Constraint Satisfaction Problem I

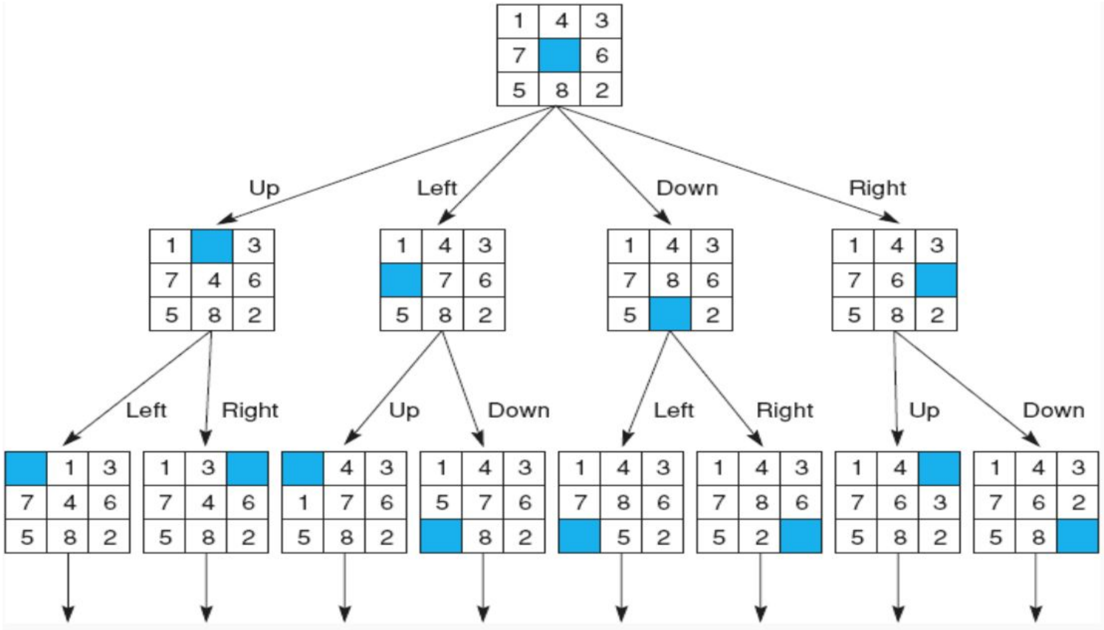
CSE 4617: Artificial Intelligence

Isham Tashdeed
Lecturer, CSE

When can search be used?

Assumptions:

- Single agent
- All the states can be fully observed
- States are discrete
- Actions taken by the agent are deterministic



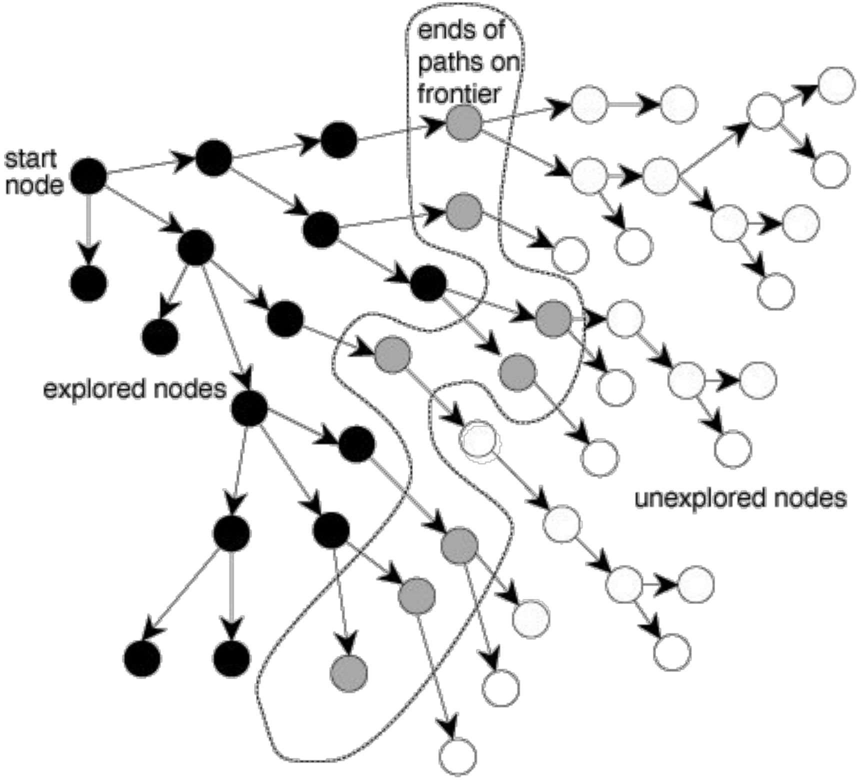
When can search be used?

Planning:

- The path leading to the goal is as important as reaching the goal
- Paths have various cost/depth
- Heuristics are a helping hand

Identification:

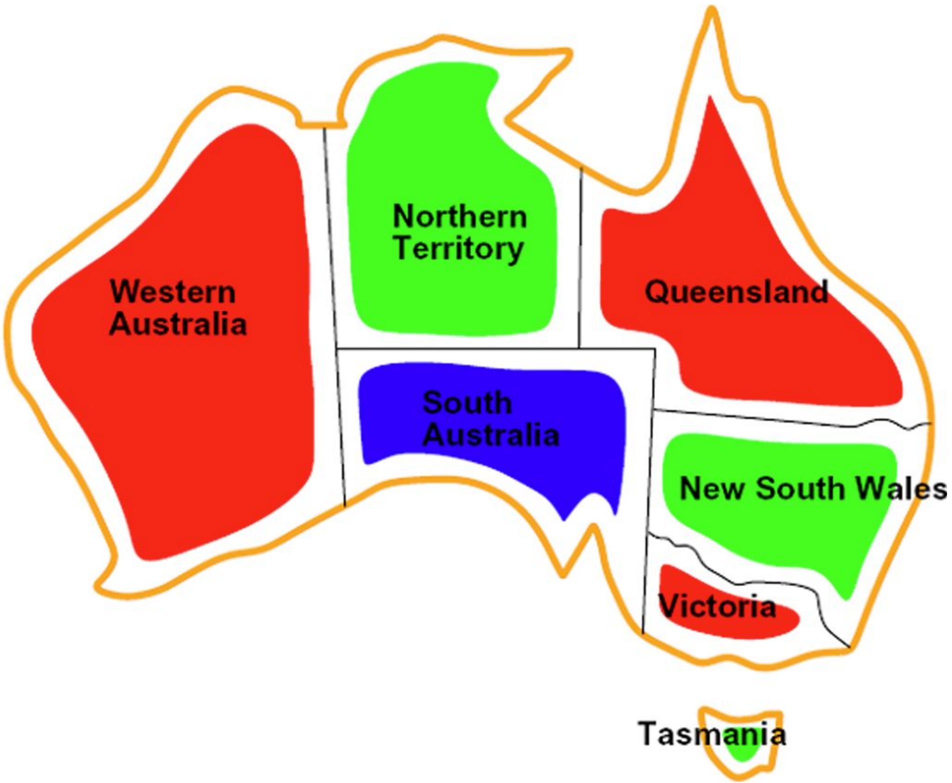
- Reaching the goal is important, not the path
- All paths are the same depth (How?)
- CSPs are special identification problem

