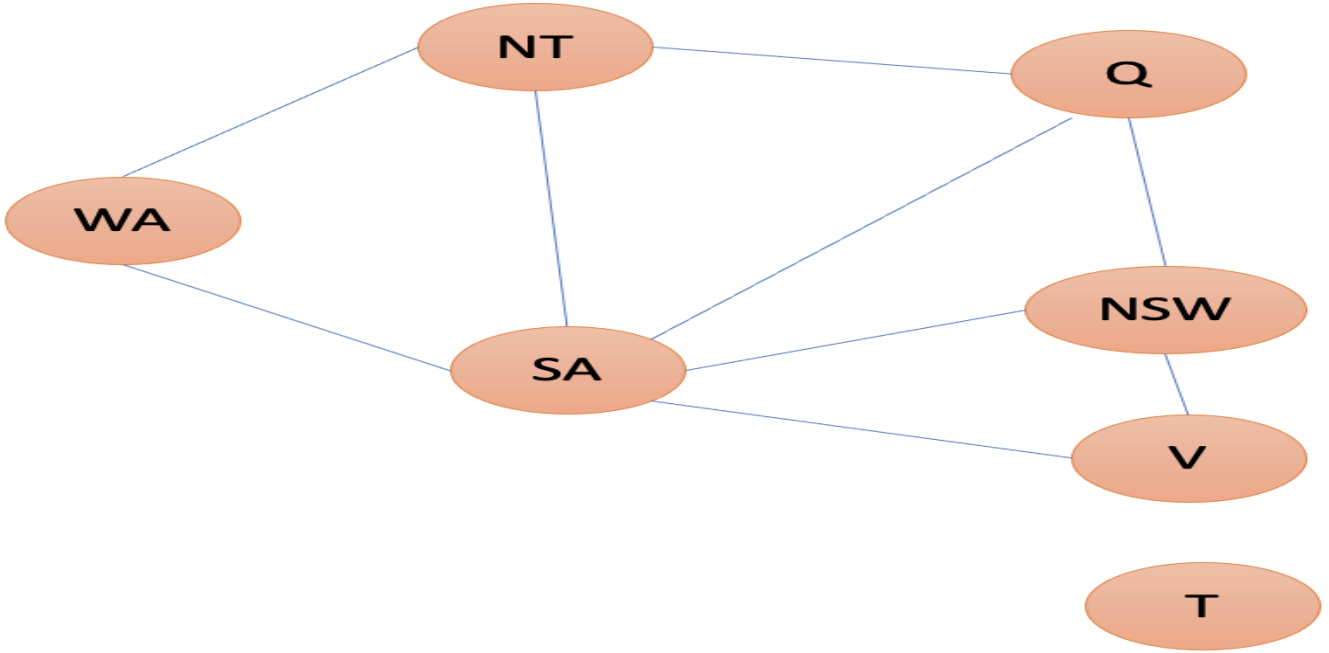
Constraint Satisfaction Problem

27/02/2024

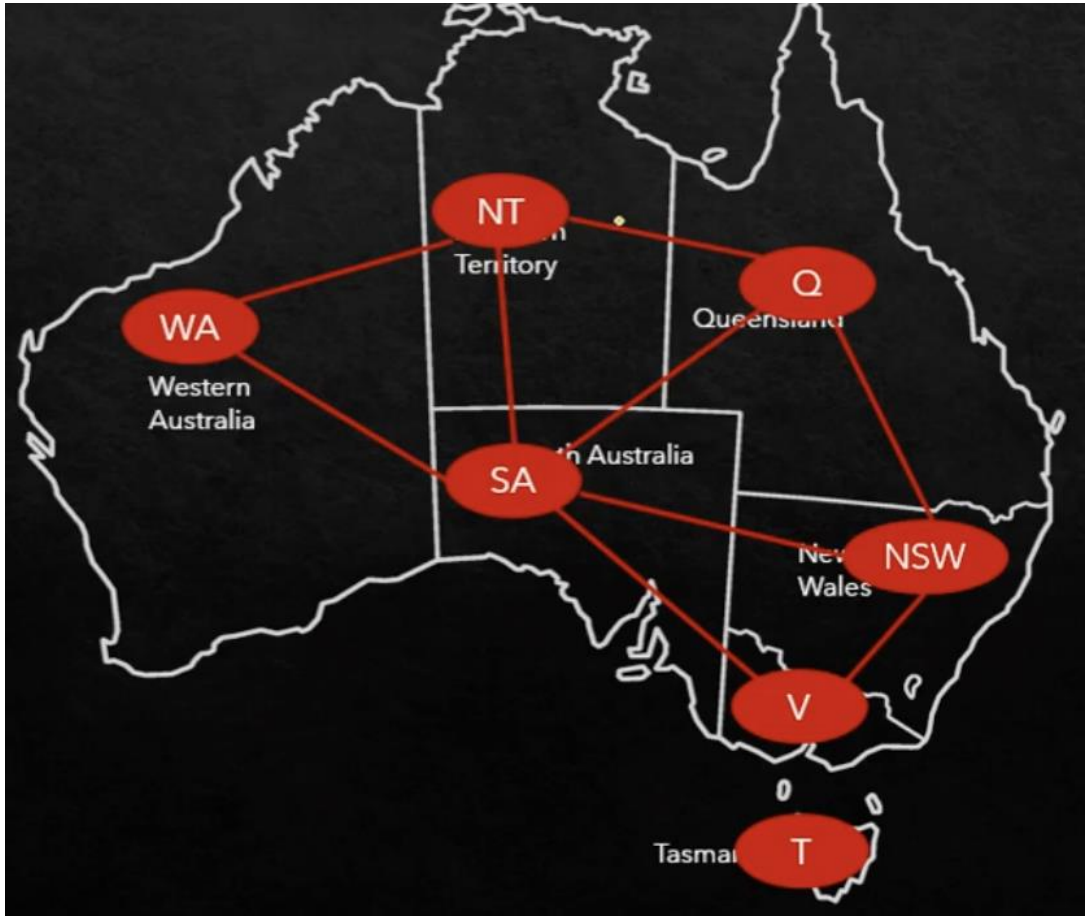
