RAPIDBI

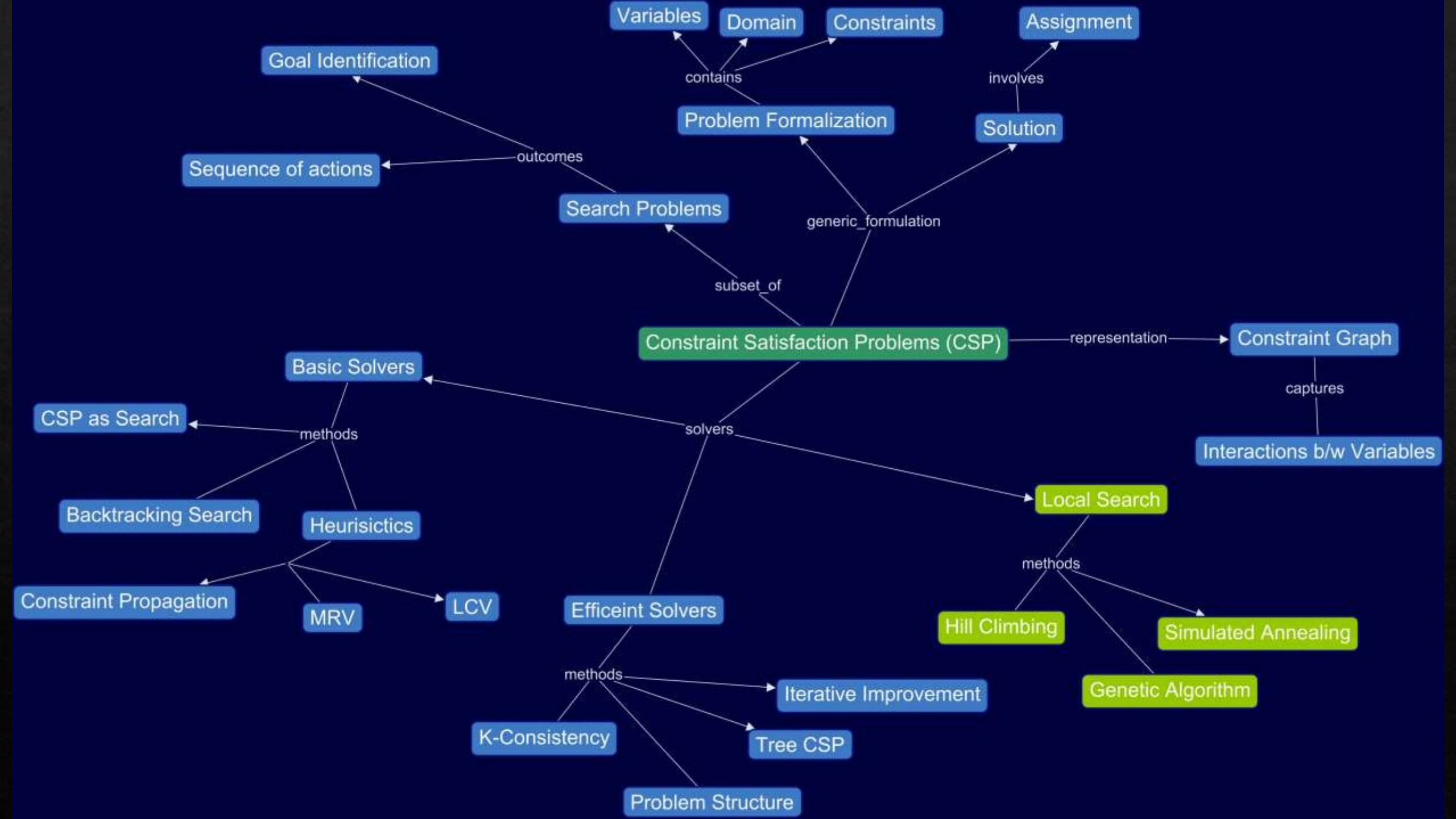
"This is a major project of utmost importance, but it has no budget, no guidelines, no support staff, and it's due in 15 minutes. At last, here's your chance to really impress everyone!"

Constraint Satisfaction Problems

AIFA (AI61005)

2022 Autumn

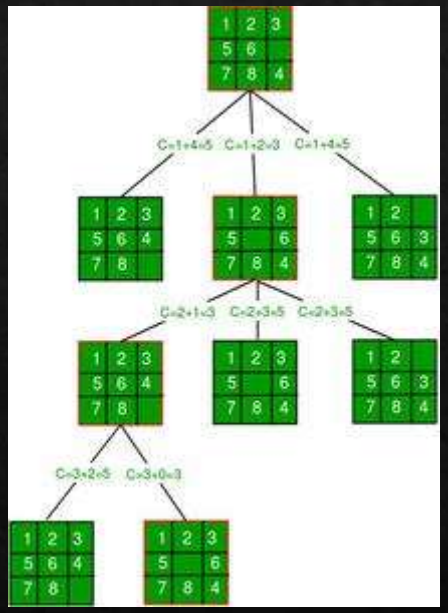
Plaban Kumar Bhowmick


Variables
Domain
Constraints
Assignment
Goal Identification
contains
involves
Problem Formalization
Solution
outcomes
Sequence of actions
Search Problems
generic_formulation
subset_of
Constraint Satisfaction Problems (CSP)
representation
Constraint Graph
Basic Solvers
captures
CSP as Search
methods
solvers
Interactions b/w Variables
Local Search
Backtracking Search
Heurisictics
methods
Constraint Propagation
MRV
LCV
Efficeint Solvers
Hill Climbing
Simulated Annealing
methods
Genetic Algorithm
Iterative Improvement
K-Consistency
Tree CSP
Problem Structure

CSP: Why and What

AI Problem Solvers: Evolution

```
def solve(puzzle puzz):
    ……
    ……
    move(c1, c2)
    check(solution)
    ……
    ……
```



```
def solve(puzzle puzz):
    ……
    ……
    puzz.isGoal()
        return true
    succ = puzz.successor()
    ……
    ……
```

Brute-Force Approach

Search Algorithms

AI Problem Solvers: Evolution

```
def solve(puzzle puzz):
    ……
    ……
    puzz.isGoal()
        return true
    succ = puzz.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` problem agnostic

Design methods and heuristics: Problem Agnostic

Revisiting Search Problems

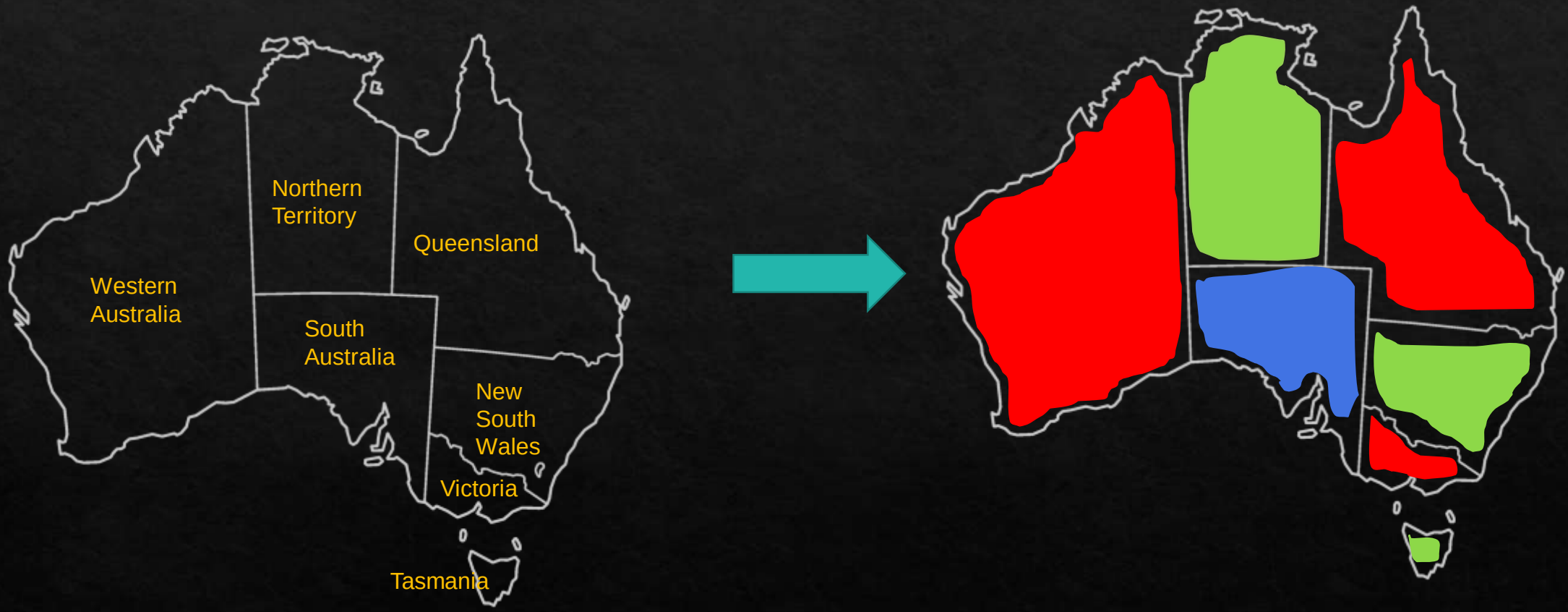
- The world
  - Single agent, deterministic action, fully observable, discrete state
- Planning a sequence of actions
  - Important: Path to goal
  - Paths: varying costs and depths
  - Heuristics to reduce search space
- Identification of goal
  - Goal is important not path
  - All paths are at same depth
  - CSPs are identification problems

Let us Solve

Constraint Satisfaction Problems

- Standard Search Problems
  - State is problem dependent ⇒ Arbitrary data structure