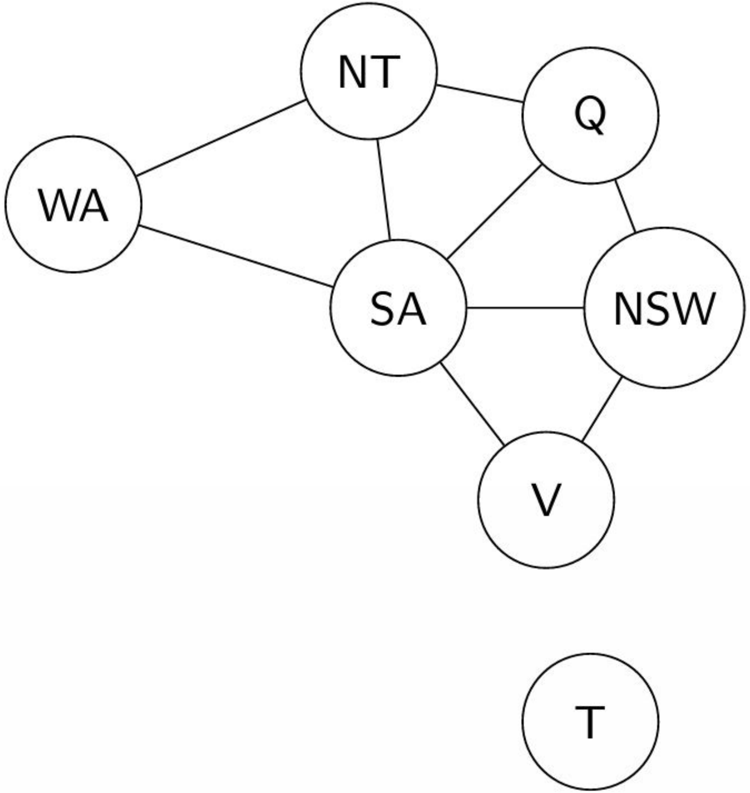


Constraint Satisfaction Problem

Examples of CSPs

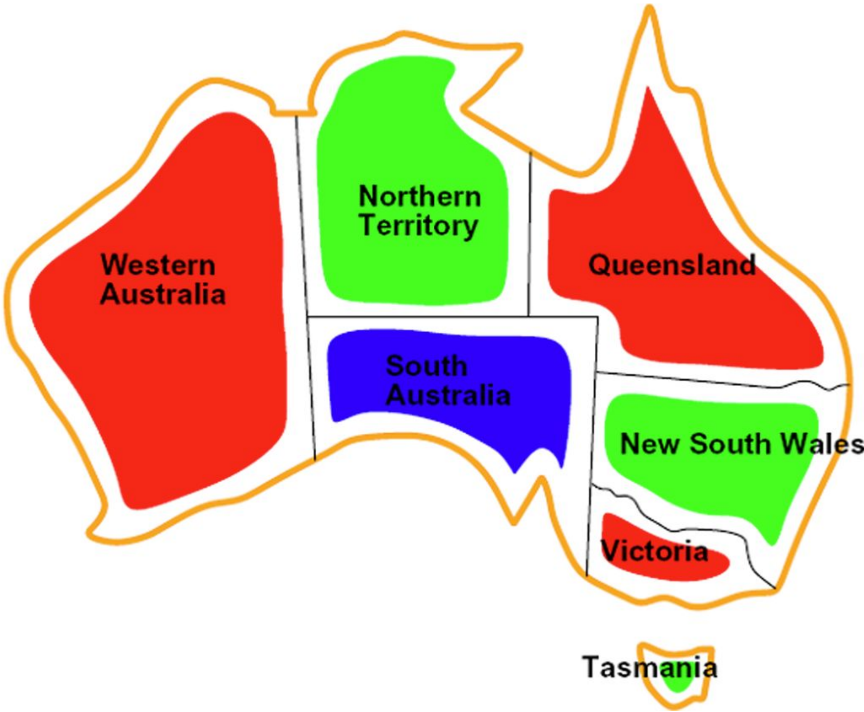
Examples of CSPs





Examples of CSPs

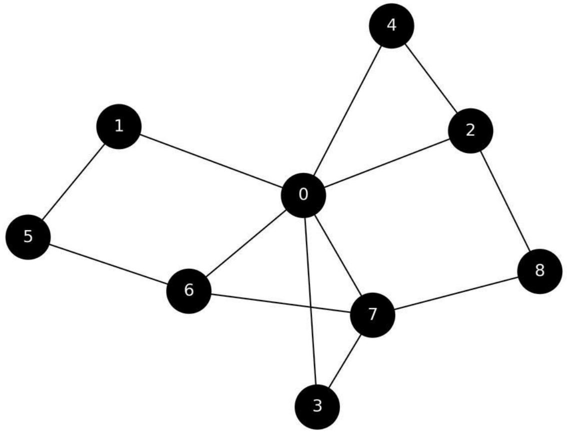
- Variables
  - WA, NT, Q, NSW, V, SA, T
- Domains
  - $D = \{\text{red, green, blue}\}$
- Constraints $\rightarrow$ Adjacent regions must be different colored
  - Implicit: WA $\neq$ NT, ...
  - Explicit: (WA, NT) $\in$ {(red, green), (red, blue), (blue, green), ...}
- Solution
  - An assignment of variables that satisfy all constraints



Examples of CSPs

CSPs are a special subset of search problems:

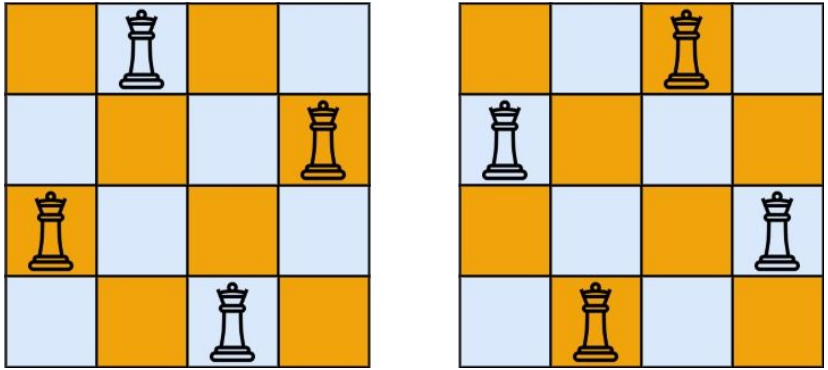
- A state is defined by variable X_i with values from a domain D
- Goal test is a set of constraints denoting valid combinations of values assigned to X_i

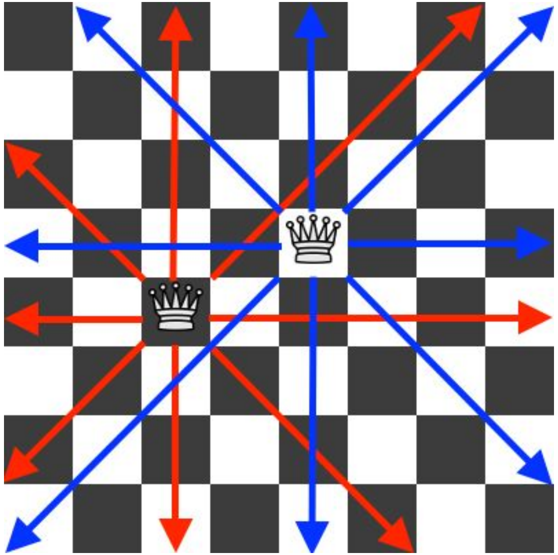


Why is it different from regular search problems?

Examples of CSPs

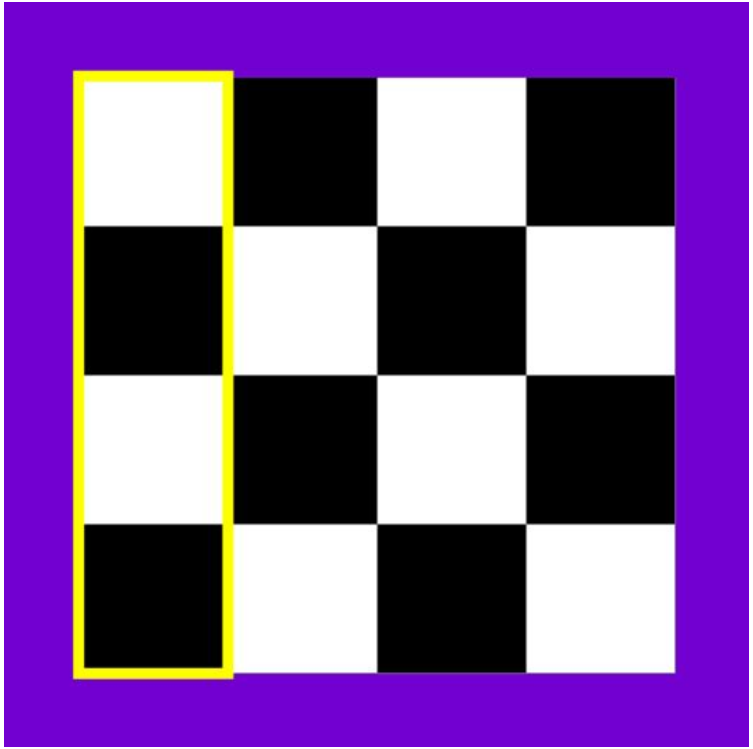
- Variables
  - X_{ij}
  - Total $n \times n$ variables
- Domains
  - $D = \{0, 1\}$
- Constraints → No queen should attack the others
  - Explicit Constraint?
  - What if all variables are 0?
- Solution
  - An assignment of variables that satisfy all constraints