Koustav Rudra

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

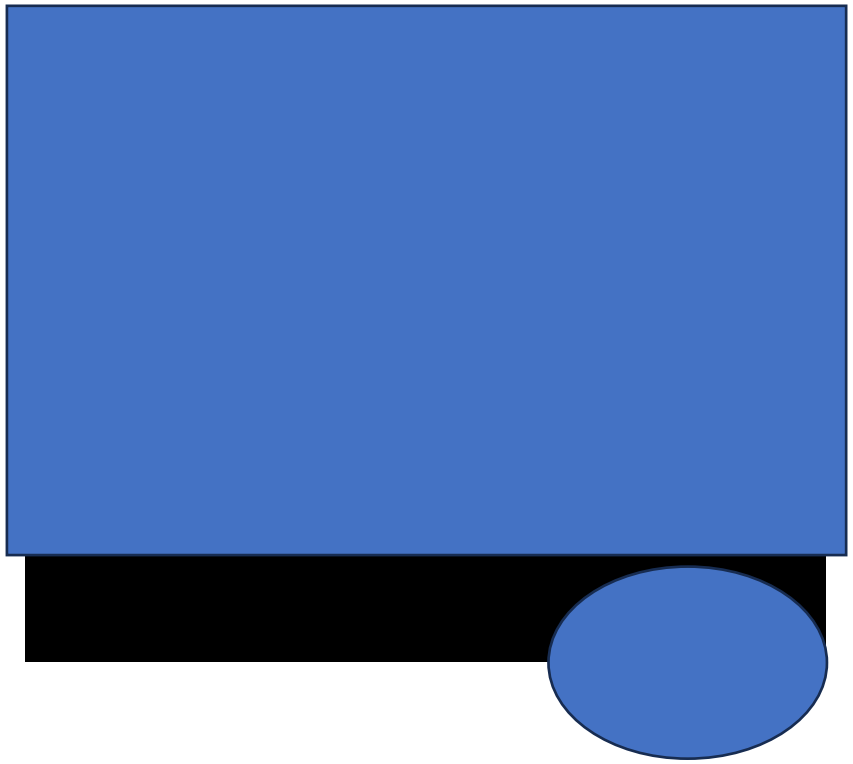
Binary Constraint Graph:

