



# Lecture 6: Search 5

[http://cs.nju.edu.cn/yuy/course\\_ai15.ashx](http://cs.nju.edu.cn/yuy/course_ai15.ashx)



# Previously...



## Path-based search

- Uninformed search
- Informed search

## Iterative-improvement search

- Hill climbing: greedy method
- Meta-heuristic algorithms

## Adversarial search

- Minimax Search
- Alpha-Beta Pruning

# Constraint satisfaction problems (CSPs)



Standard search problem:

**state** is a “black box” —any old data structure that supports goal test, eval, successor

CSP:

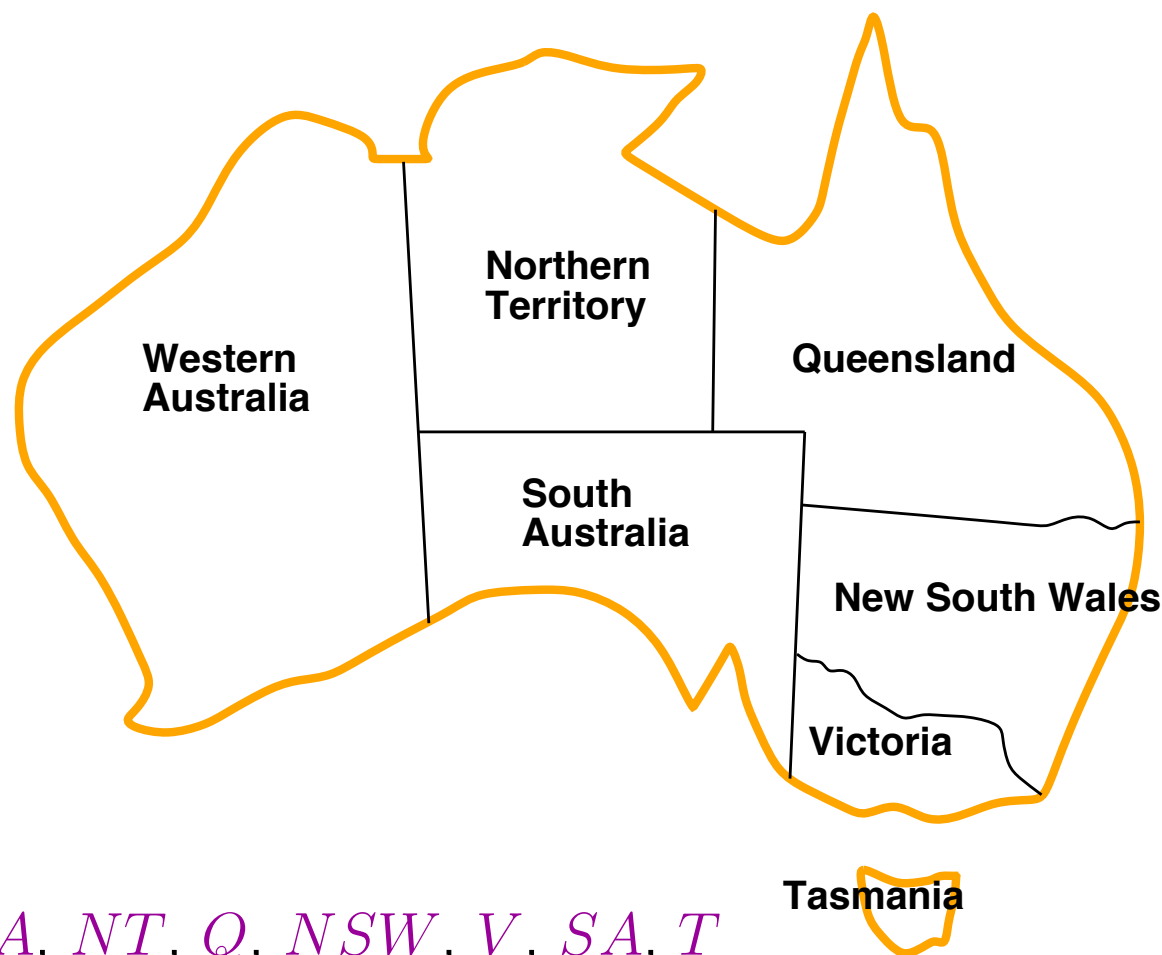
**state** is defined by **variables**  $X_i$  with **values** from domain  $D_i$

**goal test** is a set of **constraints** specifying allowable combinations of values for subsets of variables

Simple example of a **formal representation language**

Allows useful **general-purpose** algorithms with more power than standard search algorithms

