

# Artificial Intelligence

## 5. Constraint Satisfaction Problems

---

CSPs as Search Problems, Solving  
CSPs, Problem Structure

*Wolfram Burgard, Andreas Karwath,  
Bernhard Nebel, and Martin Riedmiller*

# Contents

---

- What are CSPs?
- Backtracking Search for CSPs
- CSP Heuristics
- Constraint Propagation
- Problem Structure

# Constraint Satisfaction Problems

- In **search problems**, the state does not have a structure (everything is in the data structure). In CSPs, states are explicitly represented as variable assignments.
- A CSP consists of
  - a set of **variables**  $\{x_1, x_2, \dots, x_n\}$  to which
  - **values**  $\{d_1, d_2, \dots, d_k\}$  can be assigned
  - such that a set of **constraints** over the variables is respected
- A CSP is solved by a **variable assignment** that satisfies all **given constraints**.
- *Formal representation language* with associated general inference algorithms

# Example: Map-Coloring



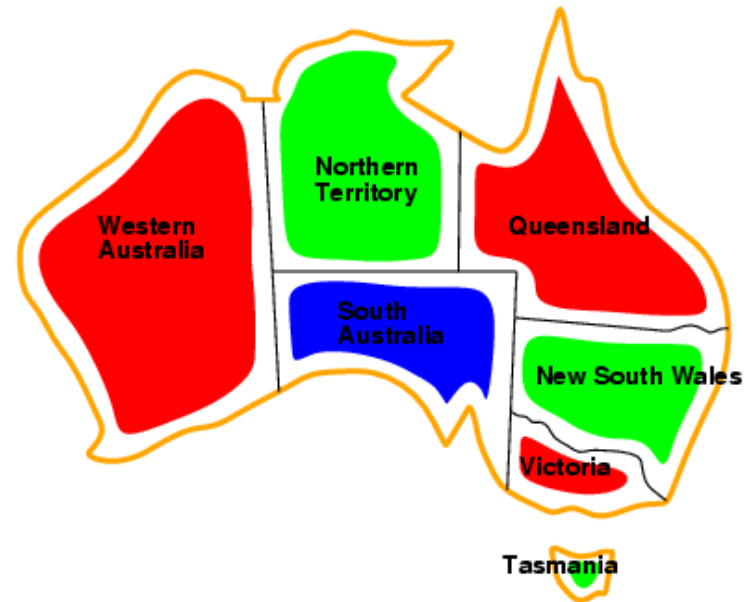
- Variables:  $WA, NT, SA, Q, NSW, V, T$
- Values:  $\{red, green, blue\}$
- Constraints: adjacent regions must have different colors, e.g.,  $NSW \neq V$

# Australian Capital Territory (ACT) and Canberra (inside NSW)



View of the Australian National University and Telstra Tower

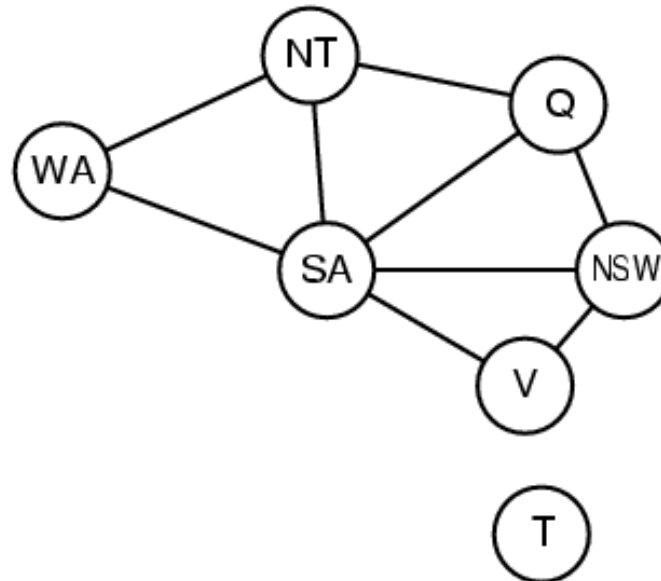
# One Solution



## ■ Solution assignment:

- $\{ WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green \}$
- Perhaps in addition  $ACT = blue$

# Constraint Graph



- Works for **binary** CSPs (otherwise hyper-graph)
- **Nodes** = variables, **arcs** = constraints
- Graph structure can be important (e.g., connected components)

Note: Our problem is 3-colorability for a planar graph

# Variations

- Binary, ternary, or even higher **arity**
- **Finite** domains ( $d$  values)  $\Rightarrow d^n$  possible variable assignments
- **Infinite** domains (reals, integers)
  - *linear constraints*: solvable (in P if real)
  - *nonlinear constraints*: unsolvable



# Applications

- Timetabling (classes, rooms, times)
- Configuration (hardware, cars, ...)
- Spreadsheets
- Scheduling
- Floor planning
- Frequency assignments
- ...