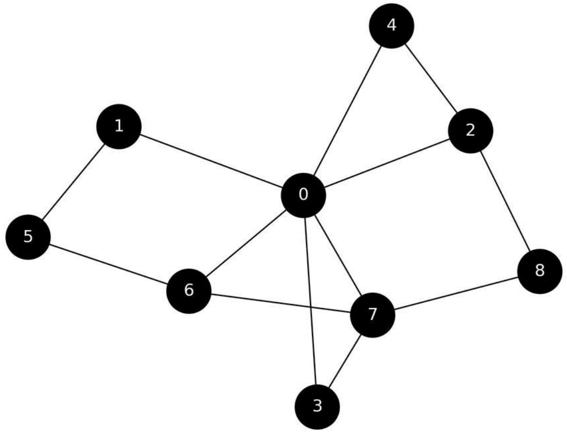
Examples of CSPs

- Variables
  - Q_k
  - Total $n \times n$ variables
- Domains
  - $D = \{1, 2, 3, \ldots, N\}$
- Constraints → No queen should attack the others
  - Explicit Constraint?
  - $(Q_i, Q_j) \in \{(1, 3), (1, 4), \ldots\}$ where $|i - j| = 1$
- Solution
  - An assignment of variables that satisfy all constraints



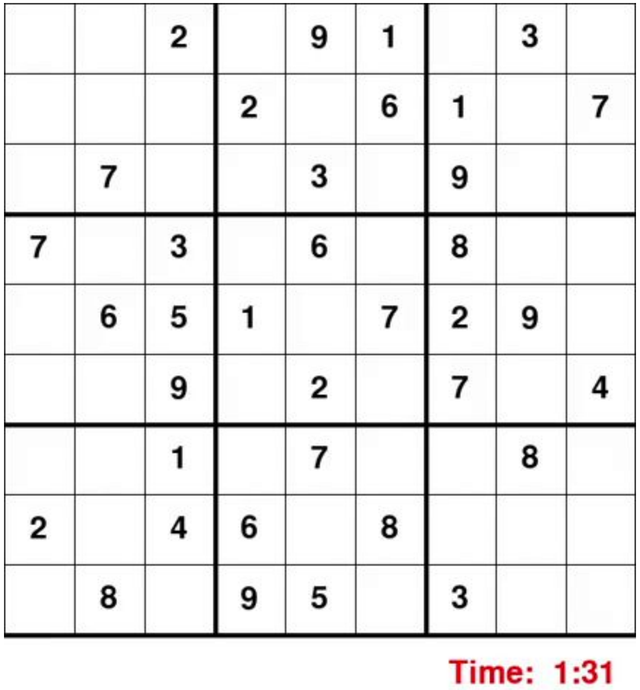
Constraint Graphs

- Binary CSPs → Each constraint relates to two variables at most
- Binary constraint graph → Nodes are variables, arcs define the constraints
- Use the graph structure to speed up the search algorithm → But how?



Examples of CSPs

- Variables
    - Each empty square
- Domains
    - $D = \{1, 2, 3, \dots, 9\}$
- Constraints → No queen should attack the others
    - Unary constraint for given values
    - 9-way all-diff for each column
    - 9-way all-diff for each row
    - 9-way all-diff for each region
    - No empty squares left
- Solution
    - An assignment of variables that satisfy all constraints

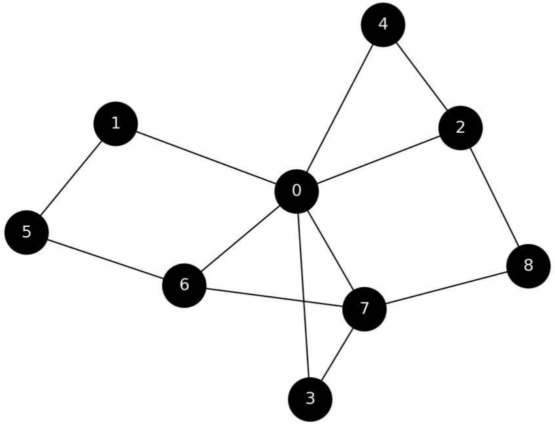


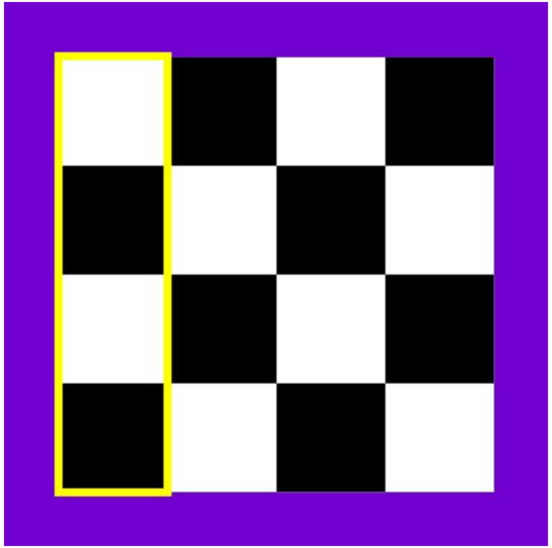
Types of Constraints

- Unary constraints involve a single variable
  - SA ≠ green
- Binary constraints involve pairs of variables
  - SA ≠ WA
- Higher-order constraints involve more that 2 variables
  - Sudoku column constraint
  - Sudoku row constraint
- Preferences
  - Often represented as cost of variable assignment → Preferred values have low cost
  - Optimization problem on top of a CSP

Solving CSPs

- State → The partial assignment of values
- Start State → Empty assignment → {}
- Successor Function → Assigns a value to an unassigned variable
- Goal test → All the variables have values assigned to them & No constraint is violated



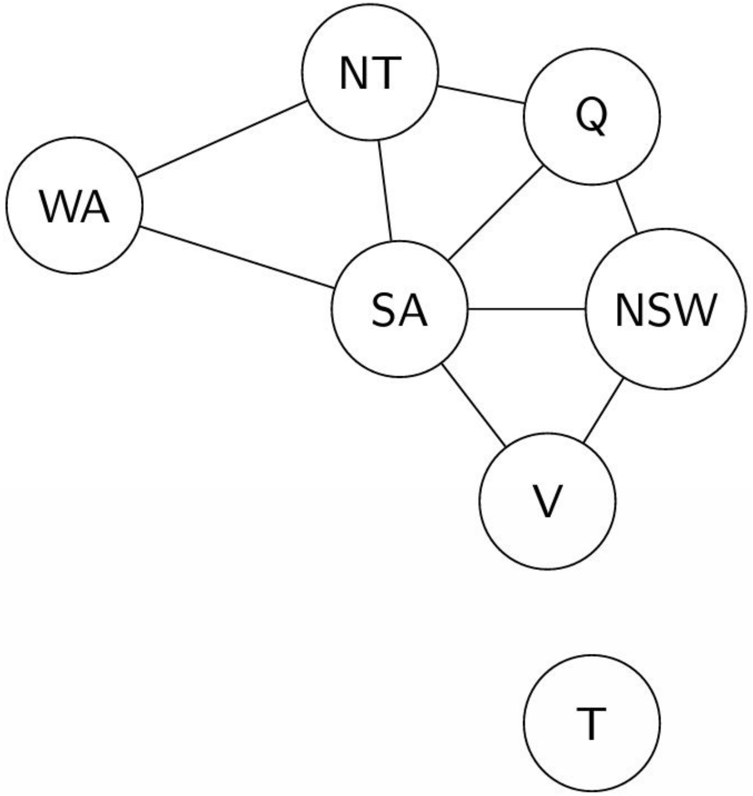


Solving CSPs

- State → The partial assignment of values
- Start State → Empty assignment → {}
- Successor Function → Assigns a value to an unassigned variable
- Goal test → All the variables have values assigned to them & No constraint is violated

What happens if we use BFS?