# Example: Map-Coloring



Variables  $WA, NT, Q, NSW, V, SA, T$

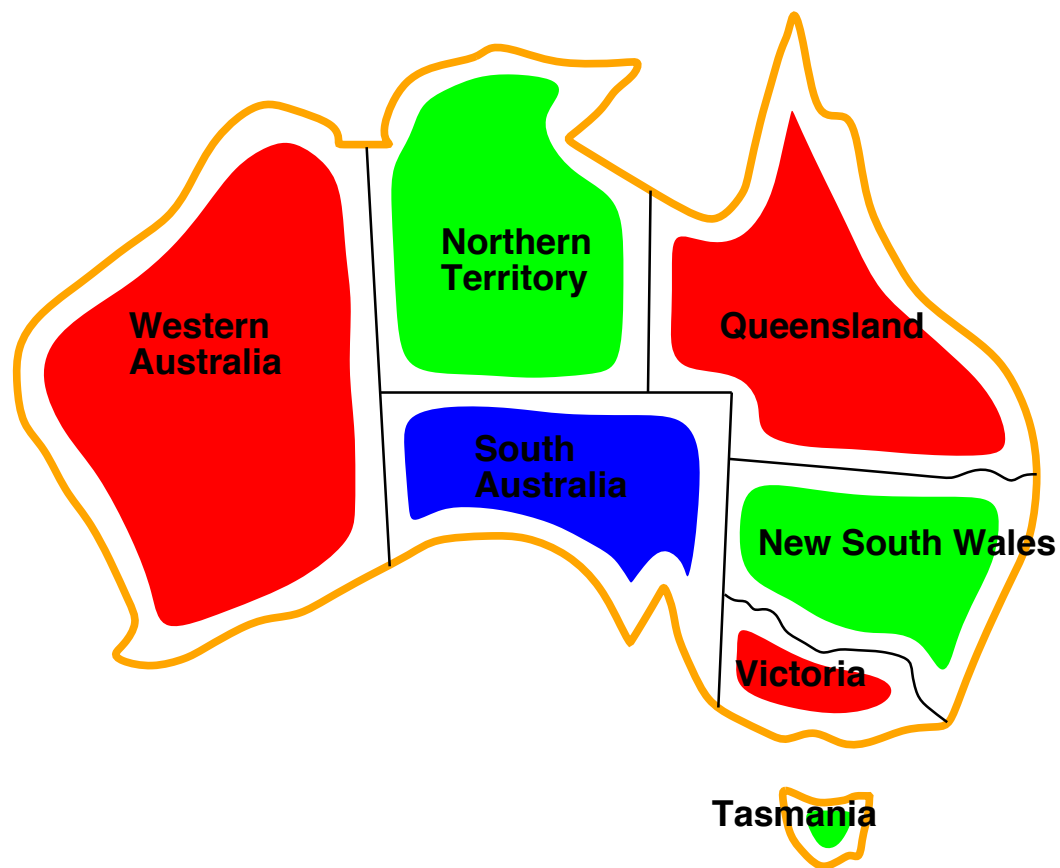
Domains  $D_i = \{red, green, blue\}$

Constraints: adjacent regions must have different colors

e.g.,  $WA \neq NT$  (if the language allows this), or

$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \dots\}$

# Example: Map-Coloring



Solutions are assignments satisfying all constraints, e.g.,

$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

# Varieties of CSPs



## Discrete variables

finite domains; size  $d \Rightarrow O(d^n)$  complete assignments

- ◇ e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)

infinite domains (integers, strings, etc.)

- ◇ e.g., job scheduling, variables are start/end days for each job
- ◇ need a **constraint language**, e.g.,  $StartJob_1 + 5 \leq StartJob_3$
- ◇ **linear** constraints solvable, **nonlinear** undecidable

## Continuous variables

- ◇ e.g., start/end times for Hubble Telescope observations
- ◇ linear constraints solvable in poly time by LP methods

# Varieties of CSPs



**Unary** constraints involve a single variable,

e.g.,  $SA \neq green$

**Binary** constraints involve pairs of variables,

e.g.,  $SA \neq WA$

**Higher-order** constraints involve 3 or more variables,

e.g., cryptarithmic column constraints

**Preferences** (soft constraints), e.g.,  $red$  is better than  $green$

often representable by a cost for each variable assignment

→ constrained optimization problems

# Real-world CSPs



Assignment problems

e.g., who teaches what class

Timetabling problems

e.g., which class is offered when and where?