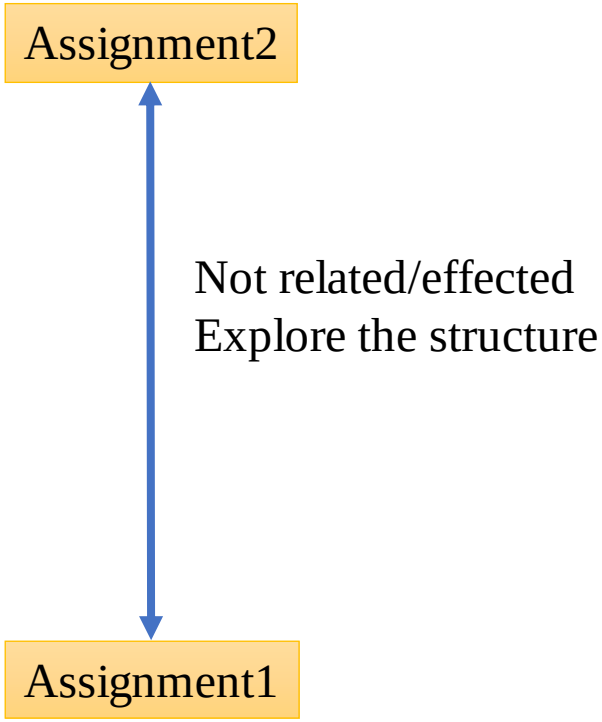
- Nodes → Variables
- Arcs → Constraints

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

What is the big deal?

- CSP solver can prune a large region of the search space



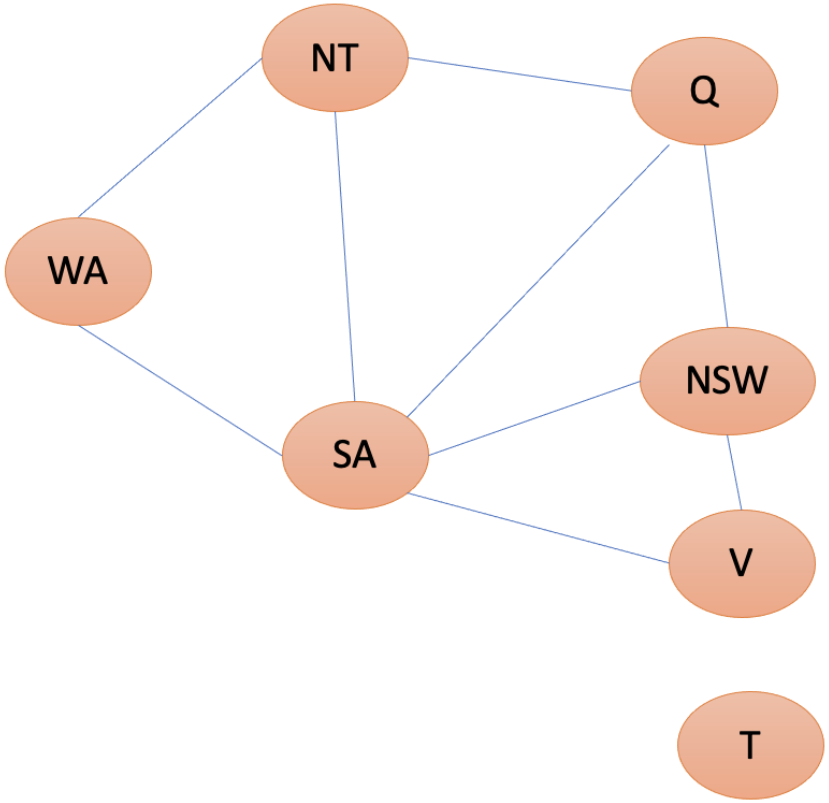


CSP Variations: Variables

- 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]