# Backtracking Search over Assignments

- Assign values to variables **step by step** (order does not matter)
- Consider only one variable per search node!
- **DFS** with single-variable assignments is called **backtracking search**
- Can solve  $n$ -queens for  $n \approx 25$

# Algorithm

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING([], csp)

function RECURSIVE-BACKTRACKING(assigned, csp) returns solution/failure
  if assigned is complete then return assigned
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assigned, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assigned, csp) do
    if value is consistent with assigned according to CONSTRAINTS[csp] then
      result ← RECURSIVE-BACKTRACKING([var = value | assigned], csp)
      if result ≠ failure then return result
  end
  return failure
```

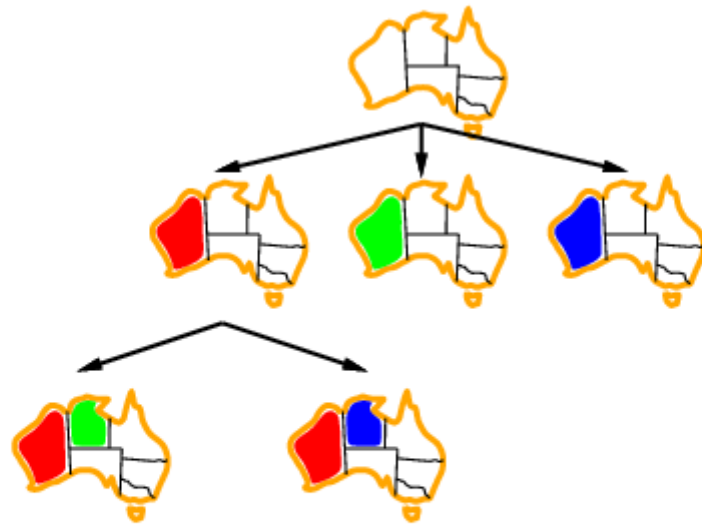
# Example (1)



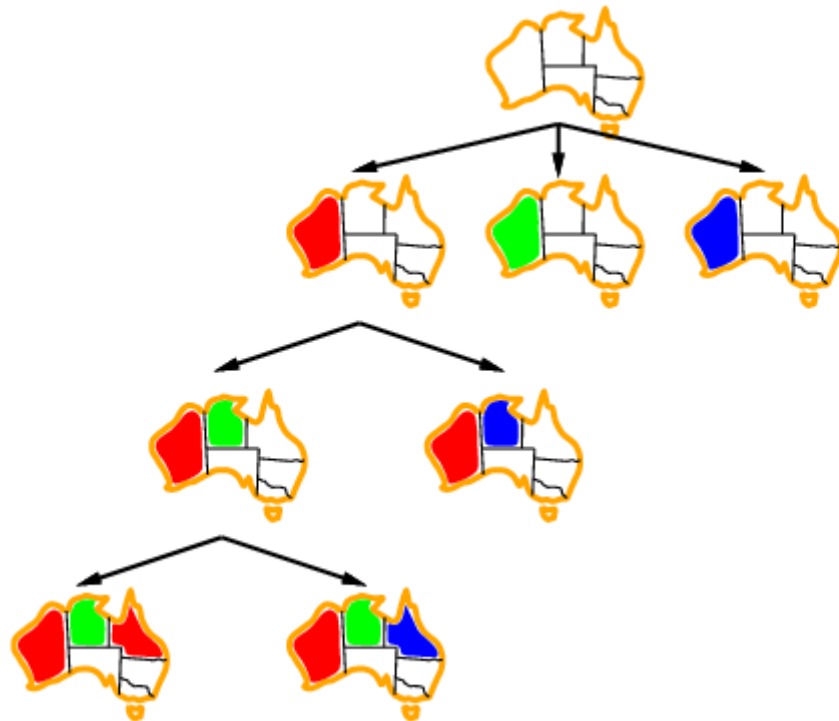
