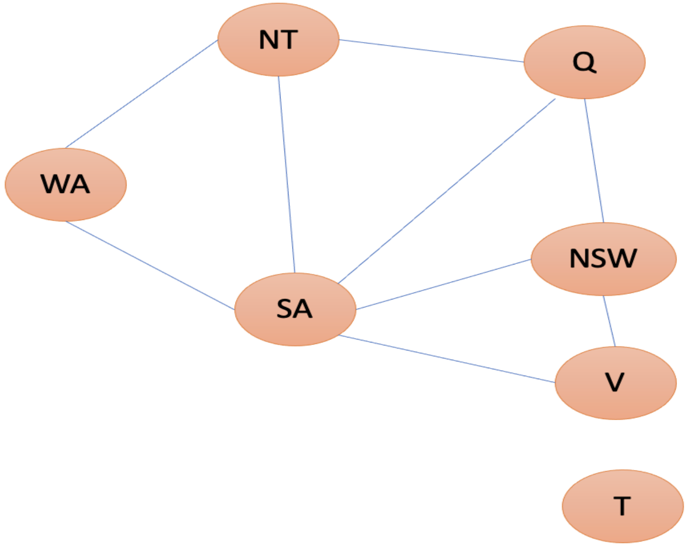
Arc consistency of CSP

- A CSP is consistent iff all the arcs are consistent



If a variable X loses a value, neighbors of X should be rechecked

Arc consistency detects failure before forward checking



- Is V →NSW consistent?
  - (Red, Blue), (Green, Blue), (Blue, Red)
- Is SA →NSW consistent?
  - (Blue, Red)
- Is NSW →SA consistent?
  - (Blue, ---)

Change in one variable affects the other

Constraints get propagated

- How to make NSW →SA consistent?
  - Remove blue from NSW
  - Always delete from the tail
- Is V →NSW consistent?
  - (Red, ---)

Arc consistency of CSP

- function AC-3(csp) returns CSP with reduced (possibly) domains
  - queue ← All the arcs in csp
  - while queue is not empty do
    - (X_i, X_j) ← REMOVE-FIRST(queue)
    - if REMOVE-INCONSISTENT-VALUES(X_i, X_j) then
      - for each X_k in NEIGHBORS[X_i] do
        - add(X_k, X_i) to queue

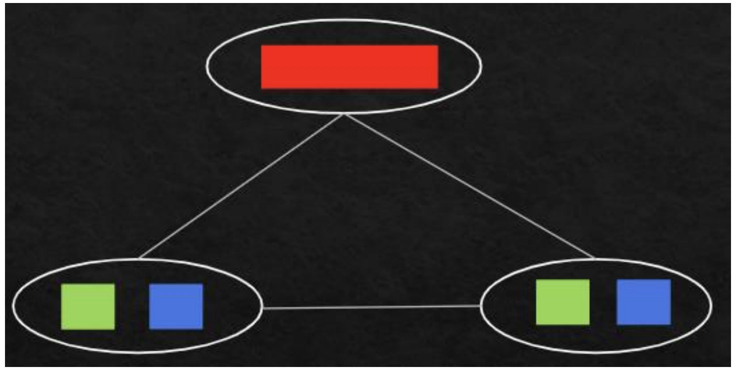
Complexity: O($n^2 d^3$)

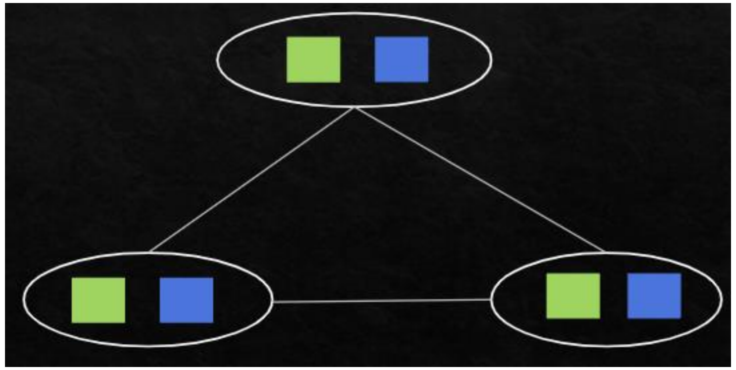
- function REMOVE-INCONSISTENT-VALUES(X_i, X_j) returns true if succeeds
  - removed ← False
  - for each x in DOMAIN[X_i] do
    - If no value in y in DOMAIN[X_j] allows (x,y) to satisfy the constraint $X_i \longrightarrow X_j$ then
      - Delete x from DOMAIN[X_i]
      - removed ← true
  - return removed

Each node has limited number of assignments

Arc Consistency: Limitations

- After enforcing arc consistency
  - Can have one solution left
  - Can have multiple solution left
  - Can have no solution left (unaware)





Thank You