  - Goal test: Function of state
  - Successor: Function of state
- Constraint Satisfaction Problems
  - Subset of search problems
  - State: $\langle X_i, D_i \rangle_N$
  - Goal Test: a set of constraints
    - Legal combinations of values for subset of variables

Constraint Satisfaction Problems



Map Coloring Problem

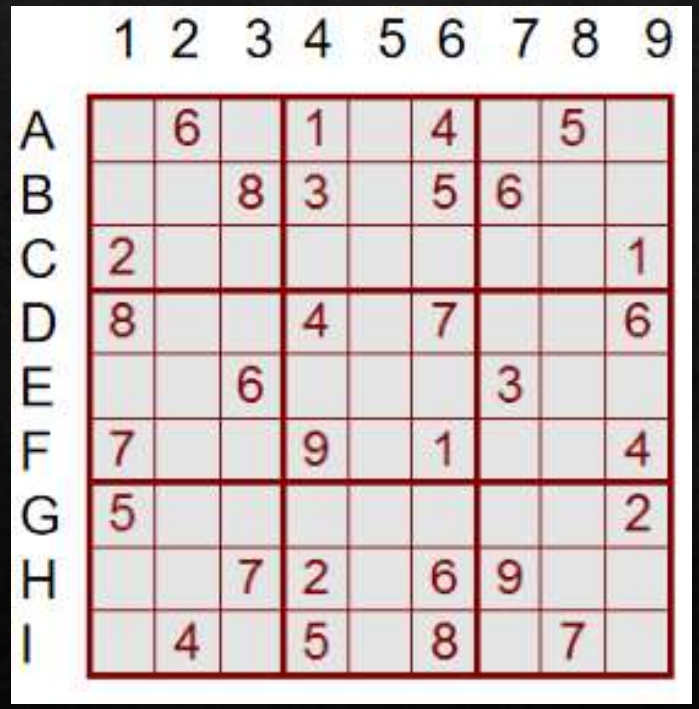
CSPs: Formulation

- CSPs Problem: $\langle X, D, C \rangle$
  - State: $X \Rightarrow$ set of variables, Domain$(X_i) = D_i$
  - Goal Test: set of constraints C
    - $C_i = f(X')$ where $X' \subseteq X$
  - Constraint Definition
    - A pair $\langle scope, rel \rangle$
  - Example: X_1 and X_2 have domain $\{A, B\}$
    - Constraints: $\langle (X_1, X_2), [(A, B), (B, A)] \rangle$, $\langle (X_1, X_2), X_1 \neq X_2 \rangle$

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

CSP 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:

Domain:

Constraint:

CSP Example: Map Coloring

Variables:

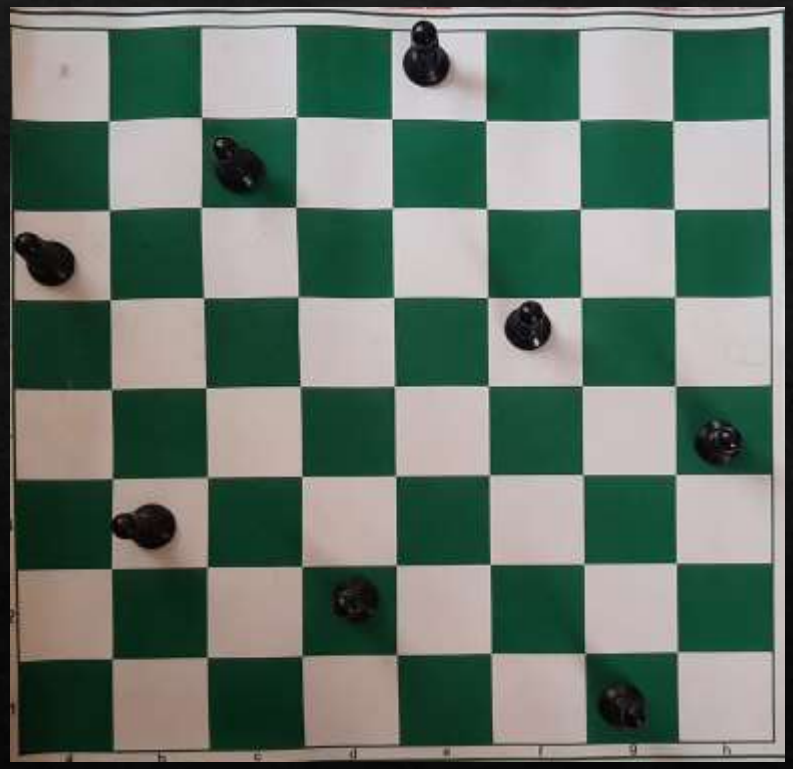
Domain:

Constraint:

Solution:

CSP Example: N-Queens

Formulation - 1



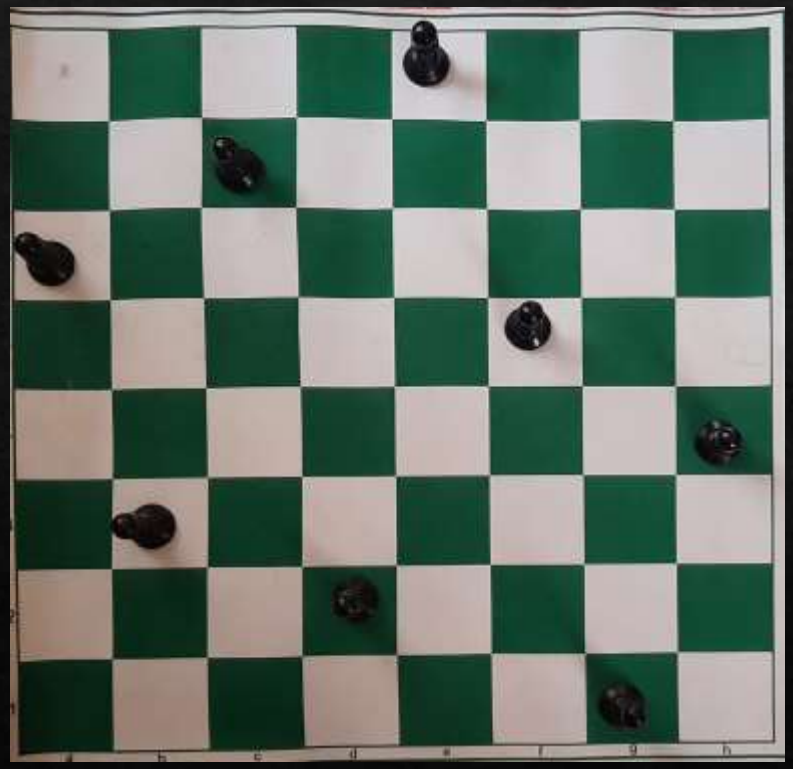
Variables:

Domain:

Constraint:

CSP Example: N-Queens

Formulation - 2

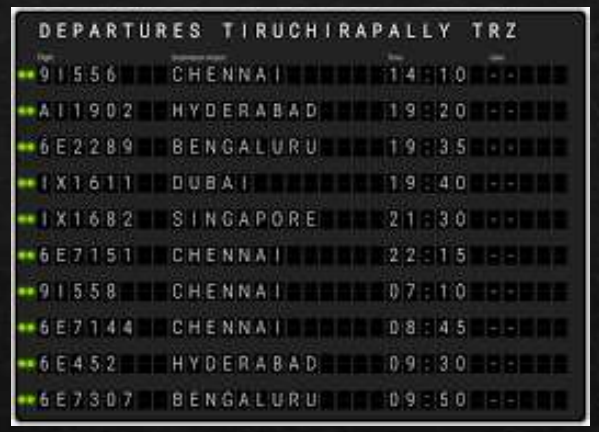


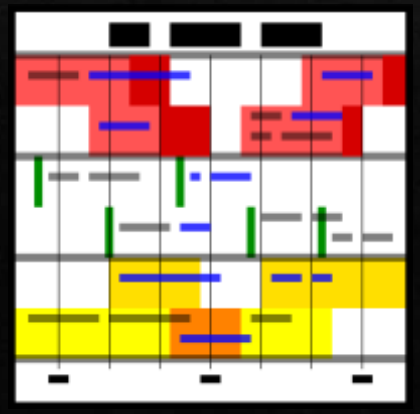
Variables:

Domain:

Constraint:

Example Applications



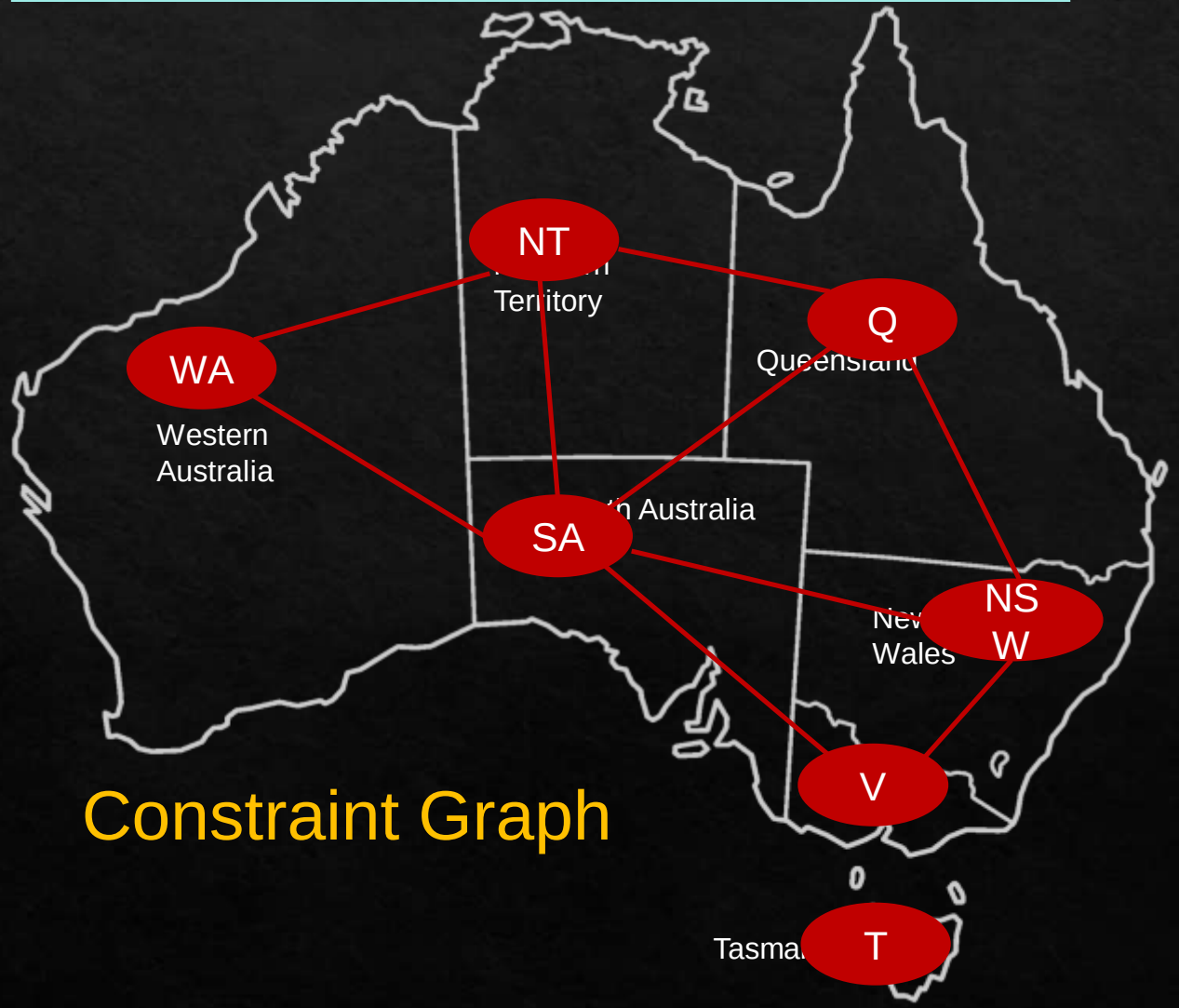


Map Coloring to Graph Coloring

- More general problem than graph coloring
- Planar Graph = Graph drawn on 2D plane without edges crossing
- "*Every planar graph can be colored with 4 or less*" –

  Guthrie (1852)
- Was proven in 1977 – Appel and Haken

Graphs as Abstraction Tool



Constraint Graph

Binary CSP:

Binary Constraint Graph:

Claim: CSP algorithms with graph to speed up search

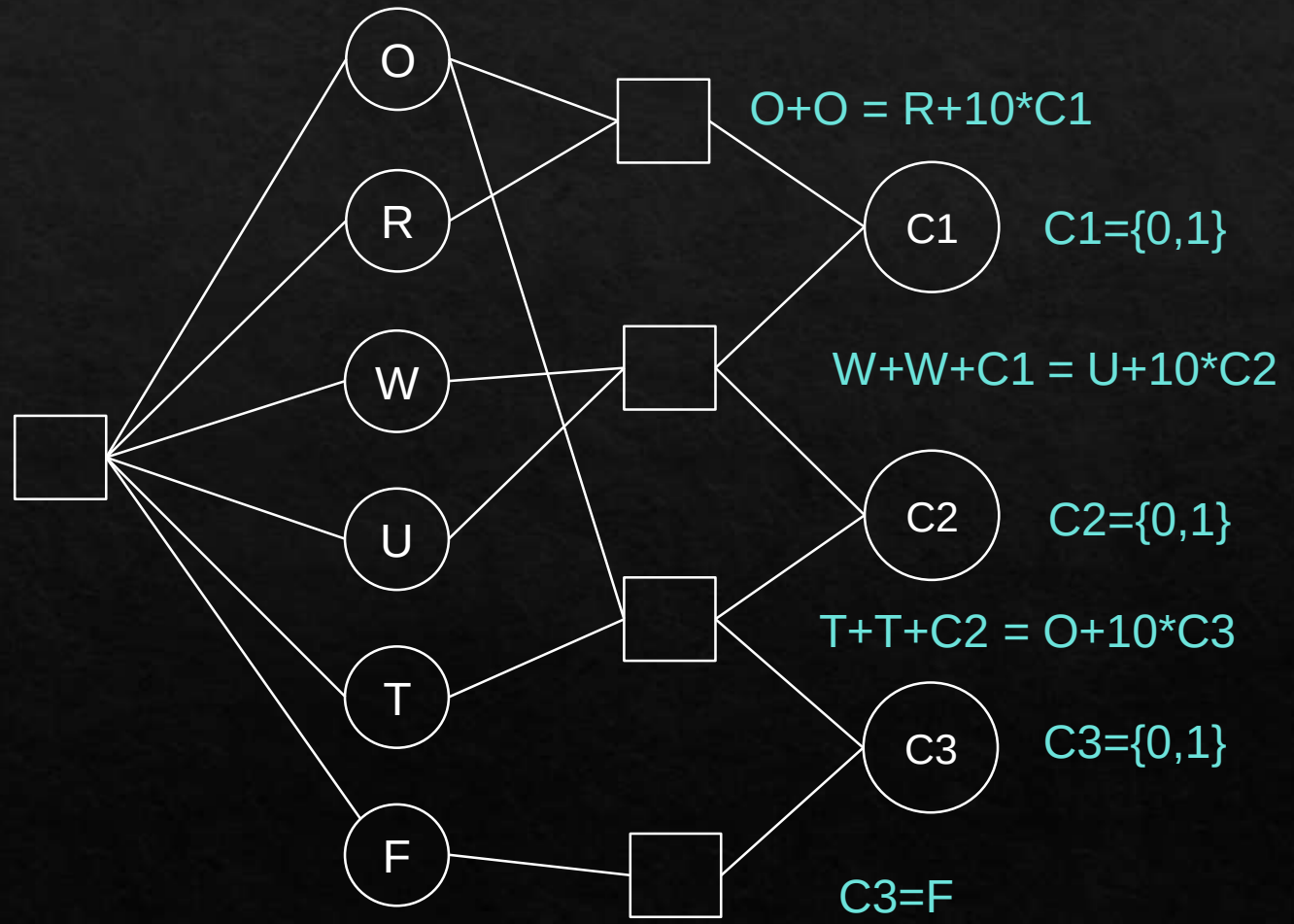
CSP Example: Cryptarithmetic

```
  T W O
+ T W O
-------
F O U R
```

Variables =

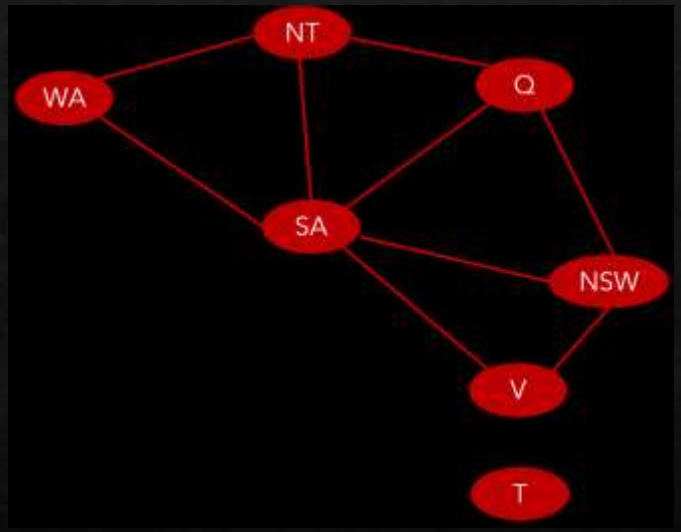
Domain =

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 $d \Rightarrow O(d^n)$ complete assignments