## Example (2)



## Example (3)



# Example (4)



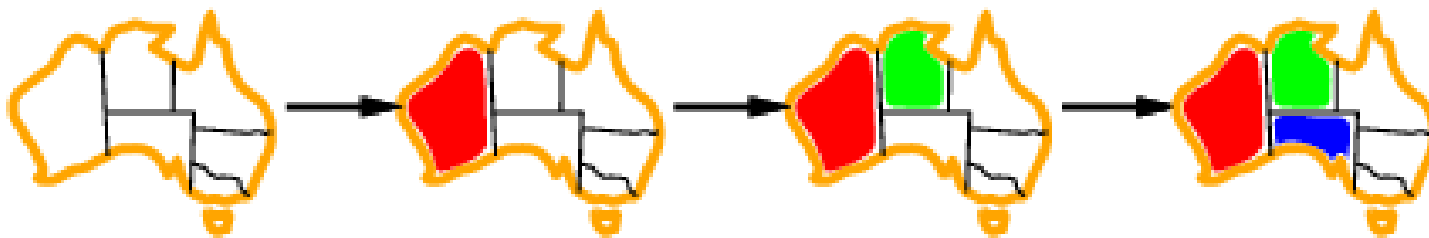
# Improving Efficiency: CSP Heuristics & Pruning Techniques

- Variable ordering: Which one to assign first?
  - Value ordering: Which value to try first?
  - Try to detect failures early on
  - Try to exploit problem structure
- Note: all this is not problem-specific!



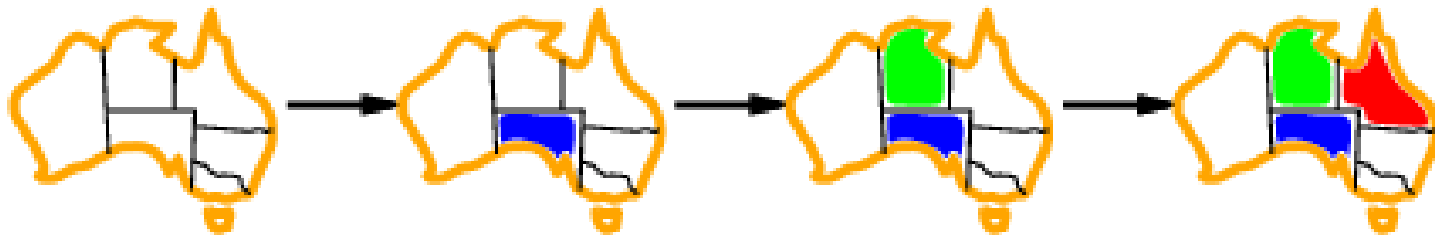
# Variable Ordering: Most constrained first

- Most constrained variable:
  - choose the variable with the **fewest remaining legal values**
  - reduces branching factor!



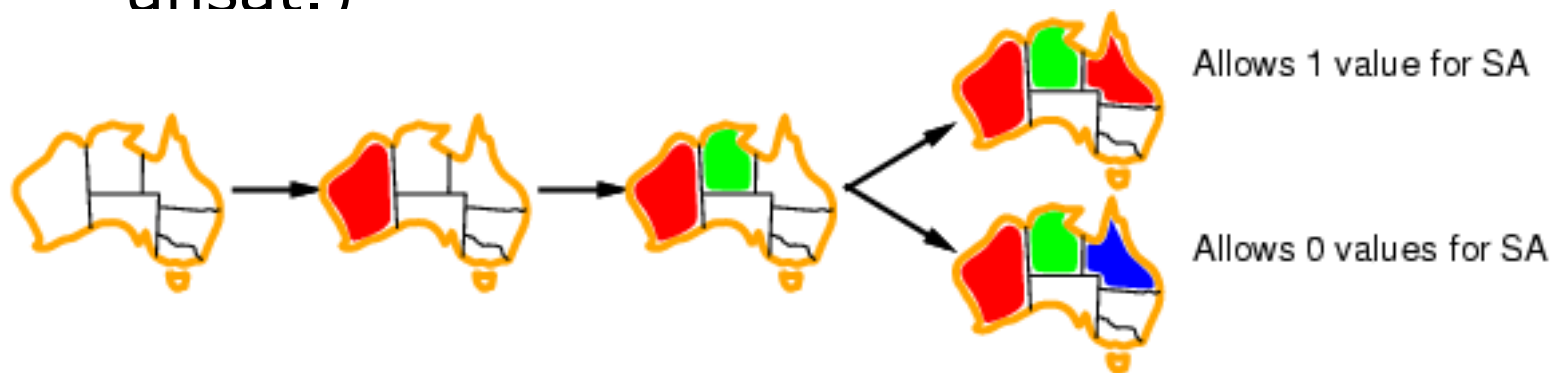
# Variable Ordering: Most Constraining Variable First

- Break ties among variables with the same number of remaining legal values:
  - choose variable with the **most constraints on remaining unassigned variables**
  - reduces branching factor in the next steps



# Value Ordering: Least Constraining Value First

- Given a variable,