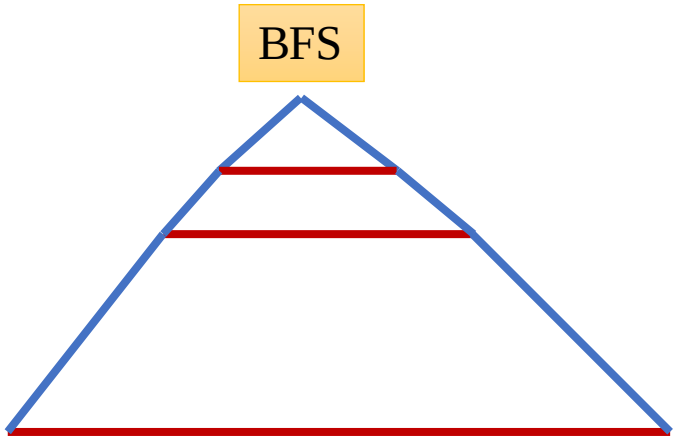


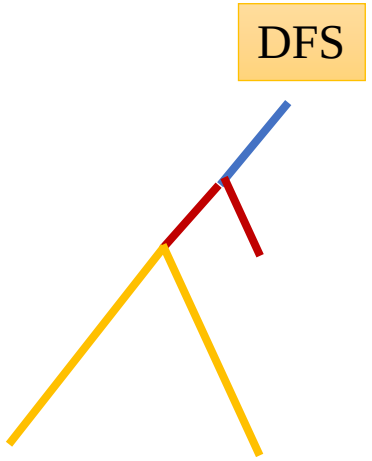
CSP as Search Problem

- 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

CSP: Search Methods





- No optimization
- Guaranteed to work as much as possible

Backtracking Search

- Do not proceed down if constraint is violated

- Backtracking search: Uninformed algorithm for CSP

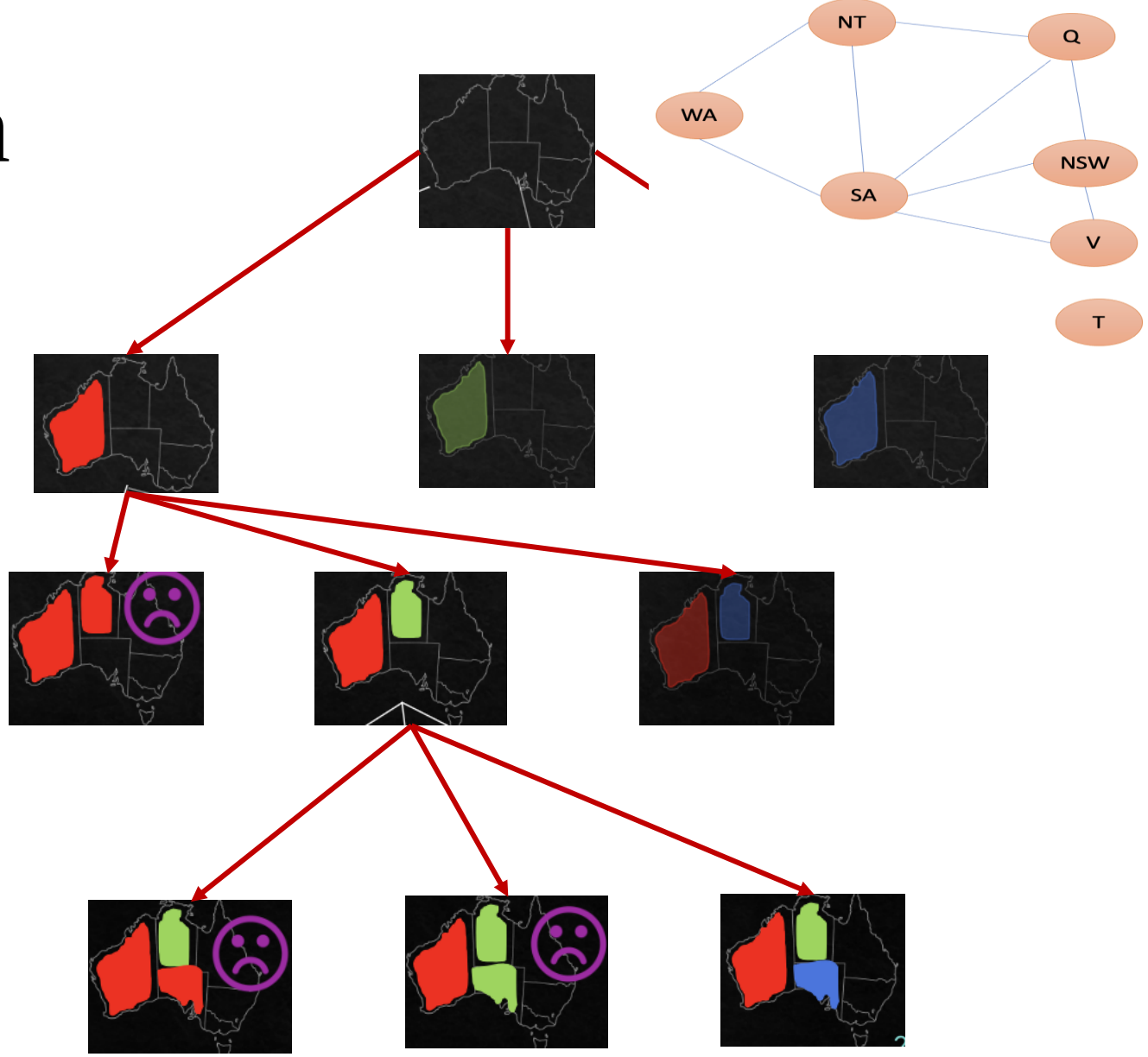
- CSP is commutative
  - Order of actions does not affect the outcome
  - [SA=red then Q=green] same as [Q=green then SA=red]