    - Example: Binary CSP, 3-SAT
    - Worst Case: Exponential size
  - Infinite domains:
    - Integer, string
    - Example: Job scheduling [start/end days for job]
    - Constraint language: StartJob1 +10 ≤ StartJob2
    - Linear Constraint: Solvable
    - Nonlinear: No general algorithm
- Continuous Variables
  - Start/End times of Hubble Space Telescope observations
  - Linear programming problems

CSP Variations: Constraints

- Unary constraints – Single variables
  - SA ≠ green
- Binary constraints – Pair of variables
  - SA ≠ WA
- Higher order constraints – 3 or more variables
  - Cryptarithmatic - W+W+C1 = U+10*C2
- Preference – Soft constraints
  - Prof. A prefers to have classes in the 2nd half
  - Optimization + CSP

CSP: Basic Solvers

CSP as Search Problem

- Initial State
  - Empty assignment {}
- Successor Function
  - Assign a value to any unassigned variable without conflict wrt 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

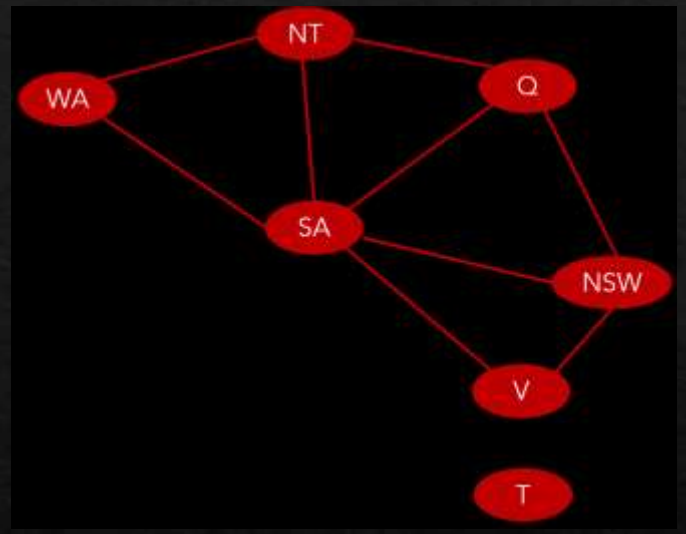
Search tree extends upto n depth

Depth first search algorithms for CSP

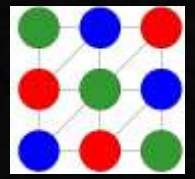
CSP: Search Methods

BFS Strategy

DFS Strategy



Analysis

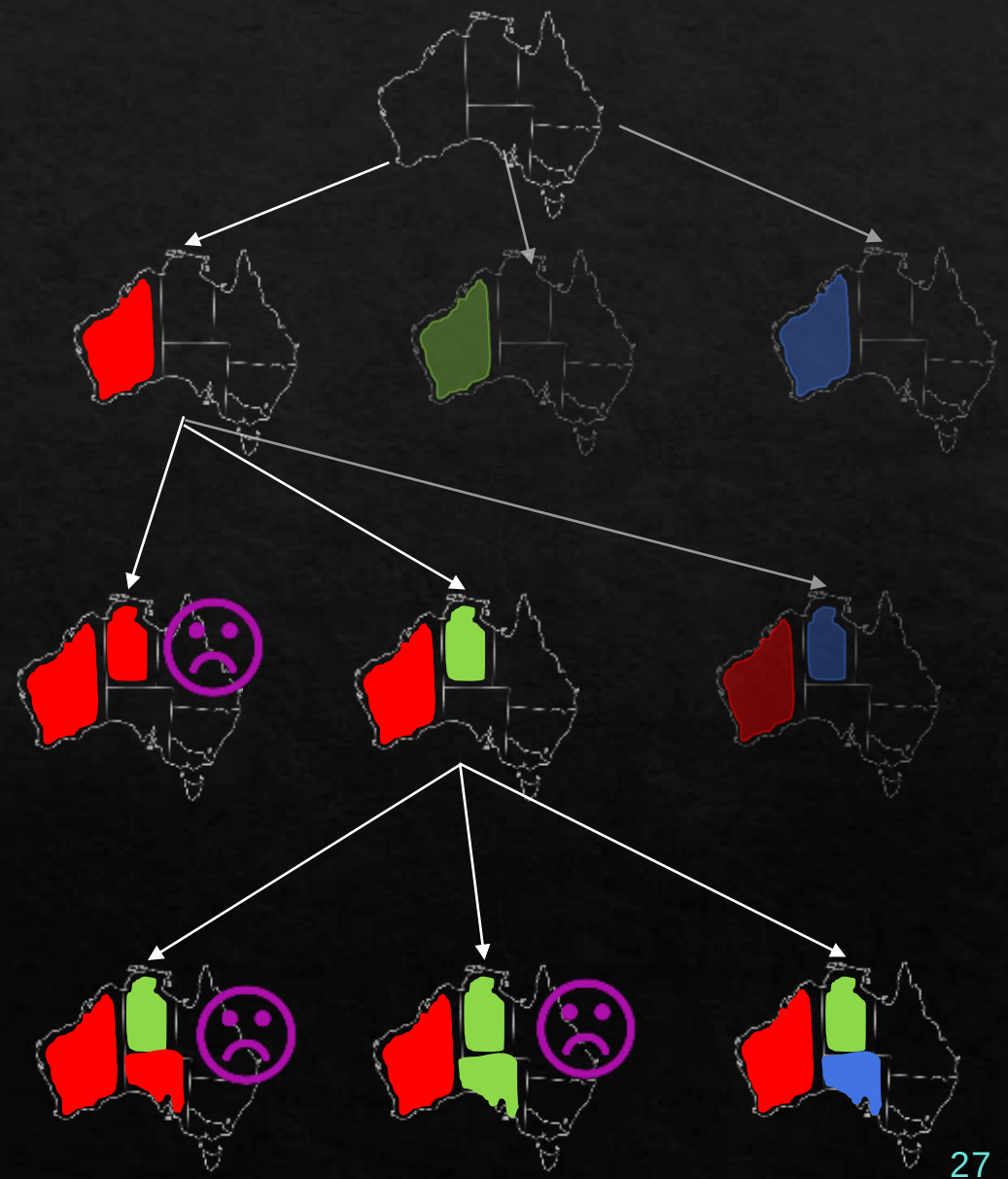


Backtracking Search

- Do not proceed down if constraint is violated
- Backtracking search: Uninformed algorithm for CSP
- CSP is commutative
  - Oder of actions does not affect the outcome
  - [SA=red then Q=green] same as [Q=green then SA=red]
- CSP algo can generate successors by considering assignment for a single variable (Independence)
  - d^n unique leaves
- Check constraints on the go

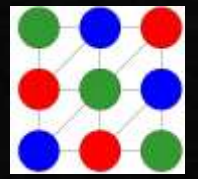
Backtracking Search

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

function RECURSIVE-BACKTRACKING (assignment, csp) returns 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

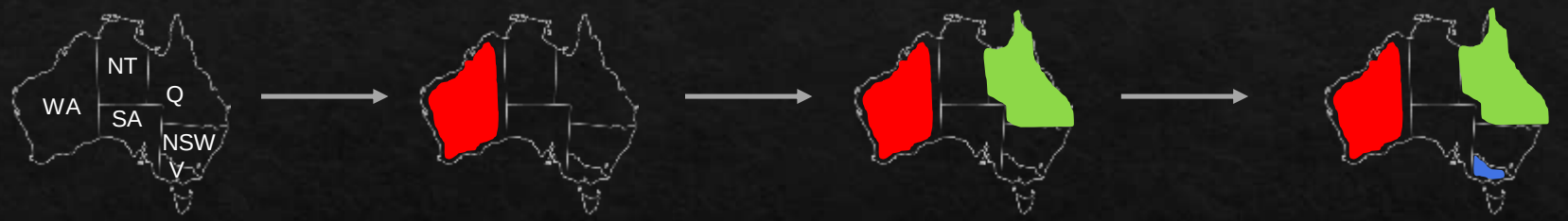
Making Backtracking More Efficient

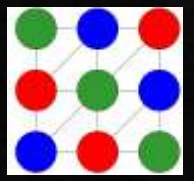
- General uninformed search facilitates huge speed gain
- Ordering
  - Which variable to assigned next?