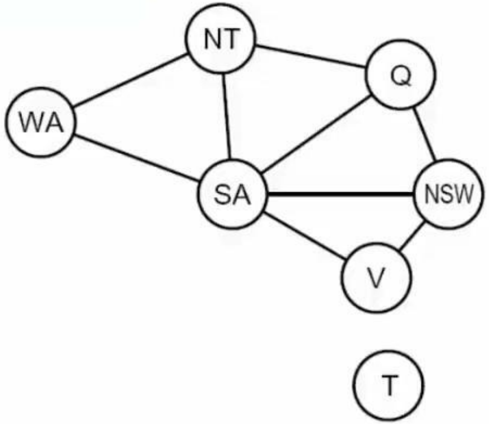
What happens if we use DFS? → Naive DFS



Naive DFS

Search Methods

- What would BFS do?
- What would DFS do?



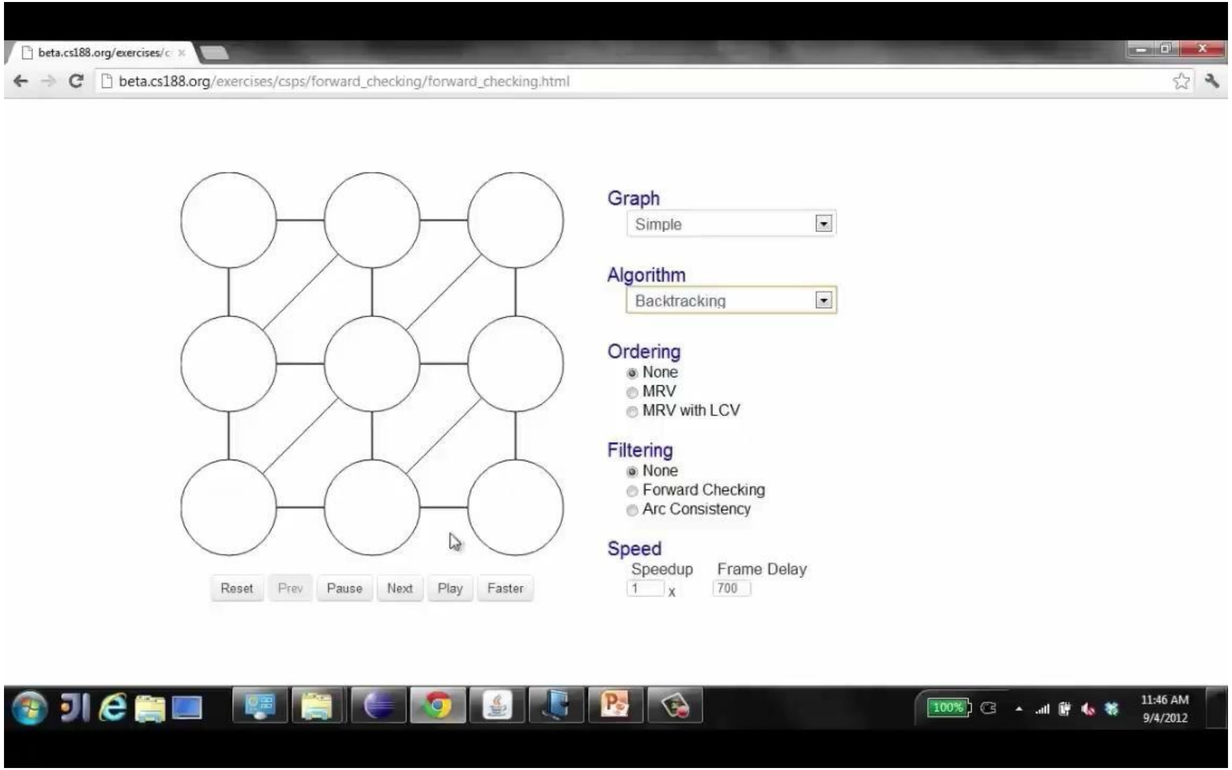
[demo: dfs]

Backtracking Search

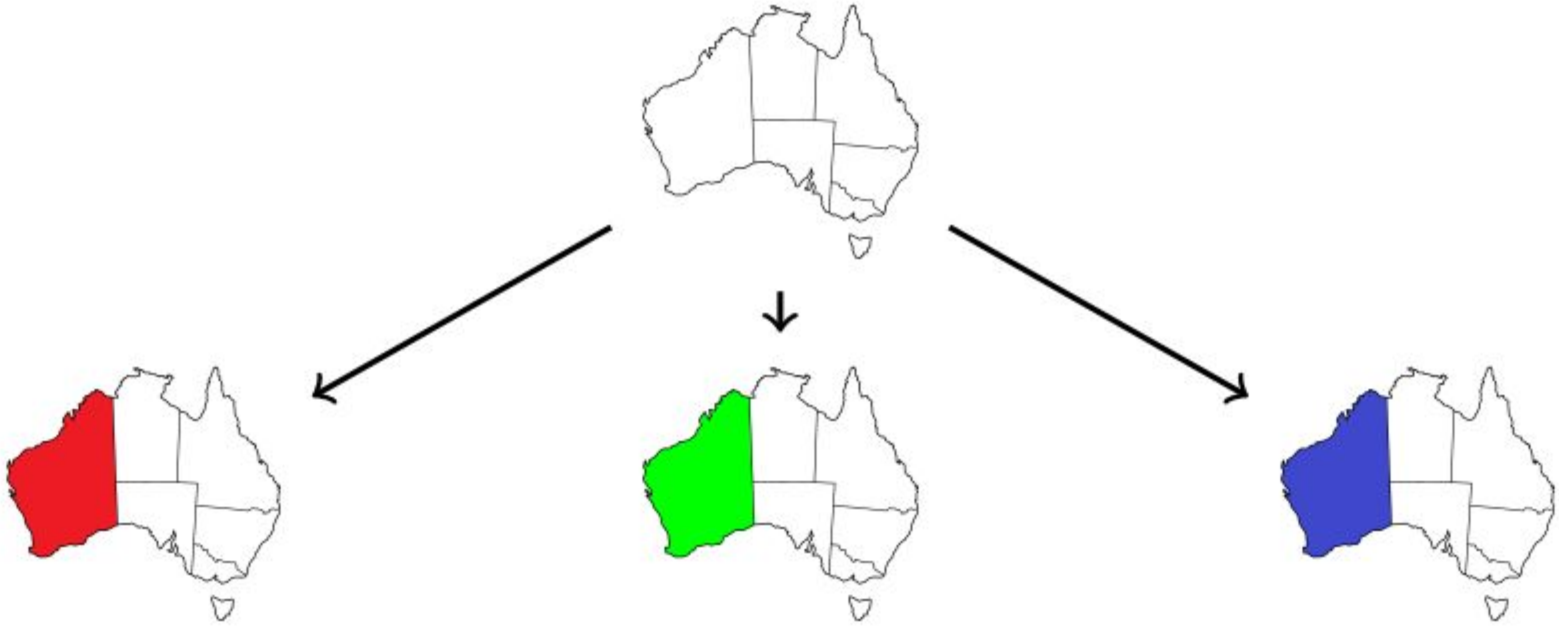
Backtracking Search

- One variable at a time
  - Only need to consider assignments to a single variable at each step
  - Variable assignments are commutative → Any ordering is OK!
  - [WA = red then NT = green] same as [NT = green then WA = red]
- Check constraints as you go
  - Consider only values which do not conflict with previous assignments
  - “Incremental goal test”
- DFS + these two improvements = Backtracking Search