- 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



Backtracking Search

- function BACKTRACKING-SEARCH(csp) return solution/failure
  - return RECURSIVE-BACKTRACKING({},csp)

- function RECURSIVE-BACKTRACKING(assignment, csp) return sol/fail
  - if assignment is complete then return assignment
  - var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  - for each value in ORDER-DOMAIN-VALUE(var, assignment, csp) do
    - if value is consistent with assignment given CONSTRAINTS[csp] then
      - add {var=value} to assignment
      - result ← RECURSIVE-BACKTRACKING(assignment, csp)
      - if result ≠ failure then return result
      - remove {var=value} from assignment
  - return failure

Backtracking = DFS + Variable ordering + Fail on conflict

https://www.cs.cmu.edu/~15281-s20/demos/csp_backtracking/

Making Backtracking more efficient

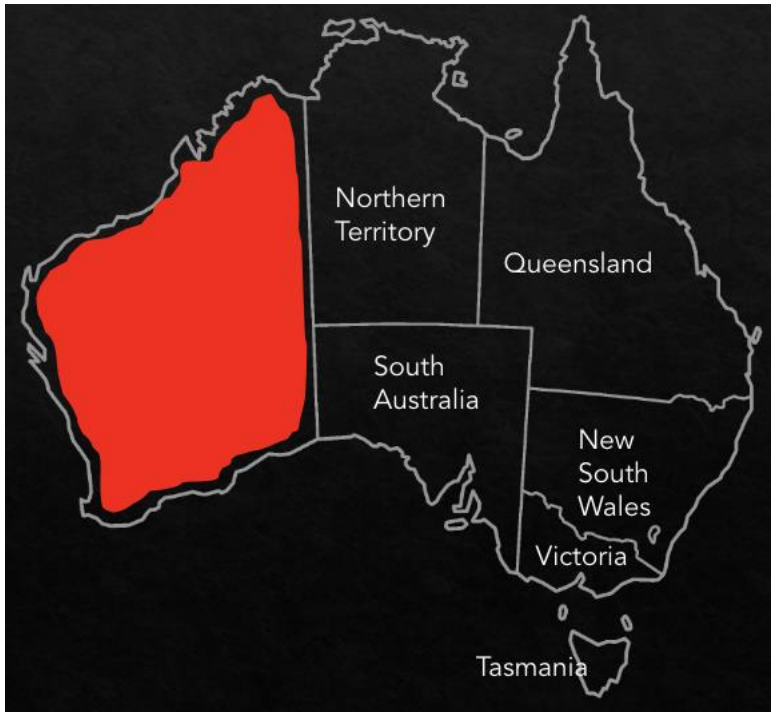
- General uninformed search facilitates huge speed gain

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

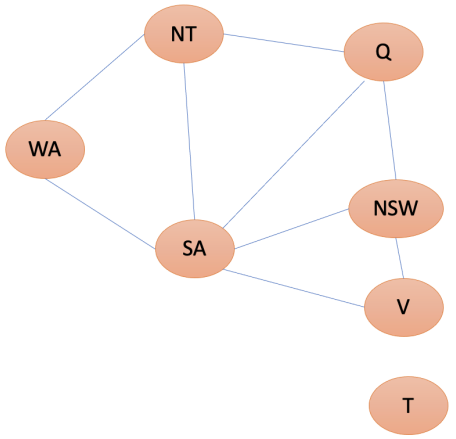
- Filter
  - Can we detect failures early?