  - choose first a value that rules out the **fewest values** in the remaining unassigned variables
  - We want to find an assignment that satisfies the constraints (of course, does not help if unsat.)

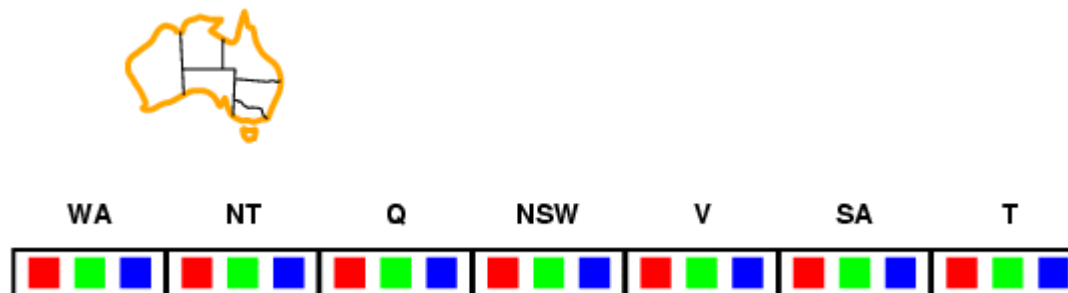


## Rule out Failures early on: Forward Checking

- Whenever a value is assigned to a variable, values that are now *illegal* for other variables are *removed*
- *Implements* what the ordering heuristics implicitly compute
- $WA = \textit{red}$ , then  $NT$  cannot become *red*
- If all values are removed for one variable, we can *stop!*

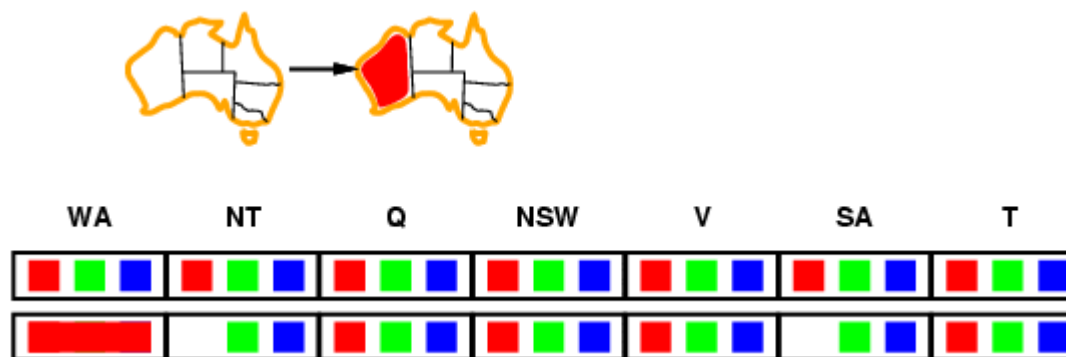
# Forward Checking (1)

- Keep track of remaining values
- Stop if all have been removed



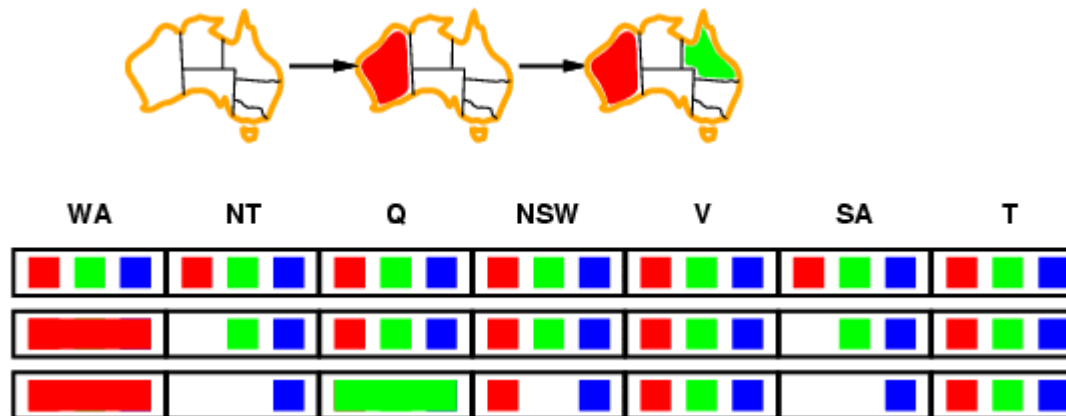
## Forward Checking (2)

- Keep track of remaining values
- Stop if all have been removed



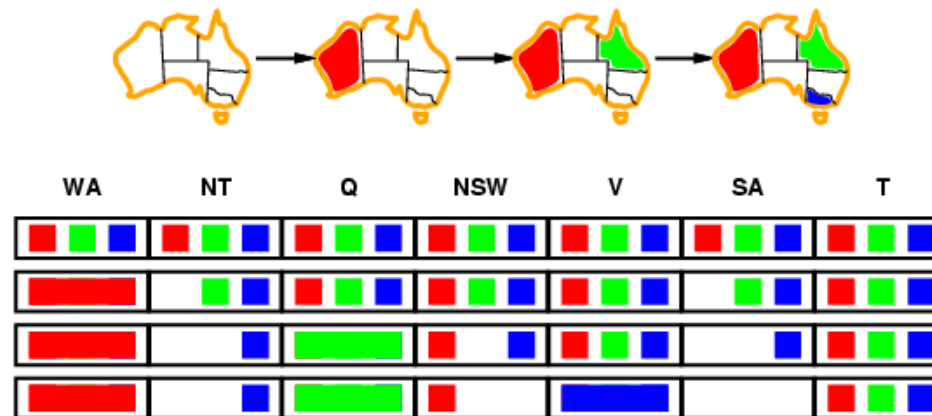