  - What would be the order of values?
- Filter
  - Can we detect failures early?
- Can we exploit problem structure?

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





Filtering: Constraint Propagation

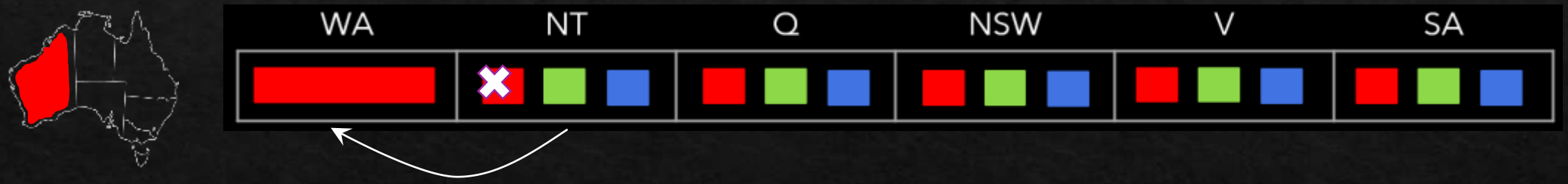
NT and SA both must be assigned blue ⇒ Constraint violation

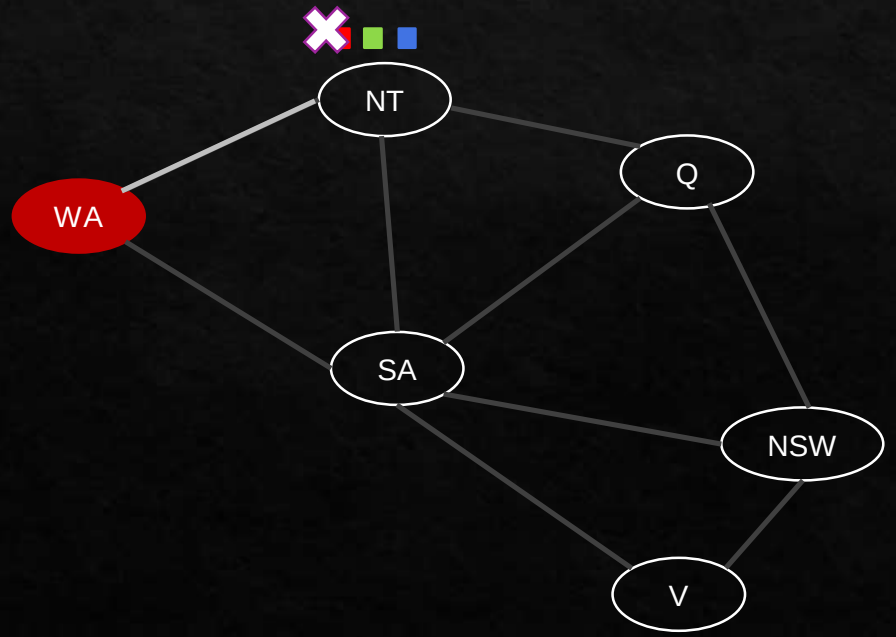
Can we detect it earlier?

Constraint Propagation: Checking interaction between unassigned variables

Constraint Propagation: Arc Consistency

An arc $X \rightarrow Y$ is consistent iff $\forall x$ in the tail $\exists y$ in the head which could be assigned without violating any constraint

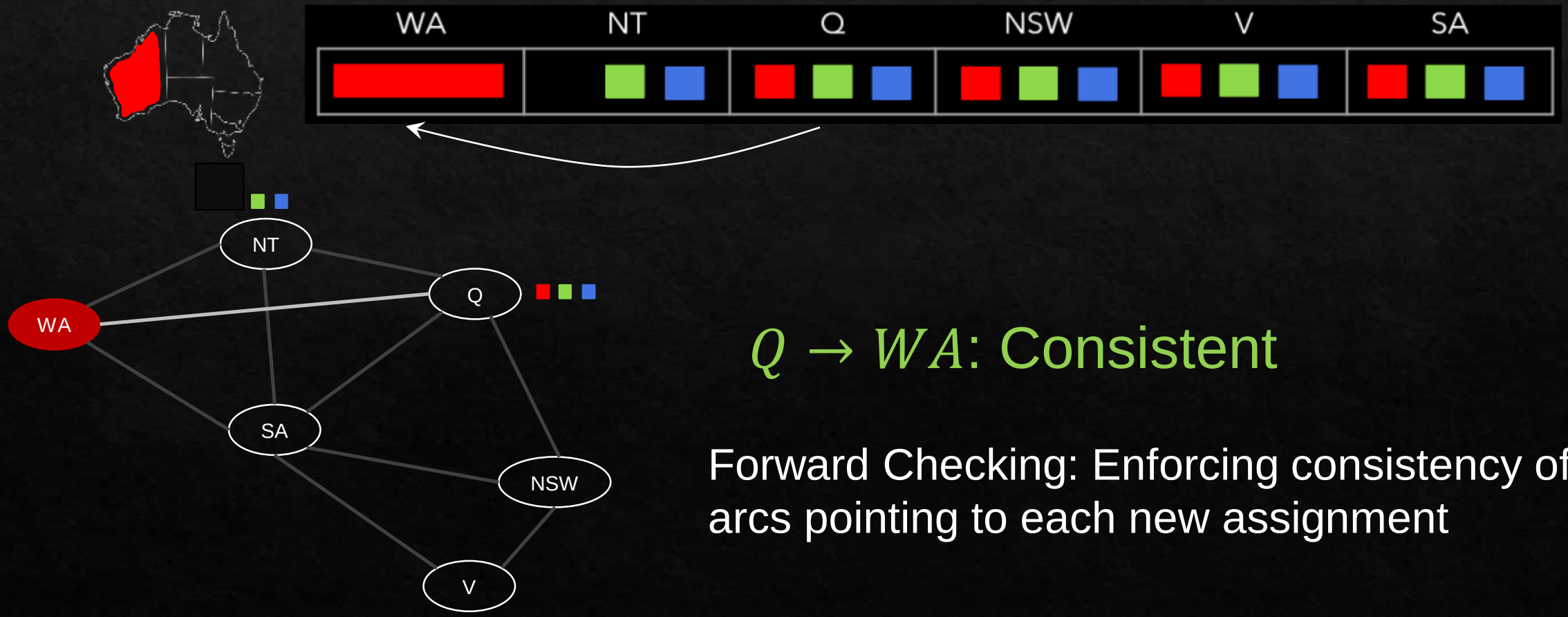




$NT \rightarrow WA$: Not Consistent

Constraint Propagation: Arc Consistency

An arc $X \rightarrow Y$ is consistent iff $\forall x$ in the tail $\exists y$ in the head which could be assigned without violating any constraint



$Q \rightarrow WA$: Consistent

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

Arc Consistency of CSP

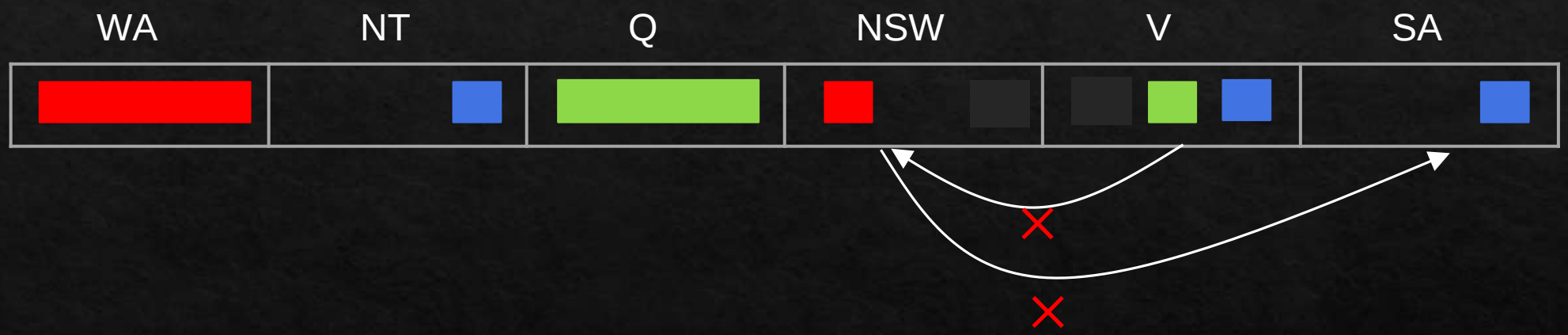
A CSP is consistent iff all the arcs are consistent

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

Arc consistency detects failure before forward checking

Arc Consistency of CSP

A CSP is consistent iff all the arcs are consistent



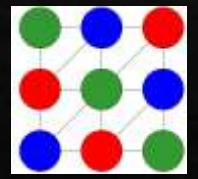
If a variable X loses a value, neighbors of X should be rechecked

Arc consistency detects failure before forward checking

Enforcing Arc Consistency in CSP

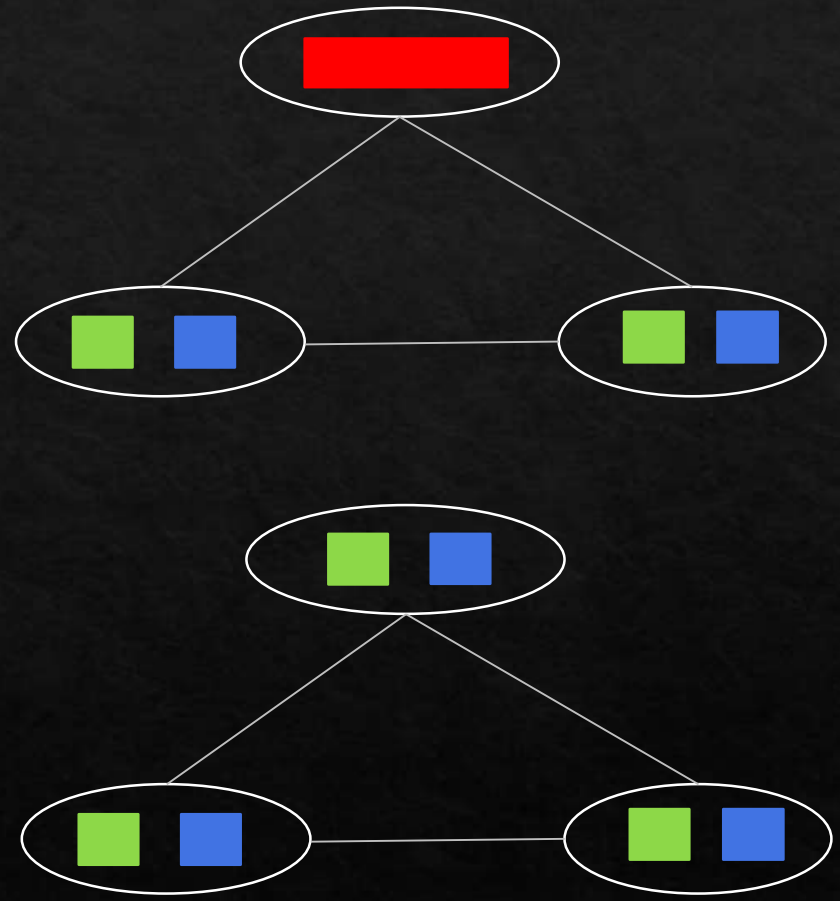
```
function AC-3 (csp) returns CSP with reduced (possibly) domains
    queue ← All the arcs in csp
    while queue is not empty do
        (X_i, X_j) ← REMOVE-FIRST(queue)
        if REMOVE-INCONSISTENT-VALUES (X_i, X_j) then
            for each X_k in NEIGHBORS[X_i] do
                add (X_k, X_i)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

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 y in DOMAIN [X_j] allows (x, y) to satisfy the constraint X_i → X_j then
            delete x from DOMAIN [X_i]
            removed ← true
    return removed
```

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



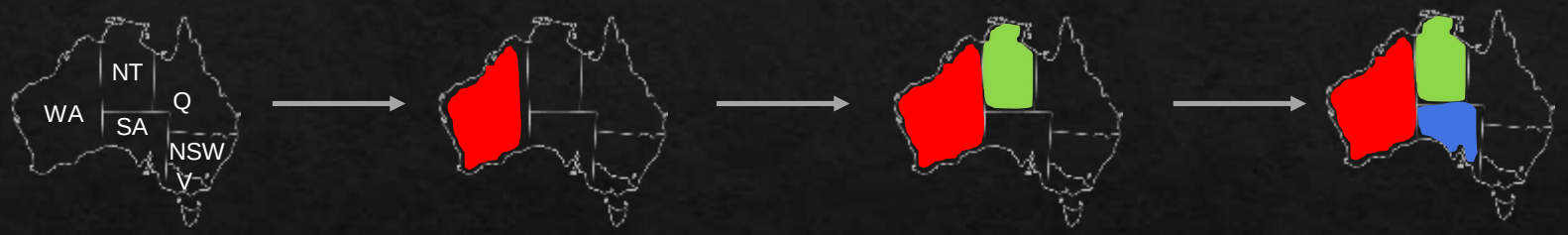
Arc Consistency: Limitation