- Can we exploit problem structure?

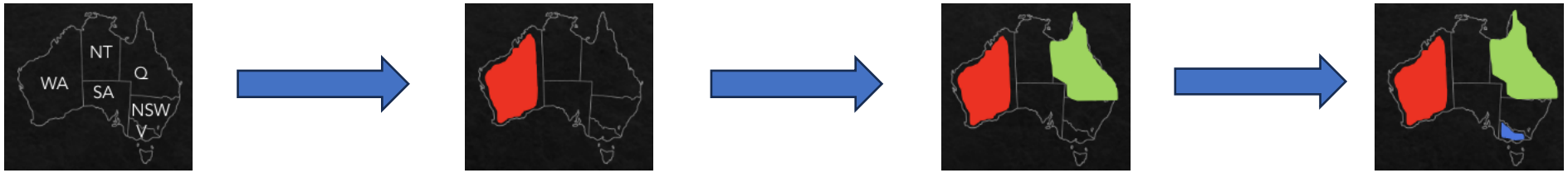
Domain Independent



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 as



https://www.cs.cmu.edu/~15281-s20/demos/csp_backtracking/

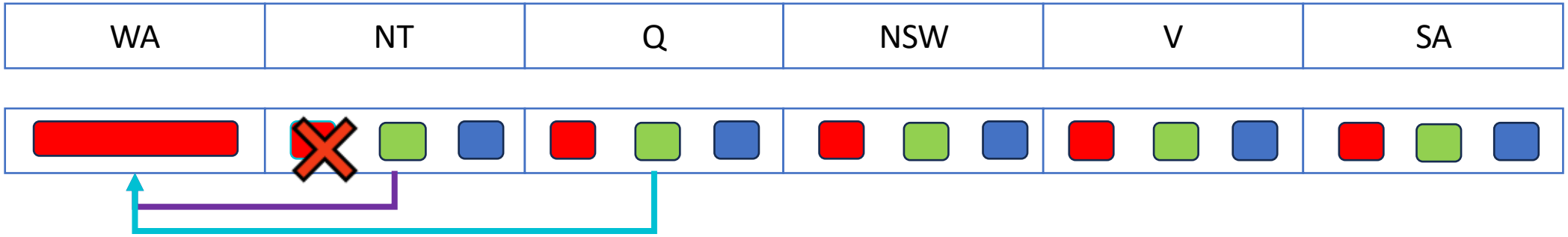
Constraint Propagation

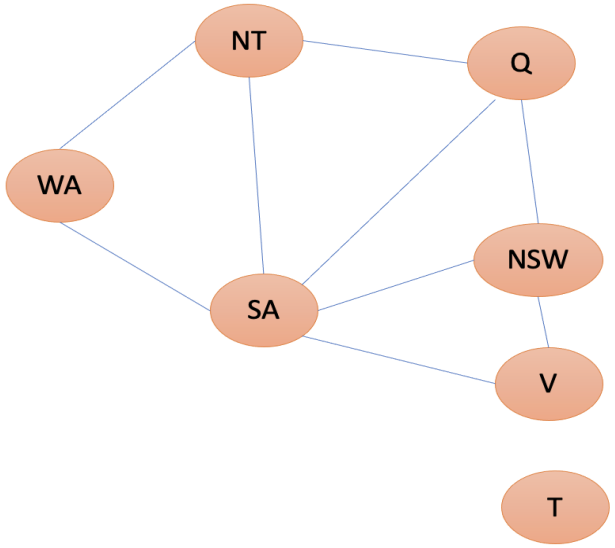
27/02/2024

Koustav Rudra

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





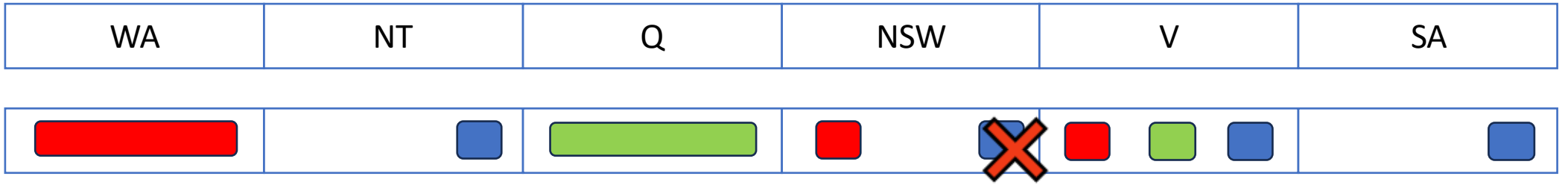
- Is NT →WA consistent?
  - No
  - For NT (RED) there is no options for WA
- How to make it consistent?
  - Delete RED from NT set