# Forward Checking (3)

- Keep track of remaining values
- Stop if all have been removed



# Forward Checking (4)

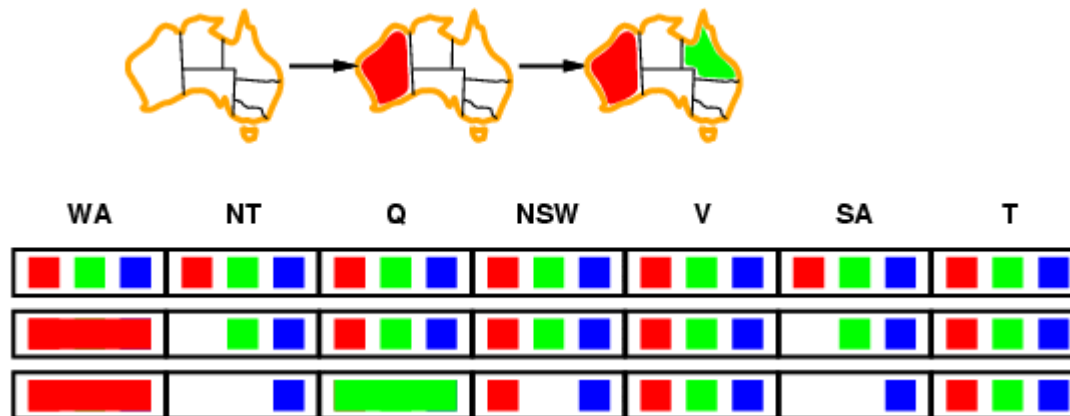
- Keep track of remaining values
- Stop if all have been removed





# Forward Checking: Sometimes it Misses Something

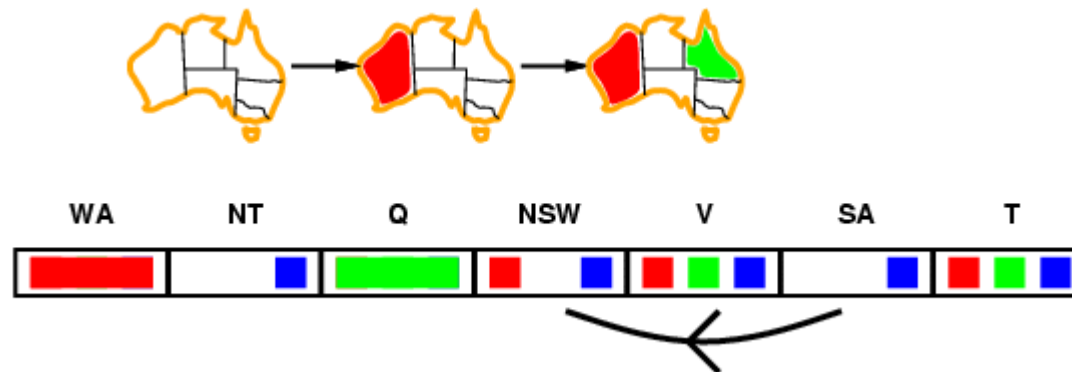
- Forward Checking propagates information from assigned to unassigned variables
- However, there is no propagation between unassigned variables



# Arc Consistency

- A directed arc  $X \rightarrow Y$  is “consistent” iff
  - for every value  $x$  of  $X$ , there exists a value  $y$  of  $Y$ , such that  $(x,y)$  satisfies the constraint between  $X$  and  $Y$
- Remove values from the domain of  $X$  to enforce arc-consistency
- Arc consistency detects failures earlier
- Can be used as preprocessing technique or as a propagation step during backtracking

# Arc Consistency Example



# AC3 Algorithm