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



Ordering: Minimum Remaining Values (MRV)

Choose the variable that has fewest legal values left in its domain

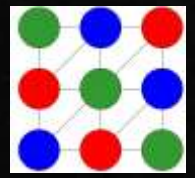


Most Constraint Variable or Fail-Fast ordering

Ordering: Least Constraining Values (LCV)

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

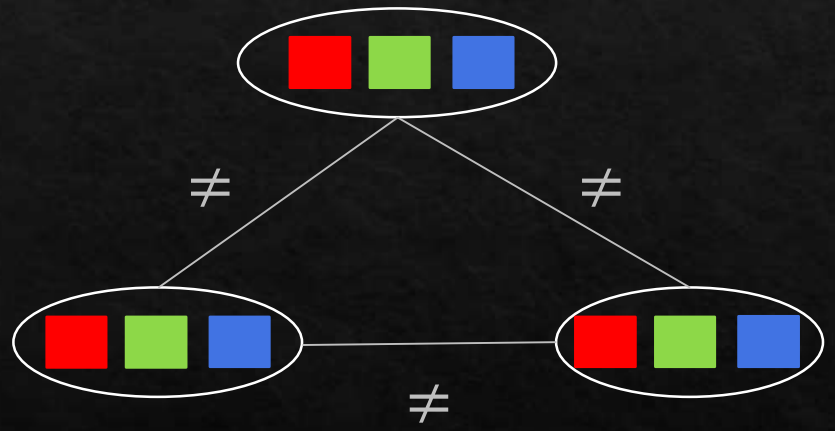
While choosing value for variable Q
“red” is the least constraining variable



Combined ordering can solve upto 1000 queen problem

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



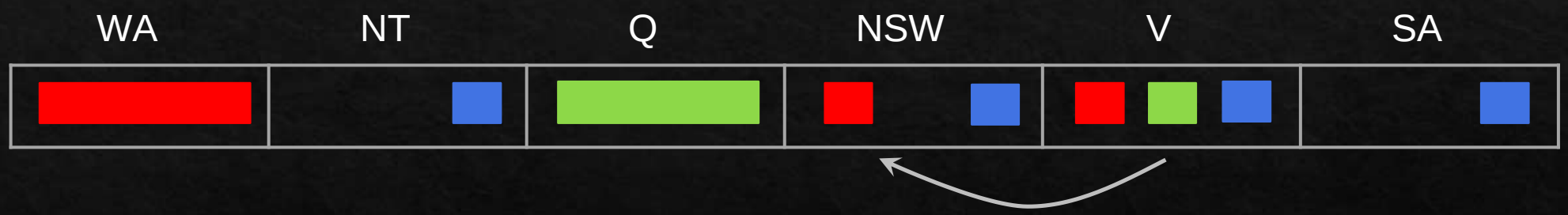
CSP Solver

- 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
- Structure
  - Can we exploit the problem structure?
- Pinch of Salt
  - In general CSPs are NP-hard

Arc Consistency of CSP

An arc $X \rightarrow Y$ is consistent iff $\forall x$ in the tail $\exists y$ in the head which could be assigned without violating any constraint

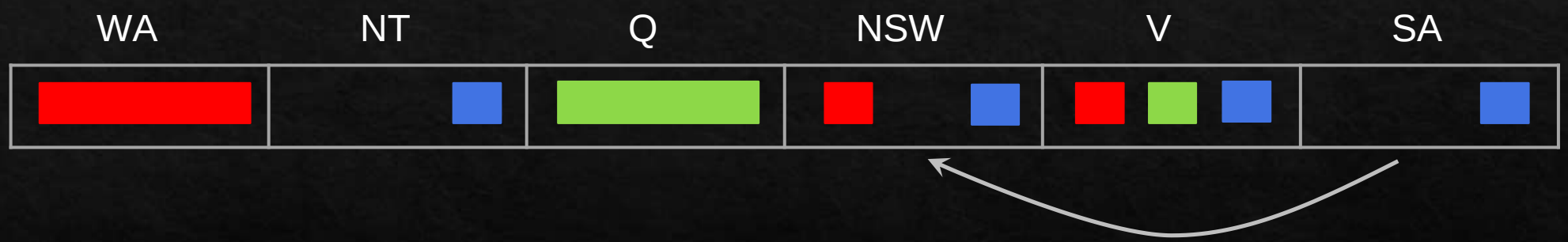
A CSP is consistent iff all the arcs are consistent



Arc Consistency of CSP

An arc $X \rightarrow Y$ is consistent iff $\forall x$ in the tail $\exists y$ in the head which could be assigned without violating any constraint

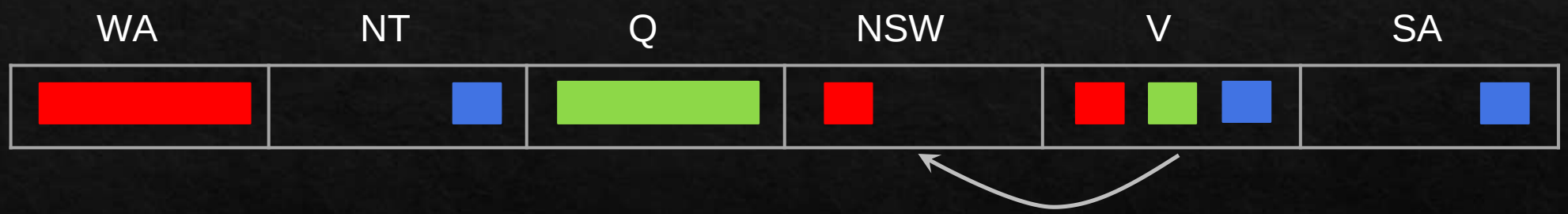
A CSP is consistent iff all the arcs are consistent



Arc Consistency of CSP

An arc $X \rightarrow Y$ is consistent iff $\forall x$ in the tail $\exists y$ in the head which could be assigned without violating any constraint

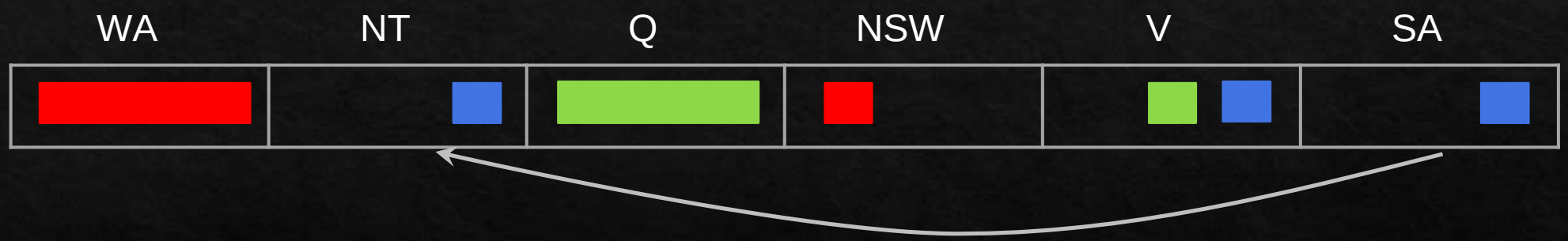
A CSP is consistent iff all the arcs are consistent



Arc Consistency of CSP

An arc $X \rightarrow Y$ is consistent iff $\forall x$ in the tail $\exists y$ in the head which could be assigned without violating any constraint

A CSP is consistent iff all the arcs are consistent



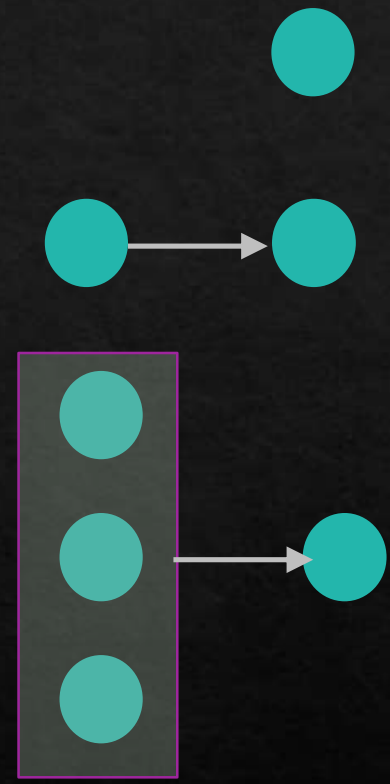
Comparison

Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV
USA	(> 1,000K)	(> 1,000K)	2K	60
n-Queens	(> 40,000K)	13,500K	(> 40,000K)	817K

CSP: Efficient Solvers

K-Consistency

- Consistency ⇒ Non violation of constraints
- Degrees of consistency
  - 1-consistency (Node consistency)
    - Unary constraints
  - 2-consistency (Arc consistency)
    - Any consistent assignment to one can be extended to other