- Is Q →WA consistent?
  - Yes

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

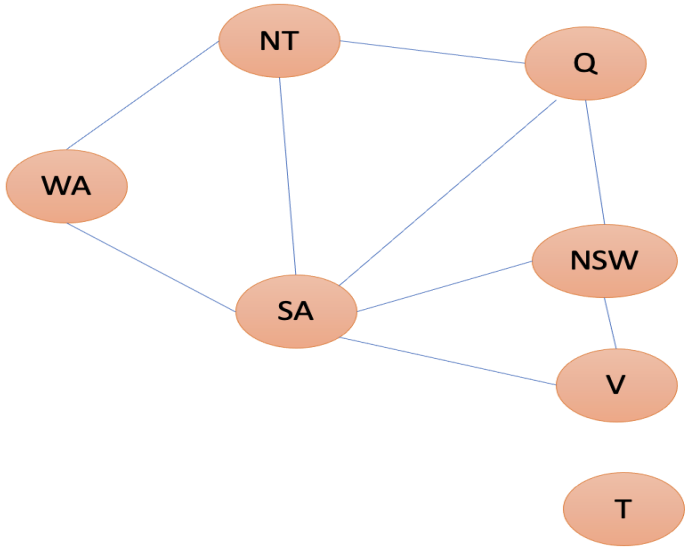
Arc consistency of CSP

- 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

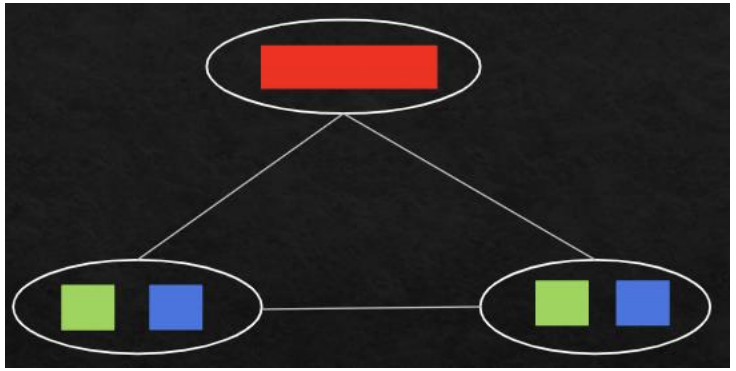
Complexity: O($n^2 d^3$)

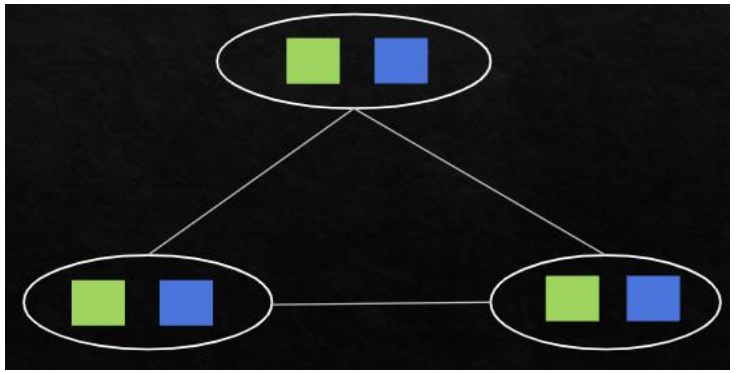
- function REMOVE-INCONSISTENT-VALUES(X_i, X_j) returns true if succeeds
  - removed ← False
  - for each x in DOMAIN[X_i] do
    - If no value in y in DOMAIN[X_j] allows (x,y) to satisfy the constraint $X_i \rightarrow X_j$ then
      - Delete x from DOMAIN[X_i]
      - removed ← true
  - return removed

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)