```
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
     $(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\textit{queue})$ 
    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
```

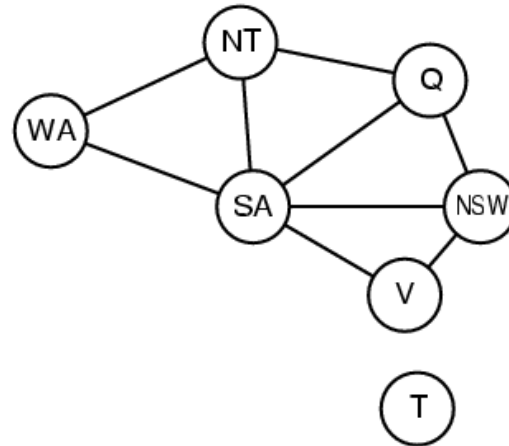
---

```
function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) returns true iff we remove
a value
  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 between  $X_i$ 
and  $X_j$ 
      then delete  $x$  from DOMAIN[ $X_i$ ]; removed  $\leftarrow$  true
  return removed
```

# Properties of AC3

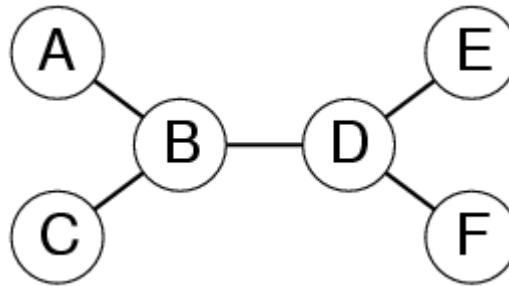
- AC3 runs in  $O(d^3n^2)$  time, with  $n$  being the number of nodes and  $d$  being the maximal number of elements in a domain
- Of course, AC3 does not detect all inconsistencies (which is an NP-hard problem)

# Problem Structure (1)



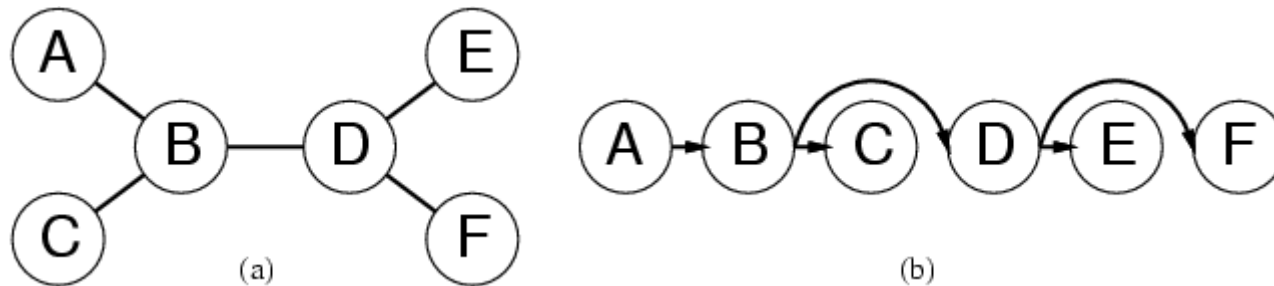
- CSP has two independent components
- Identifiable as connected components of constraint graph
- Can reduce the search space dramatically

## Problem Structure (2): Tree-structured CSPs



- If the CSP graph is a tree, then it can be solved in  $O(nd^2)$ 
  - General CSPs need in the worst case  $O(d^n)$
- *Idea*: Pick root, order nodes, apply arc consistency from leaves to root, and assign values starting at root

## Problem Structure (2): Tree-structured CSPs

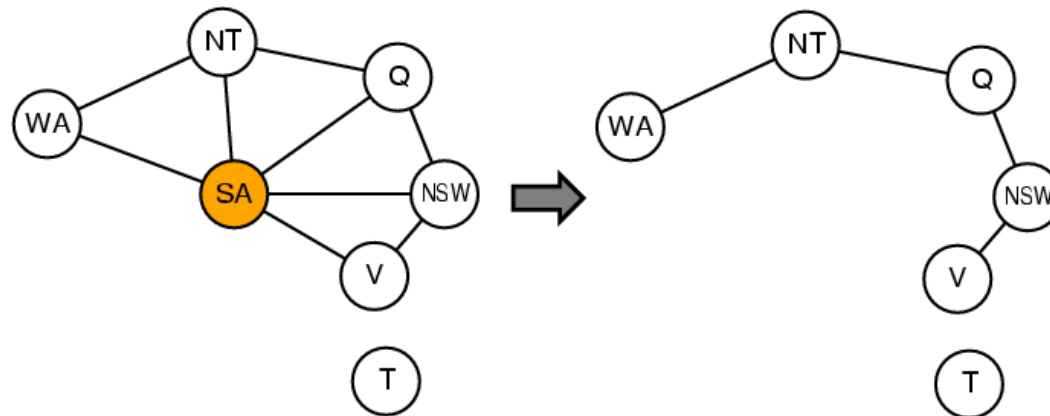