    - Binary
  - K-consistency
    - Any assignment to $K-1$ nodes can be extended to the K^{th} node
  - $(K=2)$ ⇒ Arc consistency

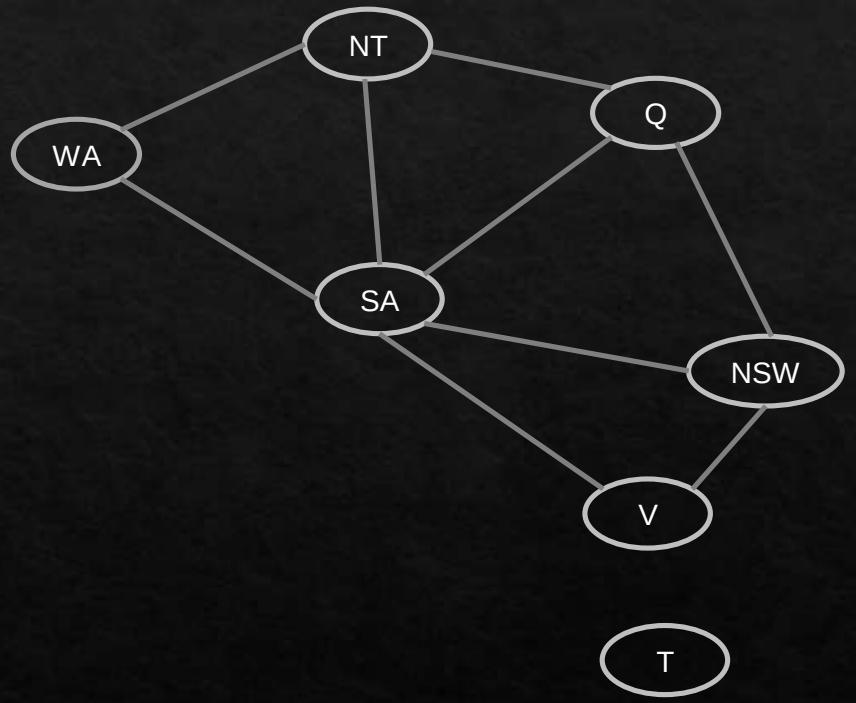


Strong K-Consistency

- Strong K-consistent ⇒ K-1, K-2, …, 1 consistent
- Strong N-consistency ensures solution without backtracking
  - Choose random assignment of any variable
  - Choose a new variable
  - 2-consistency ⇒ there is a choice consistent with the first
  - Choose another variable
  - 3-consistency ⇒ there is a choice consistent with the first 2
  - ……
- But, enforcing strong N-consistency as hard as having the solution
- Trade-off between arc consistency and K-consistency (e.g., 3-consistency aka Path Consistency)

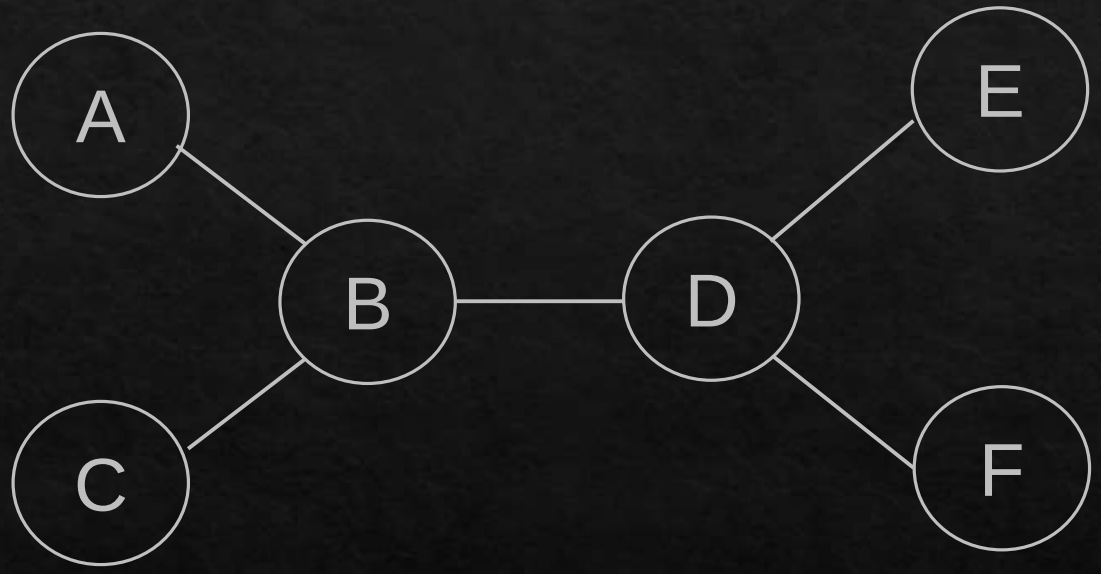
Problem Structure

- Independent Subproblems
  - Mainland and Tasmania do not interact
- How to identify independent subprobs?
  - Connected components of constraint graph
- What is the benefit?
  - Without decomposition running time: $O(d^n)$
  - Let the n variables broken into subproblem of c variables
  - Worst case: $O(\frac{n}{c} d^c)$ => Linear in n
  - Let $n = 80, c = 20, d = 2$
  - Without Decomposition: $2^{80} = 4$ billion years @ 10 million nodes/sec
  - With Decomposition: $4 \times 2^{20} = 0.4$ sec at 10 million nodes/sec



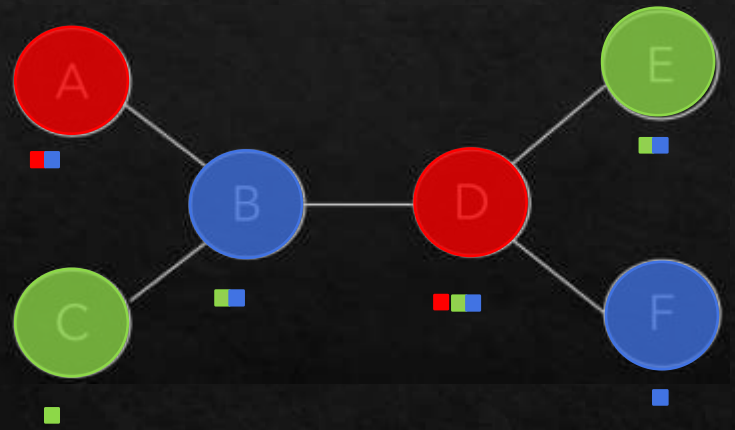
Very uncommon to find such a problem ☹

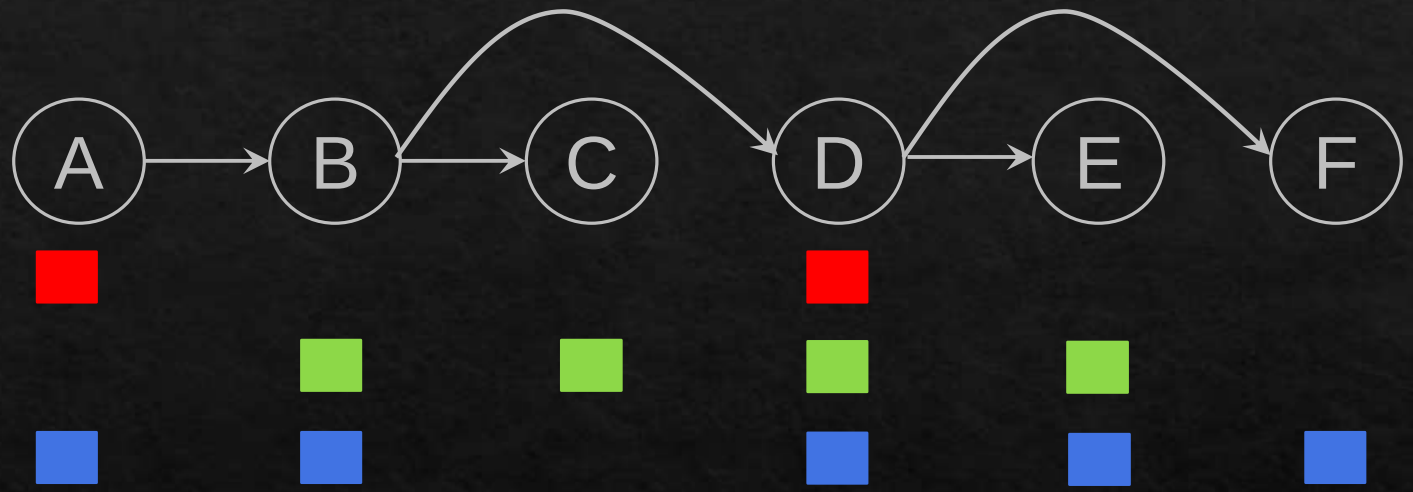
Tree Structured CSP



Theorem: If the constraint graph has no loop (i.e., tree), the CSP can be solved in $O(nd^2)$ in worst case

Tree Structured CSP: Algorithm





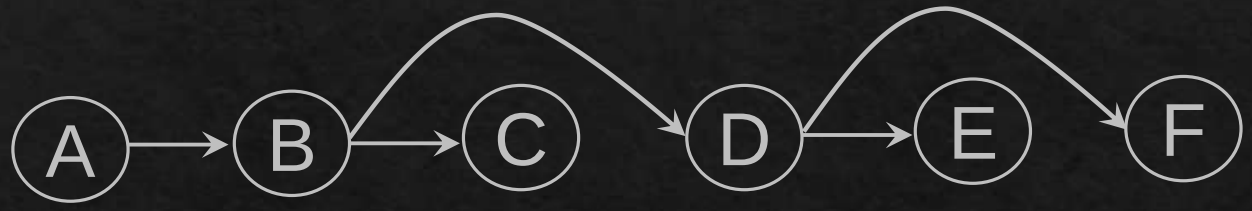
ORDER: Choose a root variable and order the other variables in such a way that parents precede children

REMOVE-BACKWARD: For i=n to 2, RemoveInconsistent(Parent[X_i], X_i)

ASSIGN-FORWARD: For i=1 to n, assign X_i consistently with Parent[X_i]

Runtime: $O(n.d^2)$

Tree Structured CSP: Algorithm



After backward pass all the root-to-leaf arcs are consistent

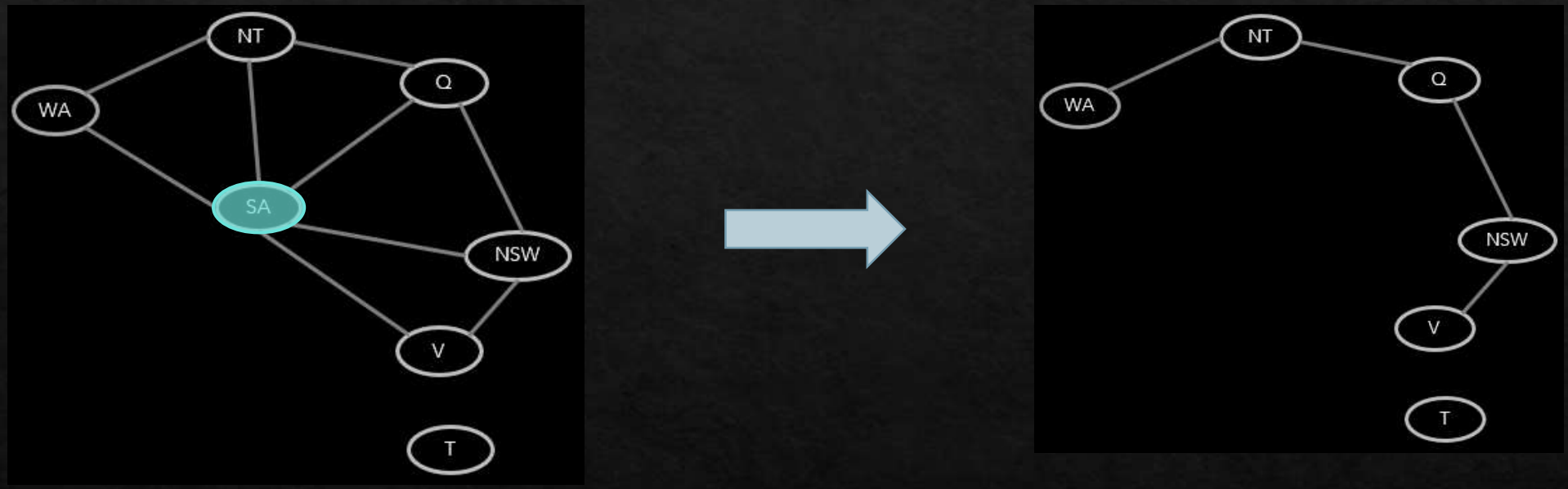
- After a backward pass, each $X \rightarrow Y$ has been made arc consistent
- Y's children have been processed before Y
- Y's cannot be reduced

Forward assignment will not backtrack

Does not work on constraint graphs with cycles. WHY?