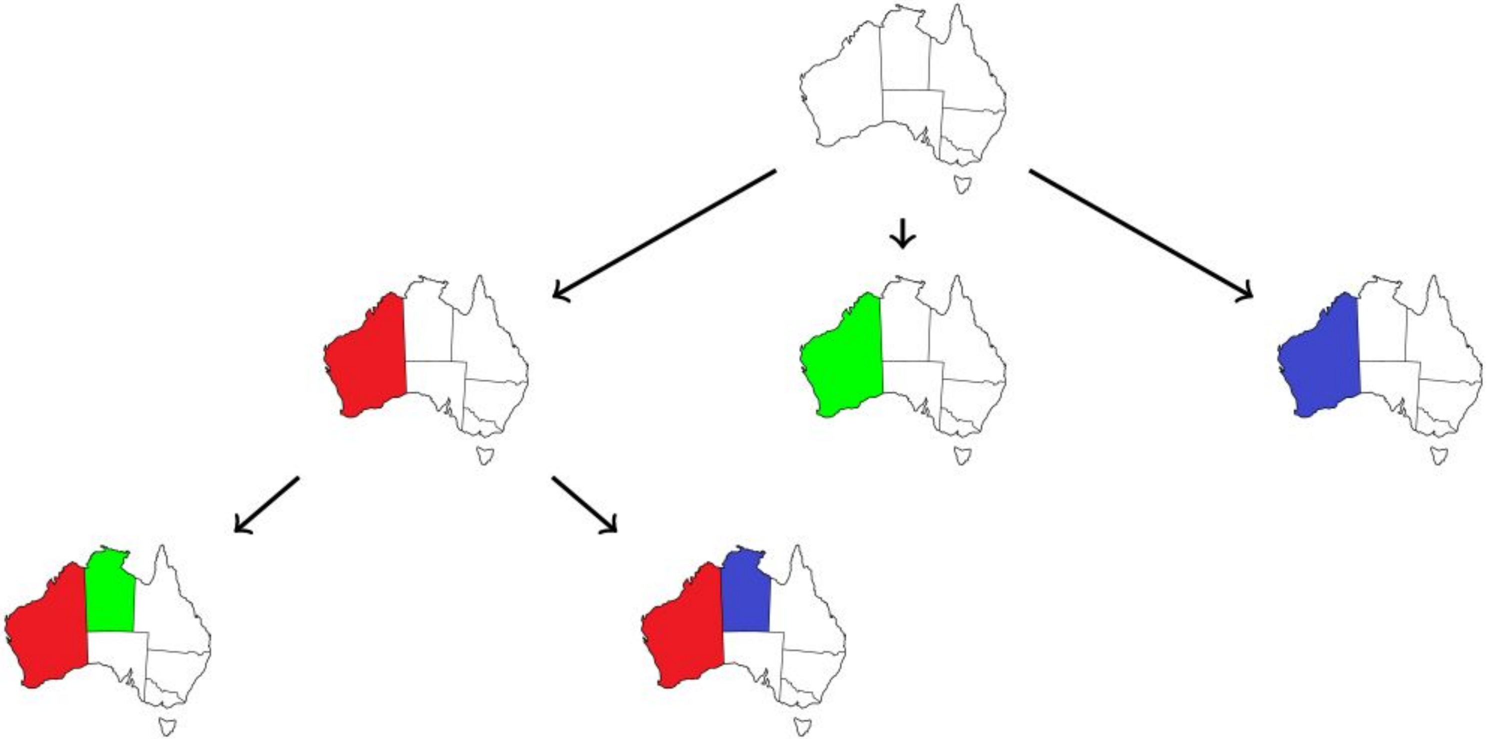
Backtracking Search



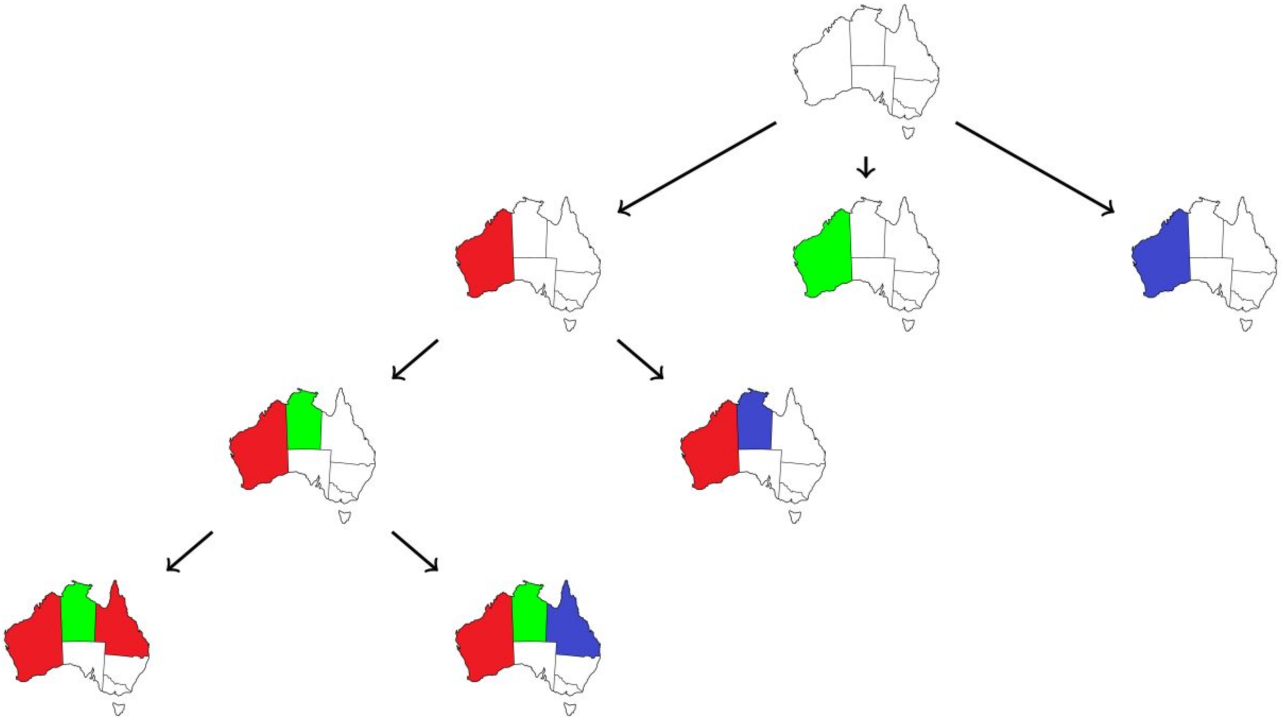
Backtracking Search



Backtracking Search



Backtracking Search



Backtracking Search

```
BACKTRACKING-SEARCH (csp) → returns a solution or failure
    return RECURSIVE-BACKTRACKING ({}, csp)


RECURSIVE-BACKTRACKING (assignment, csp) → returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for value in ORDER-DOMAIN-VALUE(var, assignment, csp):
        if value is consistent with assignment given CONSTRAINTS[csp]:
            then add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING (assignment, csp)
            if result is not failure then return result
            remove {var = value} from assignment
    return failure
```

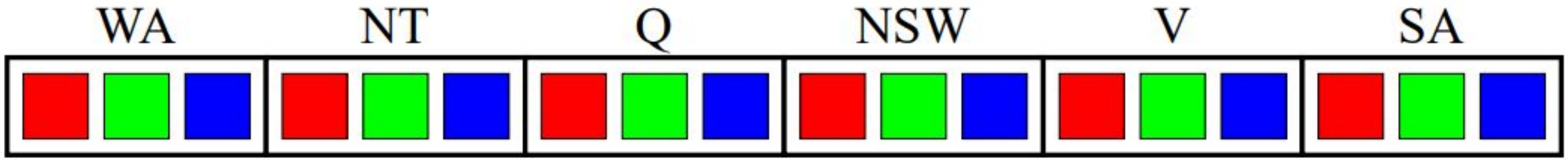
How to Improve Backtracking Search

- Filtering
  - Limiting our choices for variable assignment
  - Detects an inevitable failure early
- Ordering
  - In what order should we assign variables
  - Does it actually matter?
- Structure
  - Can we exploit the structure of a problem?