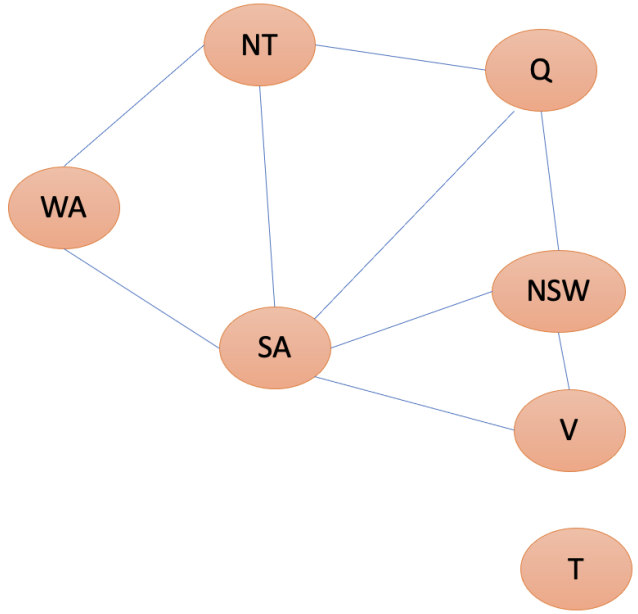
Filtering

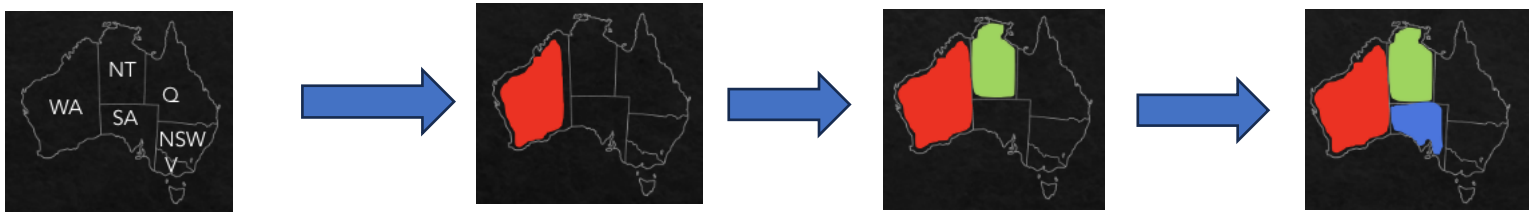
27/02/2024

Koustav Rudra

Ordering: Minimum Remaining Value

- Choose the variable to expand that has fewest legal values left in its doma
  - Most constrained variable
  - X1 = [R, G, B] X2 = [B]





Most Constraint Variable or Fail-Fast ordering

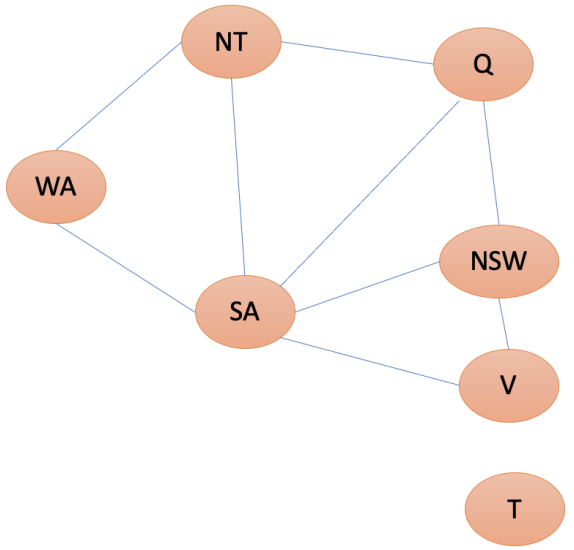
Ordering: Least Constraining Values (LCV)

- Choose the value of a variable that rules out the fewest values in the remaining variables

Q = {Red}
SA = {Blue}

Q = {Blue}
SA = {}

Least constraining variable



CSPs: Recap

- CSP Structure
  - Variables
  - Domains
  - Constraints
    - Implicit (code to compute)
    - Explicit (list of legal tuples)
    - Unary / Binary / n-ary
  - Goals
    - Find any solution
    - Find optimal solution

CSP Solver

- Backtracking give huge gain in speed

- Ordering
  - Which variable should be processed next (MRV)?
  - In what order values of the chosen variable be tried (LCV)?

- Filtering
  - Can we detect eventual failure early?
  - Arc consistency

NP-hard

- Structure
  - Can we exploit the problem structure?

Thank You