Tree structured CSPs are not common either ☹

Convert to Tree Structured CSP



CONDITIONING: Forcefully instantiate a variable and prune the domains of the neighbors

CUTSET-CONDITIONING: Obtain a cutset of variables, removing those will leave the concept graph a tree. Instantiate (in all ways) the cutset.

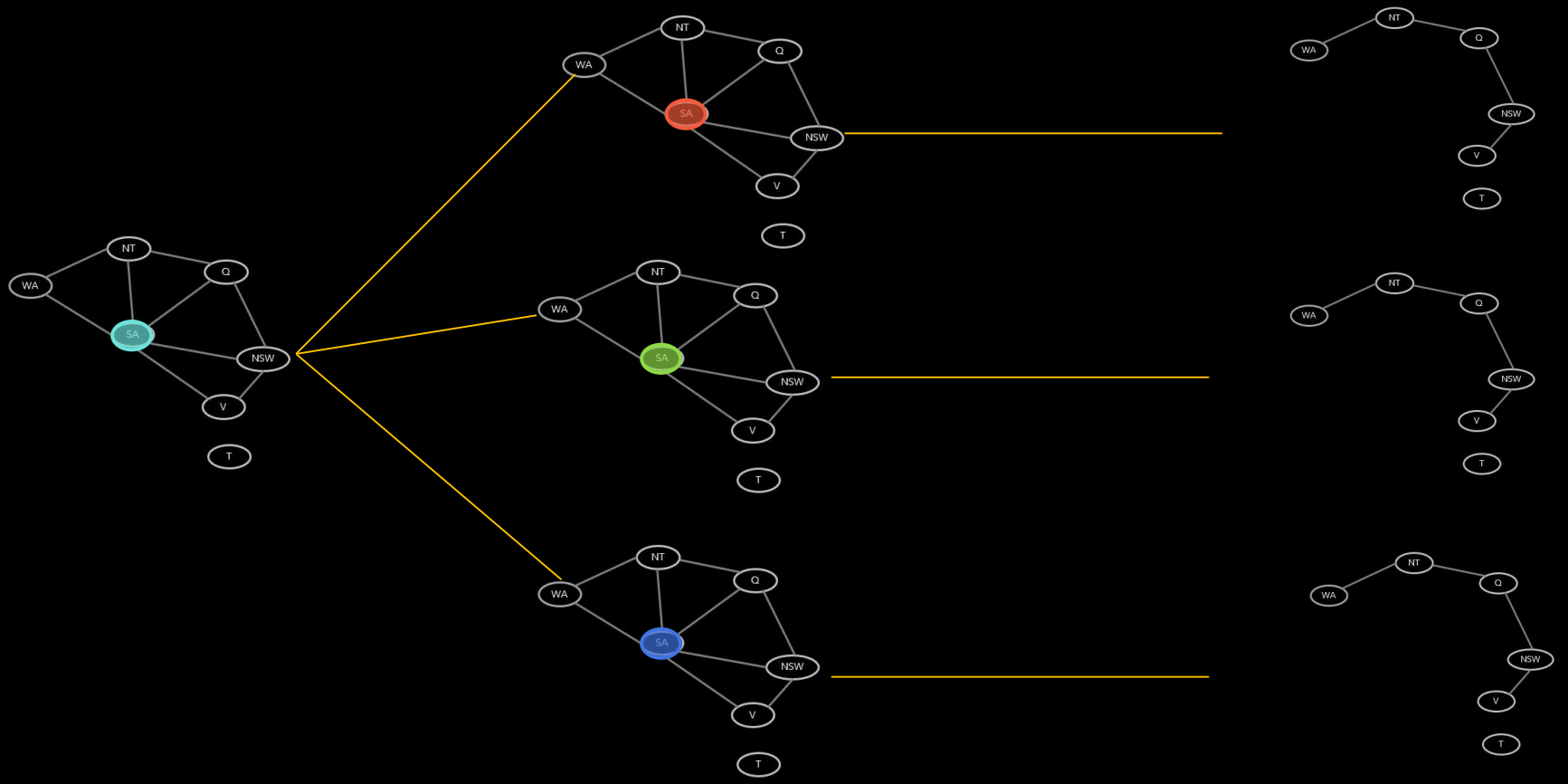
Cutset Conditioning

Choose a cutset

Instantiate cutset

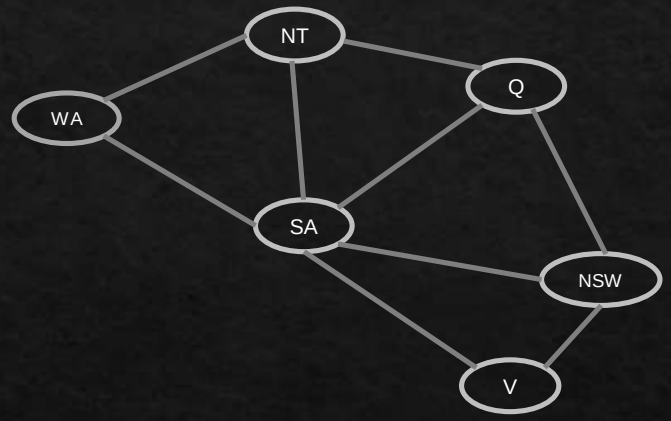
Compute residual

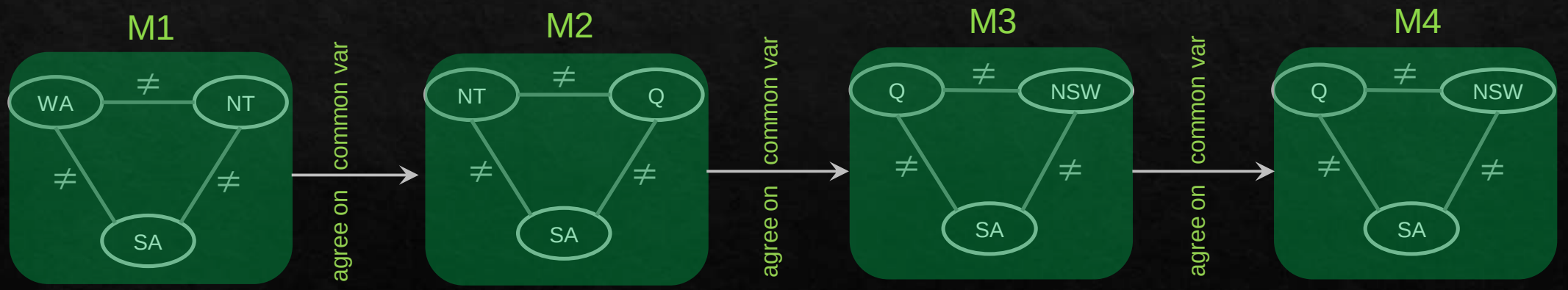
Solve residual Tree CSP



Runtime with cutset size c = $O(d^c.(n-c).d^2)$

Tree Decomposition





{(WA=r, NT=g, SA=b),
(WA=b, NT=r, SA=g),
……}

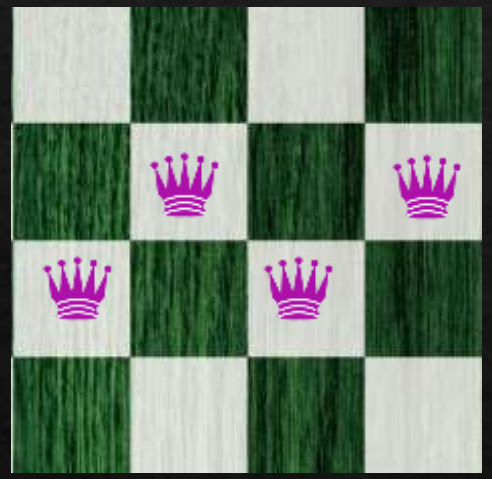
Agree: (M1,M2) ∈ {((WA=g, NT=g, SA=g), (NT=g, Q=b, SA=g)), ……..}

Iterative Improvement

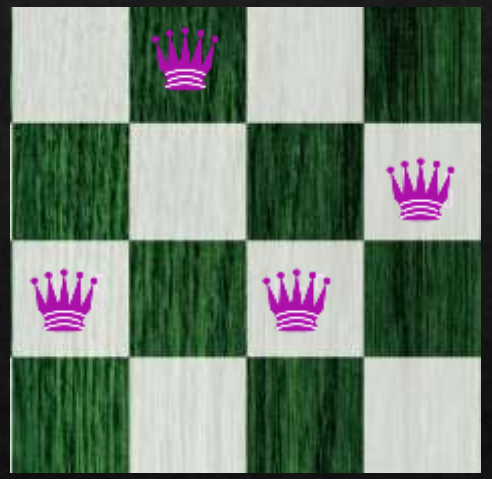
- Start with a complete assignment with unsatisfied constraints
- Iteratively change solution
  - Reassign variable values
  - No data structure like stack be maintained

- Algorithm
  - Variable selection: randomly select any conflicting variable
  - Value selection: min-conflict heuristics
    - Choose a value that violates the fewest constraints