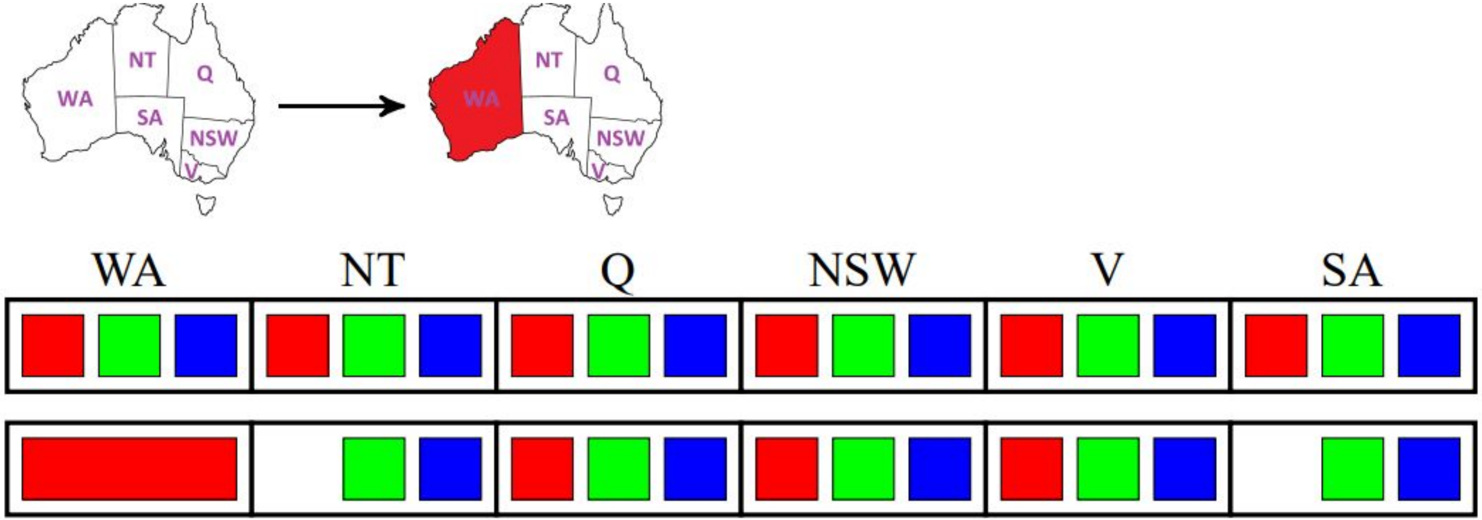
Filtering: Forward Checking

Filtering: Forward Checking

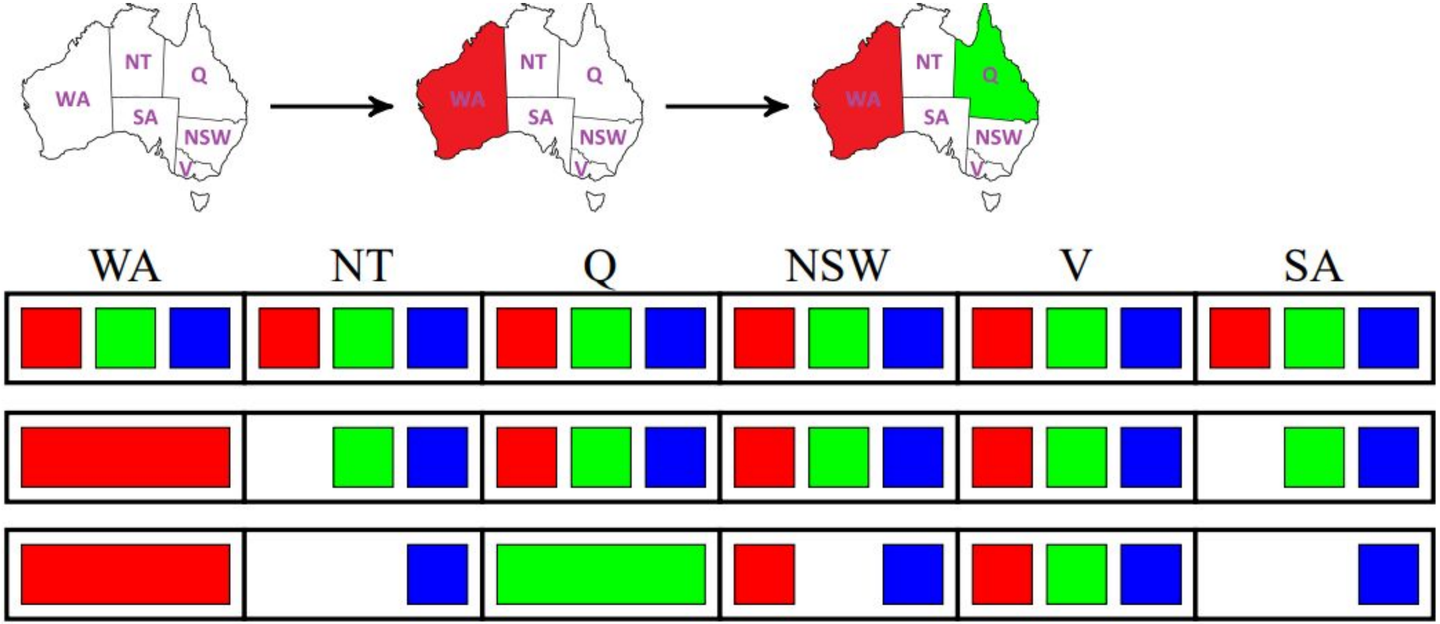
- Keep track of the domains for all unassigned variables
- Remove values that violate a constraint when added