- Apply **arc-consistency** to  $(X_i, X_k)$ , when  $X_i$  is the parent of  $X_k$ , for all  $k=n$  *downto* 2.
- Now one can start at  $X_1$  **assigning values** from the remaining domains without creating any conflict in one sweep through the tree!
- Algorithm **linear** in  $n$



## Problem Structure (3): Almost Tree-structured

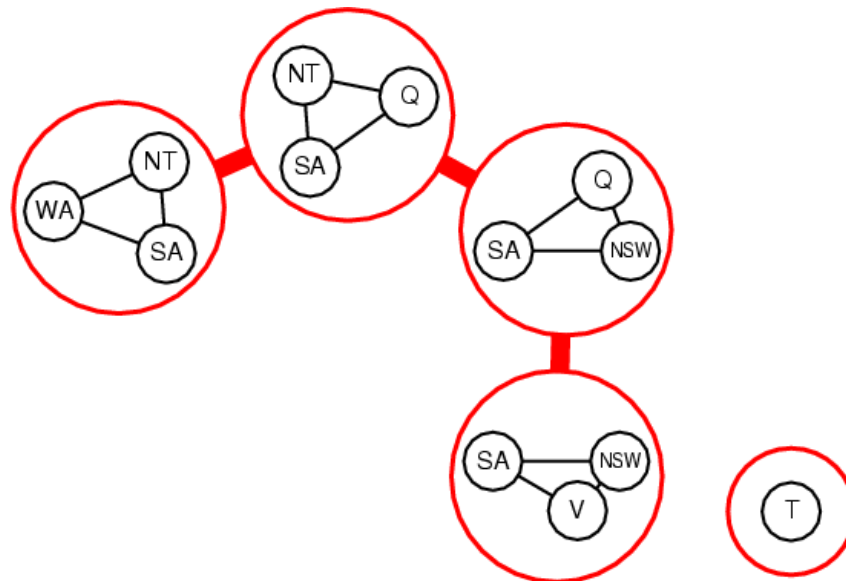
- **Conditioning:** Instantiate a variable and prune values in neighboring variables



- **Cutset conditioning:** Instantiate (in all ways) a set of variables in order to reduce the graph to a tree (note: finding minimal cutset is NP-hard)

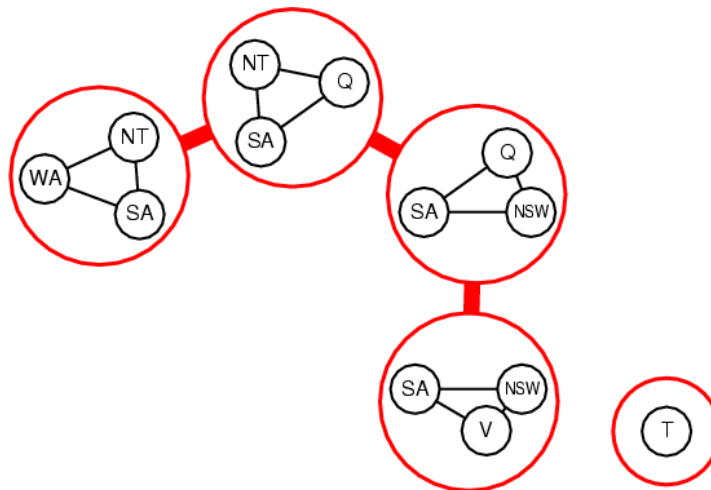
## Another Method: Tree Decomposition (1)

- Decompose problem into a set of connected **sub-problems**, where two sub-problems are connected when they share a constraint
- Solve sub-problems independently and combine solutions



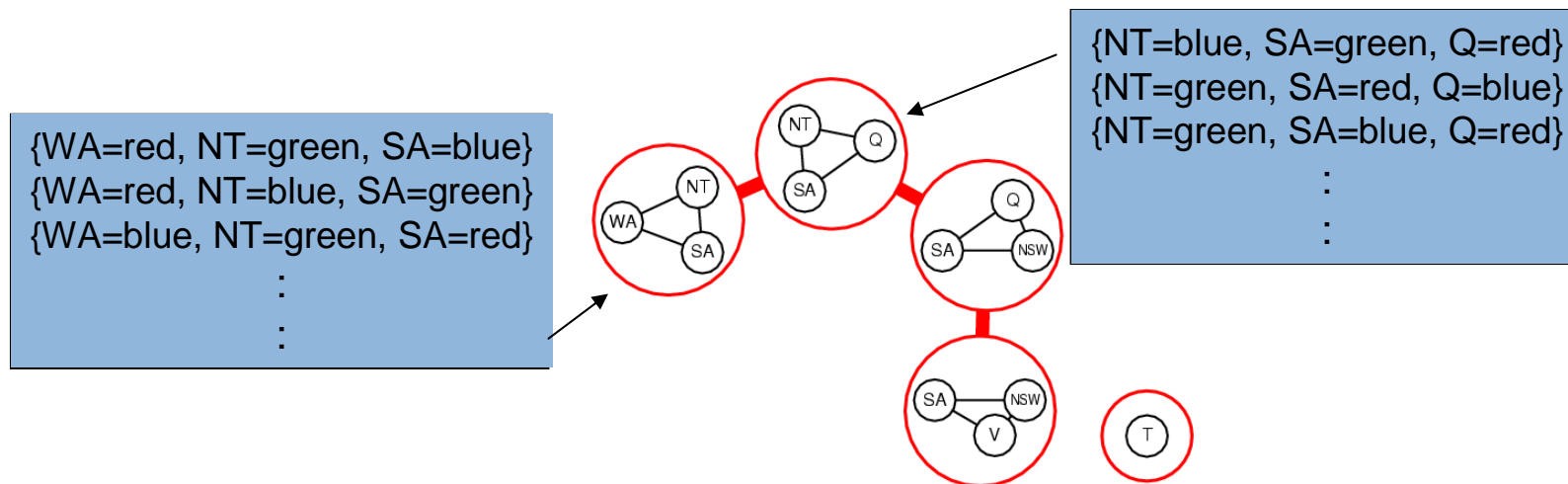
# Another Method: Tree Decomposition (2)

- A **tree decomposition** must satisfy the following conditions:
  - **Every variable** of the original problem appears in at least one sub-problem
  - **Every constraint** appears in at least one sub-problem
  - If a variable appears in two sub-problems, it must appear in **all sub-problems on the path** between the two sub-problems
  - The connections form a **tree**



## Another Method: Tree Decomposition (3)

- Consider sub-problems as new **mega-nodes**, which have values defined by the solutions to the sub-problems
- Use technique for **tree-structured CSP** to find an overall solution (constraint is to have identical values for the same variable).



# Tree Width

- Tree width of a tree decomposition = size of largest sub-problem minus 1
- Tree width of a graph is minimal tree width over all possible tree decompositions
- If a graph has tree width  $w$  and we know a tree decomposition with that width, we can solve the problem in  $O(nd^{w+1})$
- Finding a tree decomposition with minimal tree width is NP-hard

# Summary & Outlook

- CSPs are a special kind of search problem:
  - states are value assignments
  - goal test is defined by constraints
- Backtracking = DFS with one variable assigned per node. Other intelligent backtracking techniques possible
- Variable/value ordering heuristics can help dramatically
- Constraint propagation prunes the search space
- Path-consistency is a constraint propagation technique for triples of variables
- Tree structure of CSP graph simplifies problem significantly
- Cutset conditioning and tree decomposition are two ways to transform part of the problem into a tree
- CSPs can also be solved using local search