    - (hill climb with h(n)=total number of constraints violated)

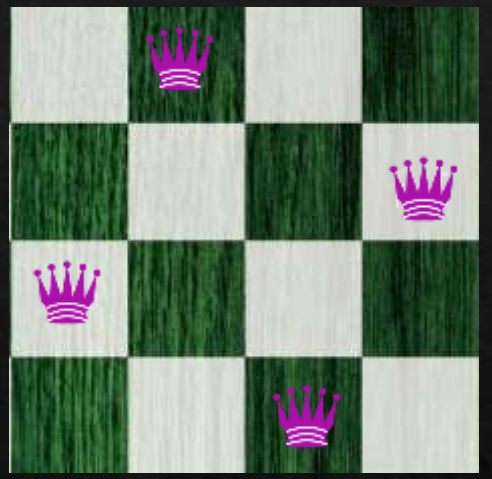
4-Queen Problem



$h = 5$

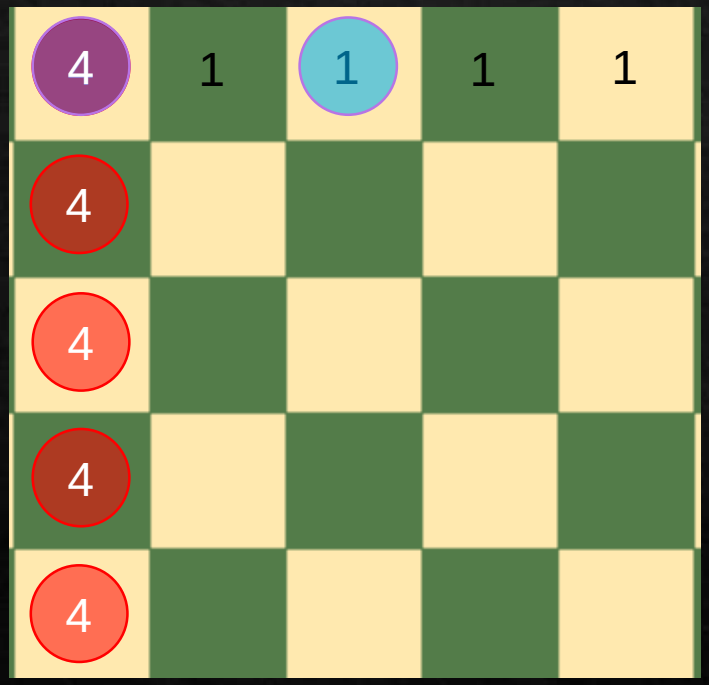


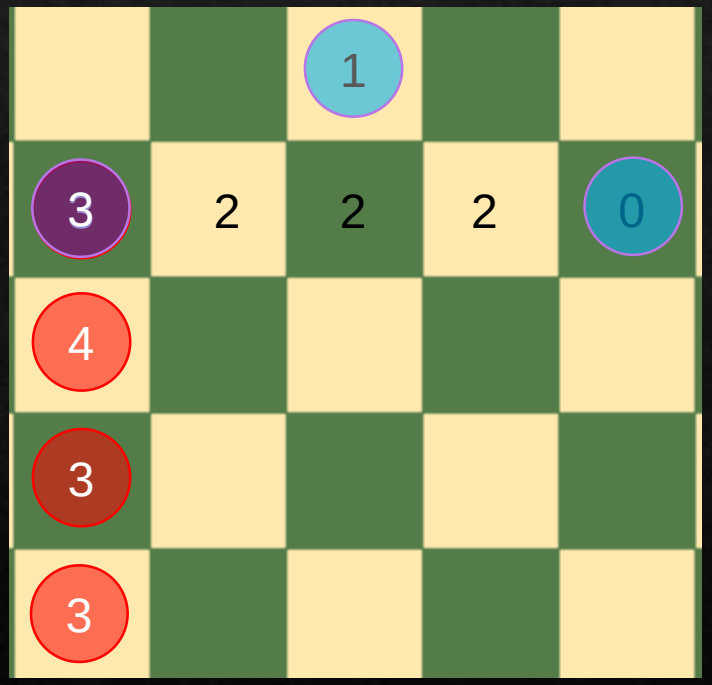
$h = 2$

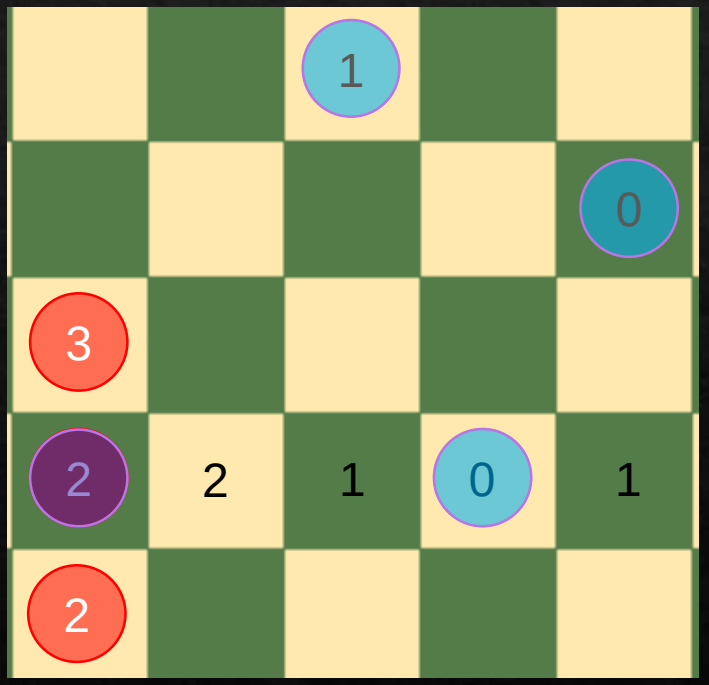


$h = 0$

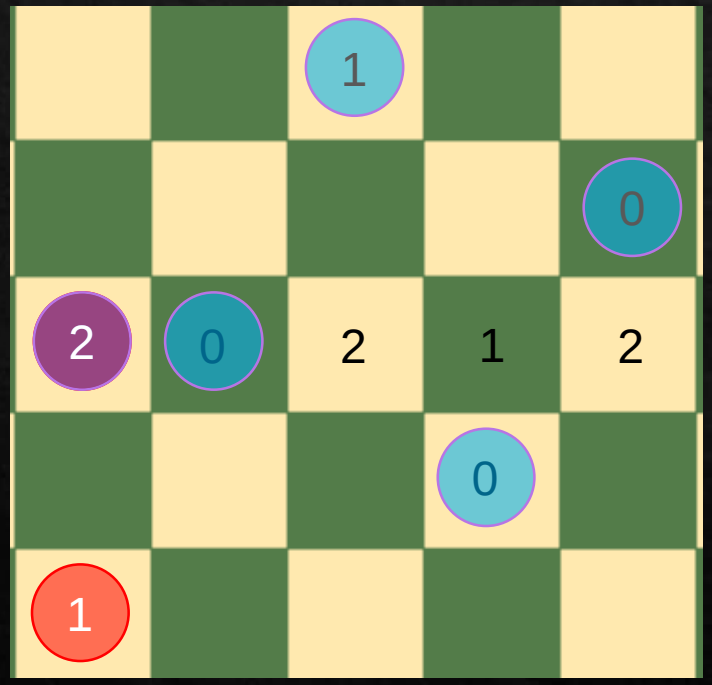
Min Conflict Heuristics

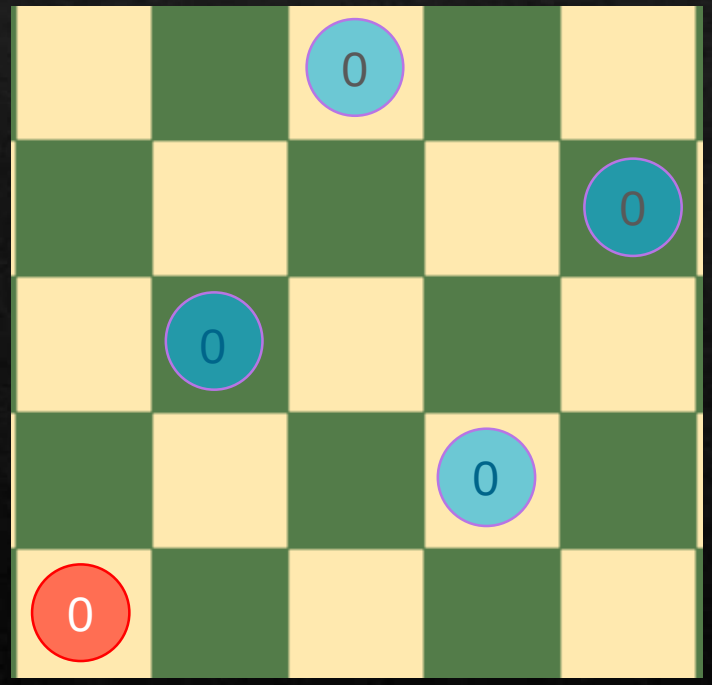


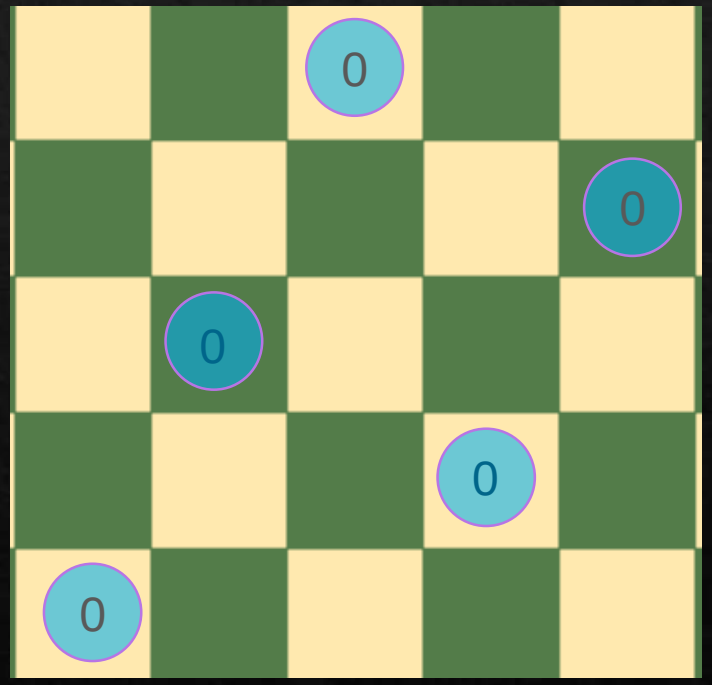


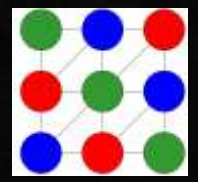


Min Conflict Heuristics





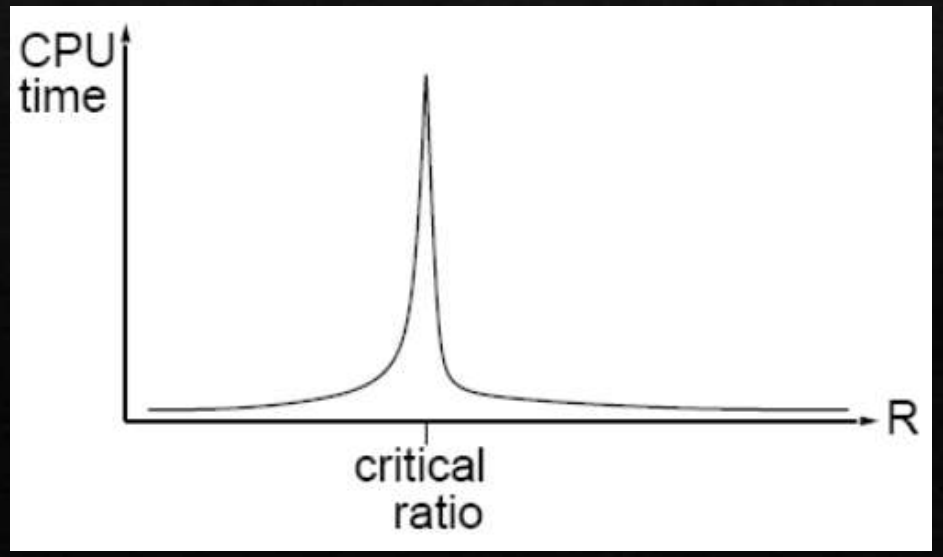




Min Conflict Heuristics

- Can solve n-queen problem for arbitrary n (~10,000,000) with high probability in constant time
- Similar performance on random CSPs except for a narrow range:

$$R = \frac{|Constraint\ set|}{|Variable\ set|}$$



Comparison

Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV	Min-Conflicts
USA	(> 1,000K)	(> 1,000K)	2K	60	64
n-Queens	(> 40,000K)	13,500K	(> 40,000K)	817K	4K

Summary

- CSP: Special instance of search problem
  - Generic (i.e., problem agnostic)
- Basic algorithm: Backtracking
- Speedup: Ordering, Filtering, Problem structure
- Iterative min-conflict (More Practical)