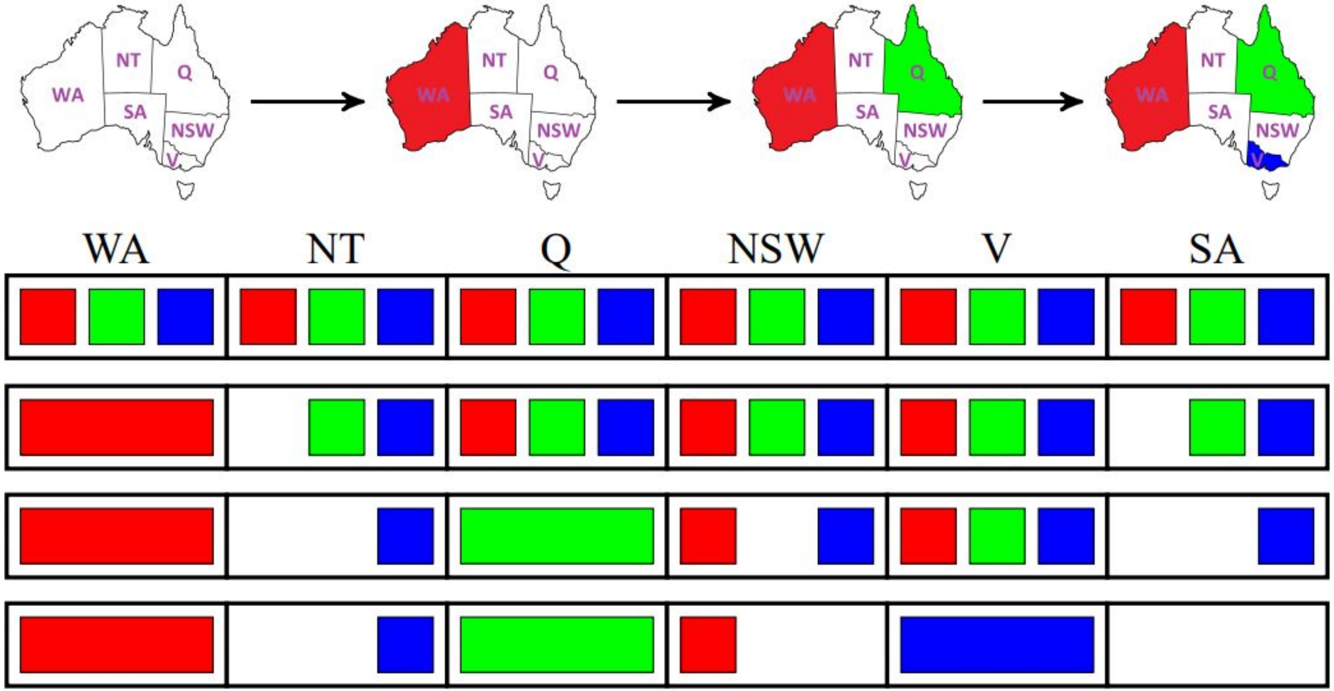
Filtering: Forward Checking



Filtering: Forward Checking

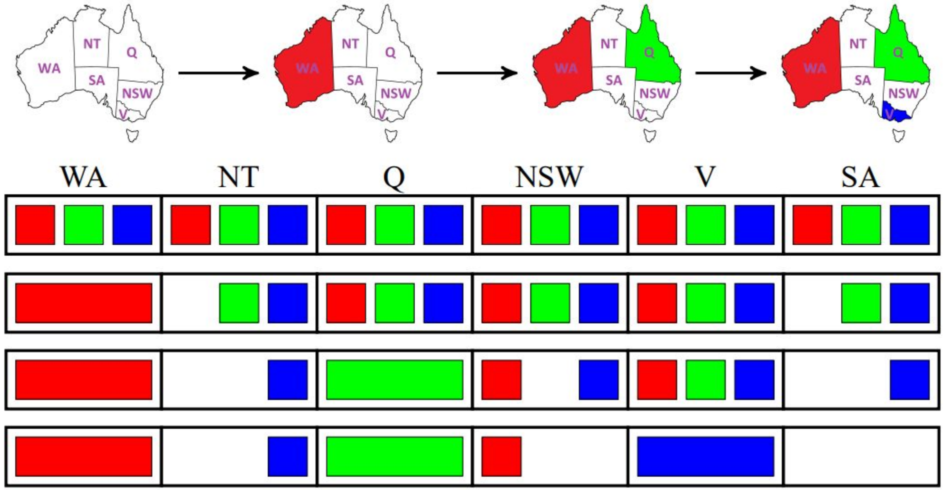


Filtering: Forward Checking



Filtering: Forward Checking

- Propagates information from assigned to unassigned variables
- Does not check pairs of unassigned variables for early detection of failures

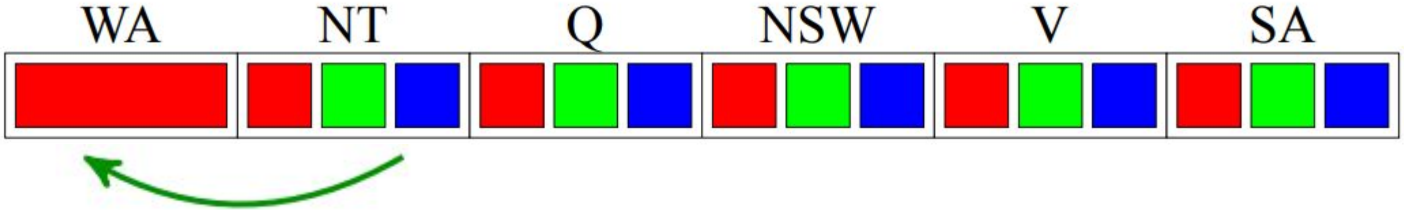


We need to propagate the changed constraints from variable to variable

Filtering: Arc Consistency

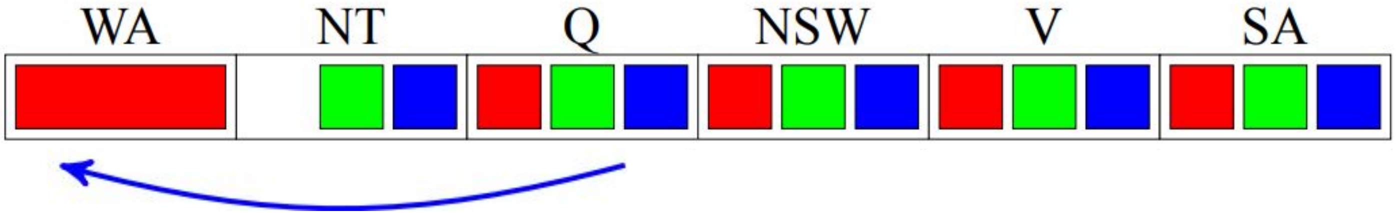
Filtering: Forward Checking

An arc, X → Y, is consistent if and only if for every *x* in the tail, there is some *y* in the head which could be assigned without violating a constraint



Filtering: Forward Checking

An arc, X → Y, is consistent if and only if for every *x* in the tail, there is some *y* in the head which could be assigned without violating a constraint

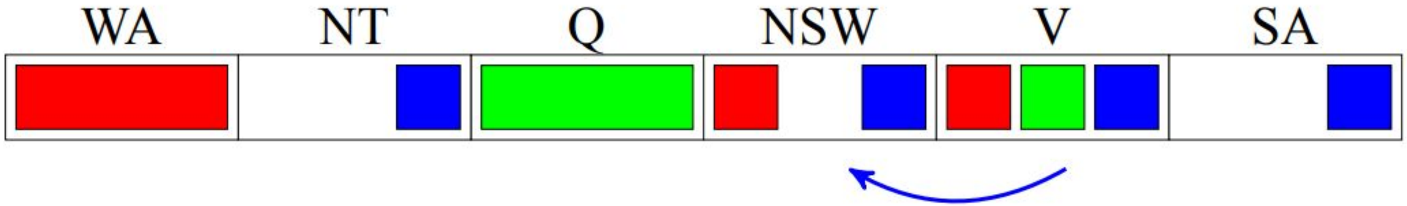


Delete from the tail!

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

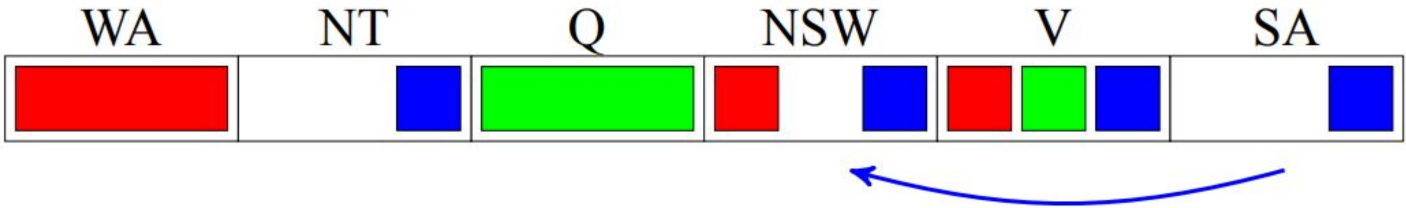
Filtering: Arc Consistency

A simple form of propagation makes sure **all arcs** are consistent



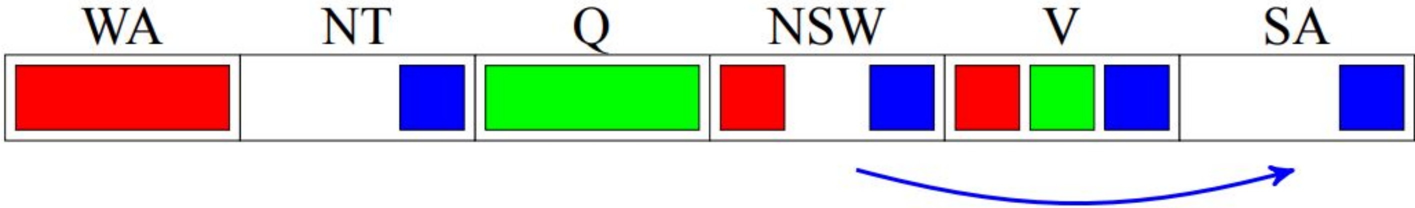
Filtering: Arc Consistency

A simple form of propagation makes sure **all arcs** are consistent



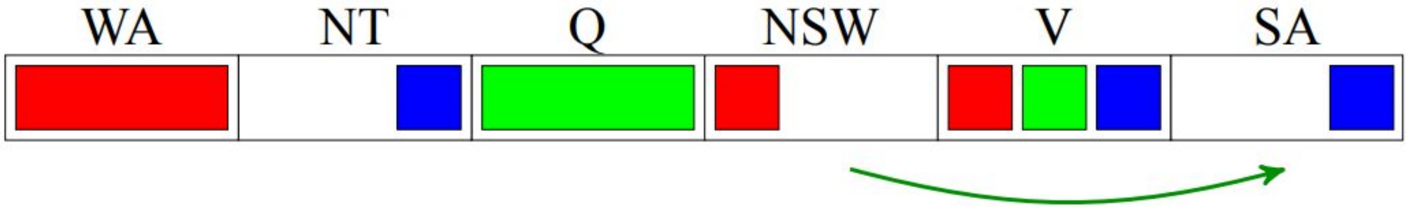
Filtering: Arc Consistency

A simple form of propagation makes sure **all arcs** are consistent



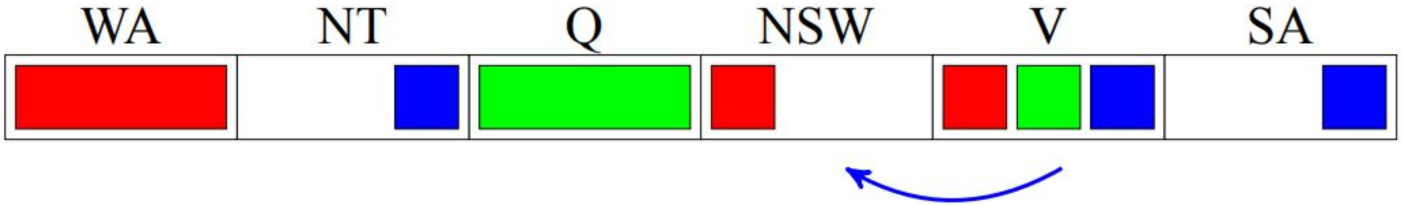
Filtering: Arc Consistency

A simple form of propagation makes sure **all arcs** are consistent



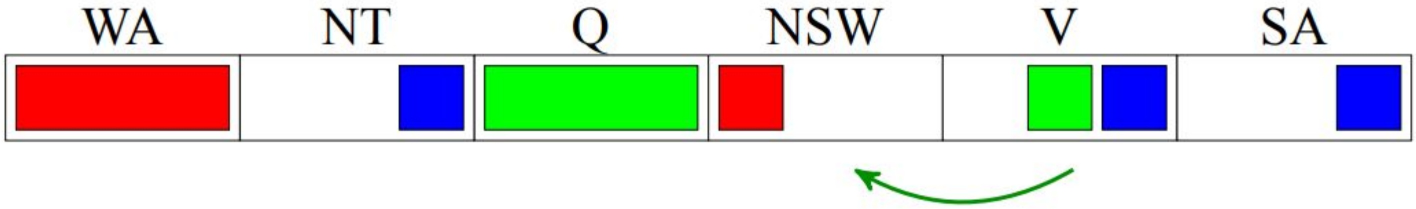
Filtering: Arc Consistency

A simple form of propagation makes sure **all arcs** are consistent



Filtering: Arc Consistency

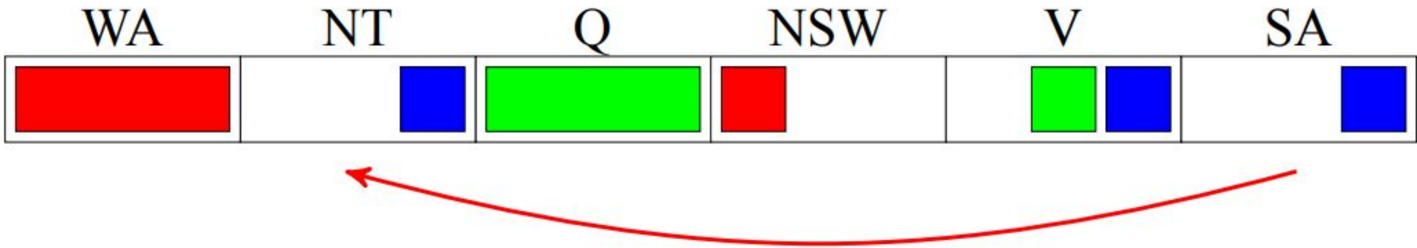
A simple form of propagation makes sure **all arcs** are consistent



If X loses a value, arcs leading to X need to be rechecked!

Filtering: Arc Consistency

A simple form of propagation makes sure **all arcs** are consistent



If X loses a value, arcs leading to X need to be rechecked!

Filtering: Arc Consistency

A simple form of propagation makes sure **all arcs** are consistent

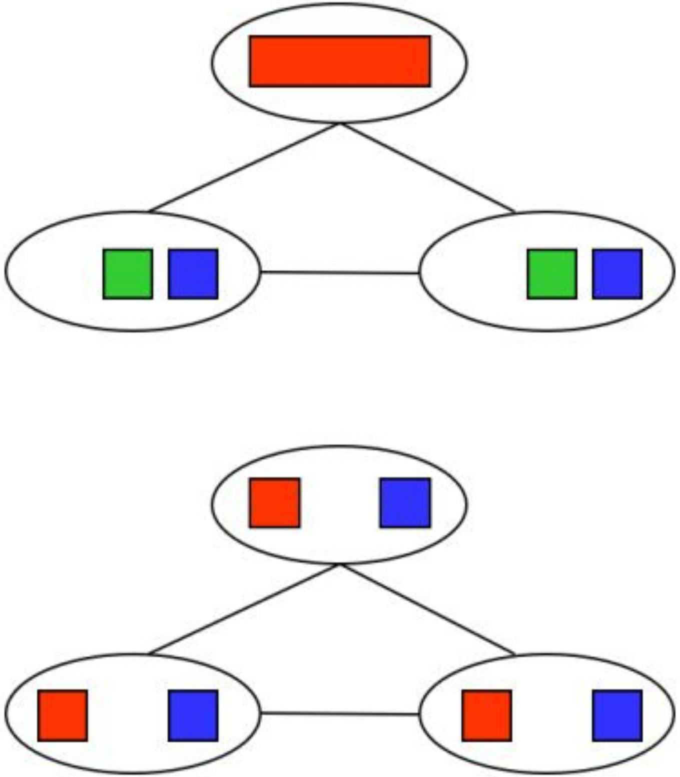
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment → Complexity?

Filtering: Arc Consistency

After enforcing arc consistency:

- Can have one solution left
- Can have multiple solutions left
- Can have no solutions left (and not know it)

Arc consistency only checks pairs :(

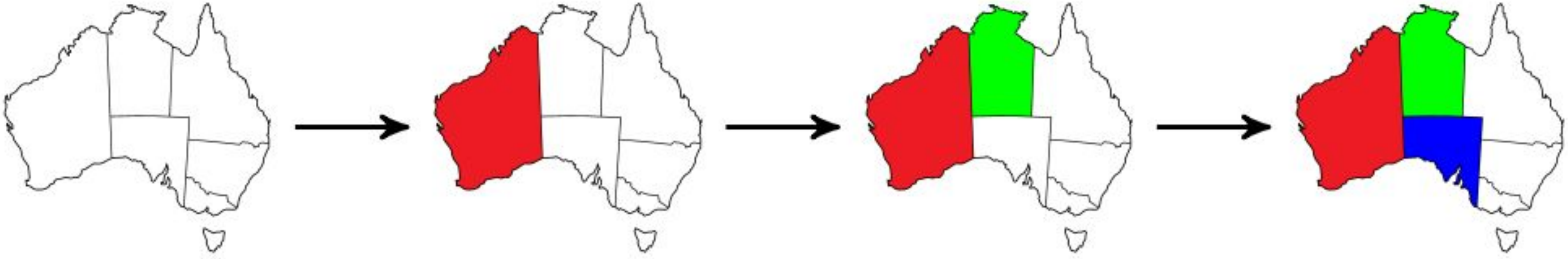


Ordering

Ordering: Minimum Remaining Values

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

- Why min rather than max? → Fails faster
- This is also called the most constrained variable



Ordering: Least Constrained Value

- Given a choice of variable, choose the least constraining value
- The one that rules out the fewest values in the remaining variables
- Must do further filtering to determine which value is the least constrained
- Why least rather than most?
    - Leave more options for others

Additional Resources

- Backtracking Search Simulator
- Procedural Generation using Constraint Satisfaction

Thank you