Hardware configuration

Spreadsheets

Transportation scheduling

Factory scheduling

Floorplanning

Notice that many real-world problems involve real-valued variables

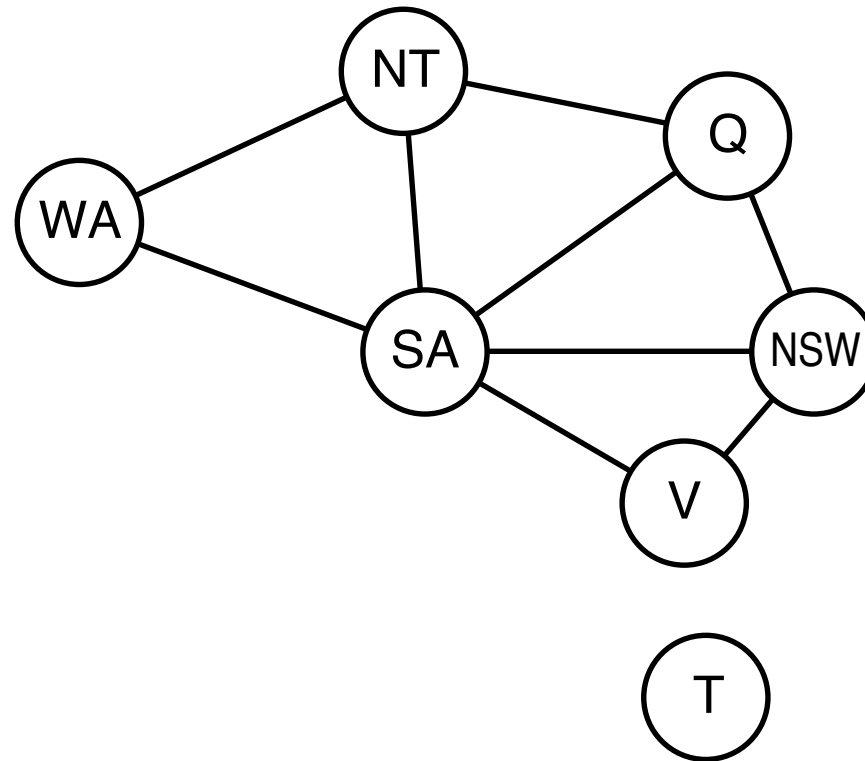


# Constraint graph

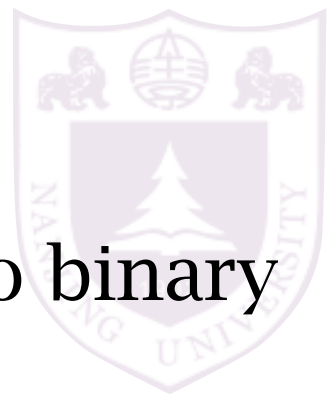


Binary CSP: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs show constraints



General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!



# Convert higher-order to binary

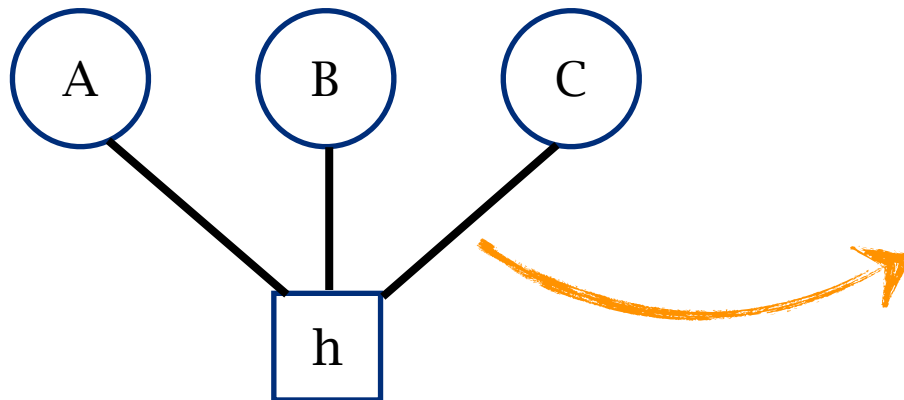
A higher-order constraint can be converted to binary constraints with a *hidden-variable*

variable: A, B, C    domain: {1,2,3}    constraint:  $A+B=C$

all possible assignments:  $(A,B,C) = (1,1,2), (1,2,3), (2,1,3)$

*hidden-variable*: h with domain: {1,2,3}  
(each value corresponds to an assignment)

the constraint graph:



constraint:

$h=1, C=2$

$h=2, C=3$

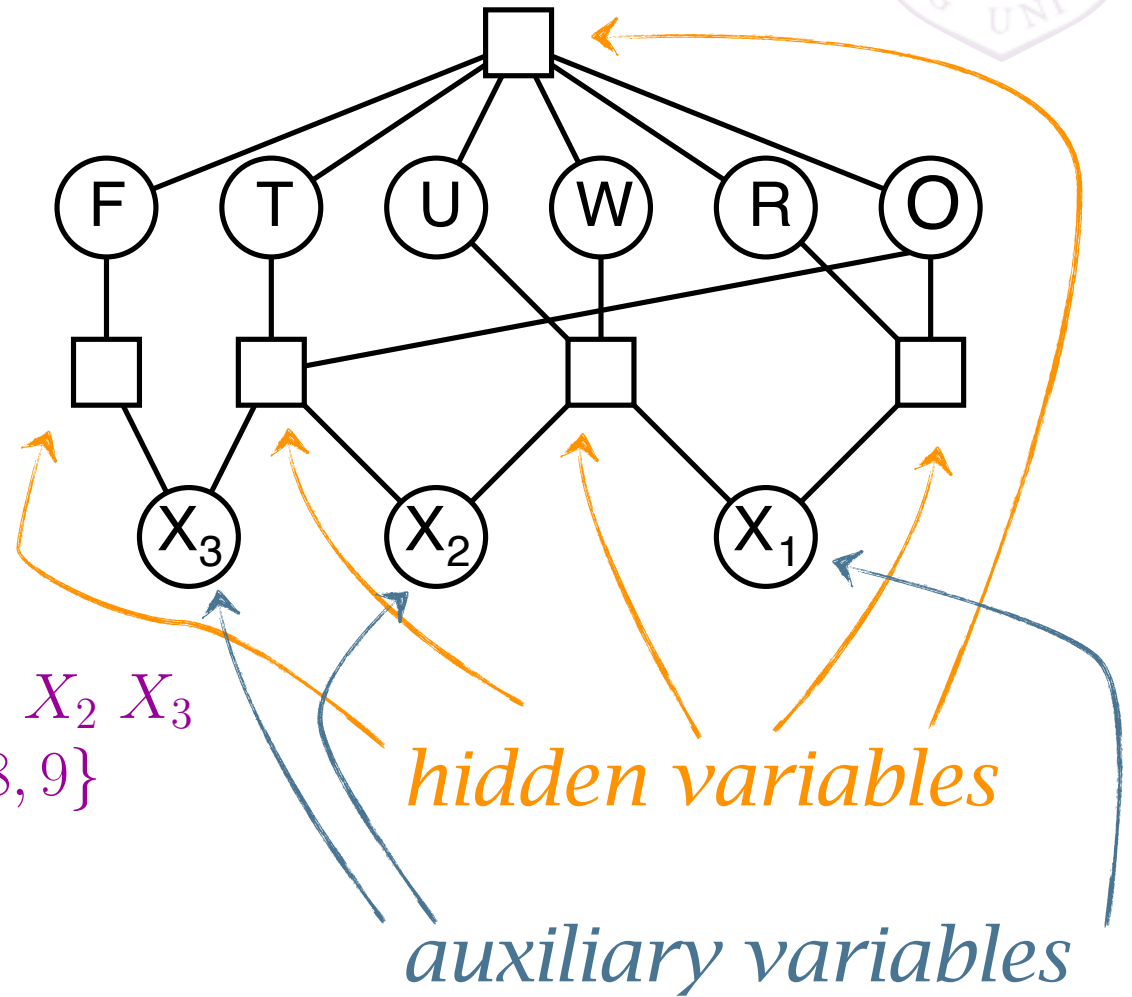
$h=3, C=3$

from the definition of h

# Example: Cryptarithmic



$$\begin{array}{r} \text{TWO} \\ + \text{TWO} \\ \hline \text{FOUR} \end{array}$$



Variables:  $F T U W R O X_1 X_2 X_3$

Domains:  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Constraints

$alldiff(F, T, U, W, R, O)$

$O + O = R + 10 \cdot X_1$ , etc.



# Standard search formulation (incremental)

Let's start with the straightforward, dumb approach, then fix it

States are defined by the values assigned so far

- ◇ **Initial state:** the empty assignment,  $\{\}$
- ◇ **Successor function:** assign a value to an unassigned variable that does not conflict with current assignment.  
⇒ fail if no legal assignments (not fixable!)
- ◇ **Goal test:** the current assignment is complete

- 1) This is the same for all CSPs! 😊
- 2) Every solution appears at depth  $n$  with  $n$  variables  
⇒ use depth-first search
- 3) Path is irrelevant, so can also use complete-state formulation
- 4)  $b = (n - \ell)d$  at depth  $\ell$ , hence  $n!d^n$  leaves!!!! 😞

# Backtracking search



Variable assignments are **commutative**, i.e.,

$[WA = red \text{ then } NT = green]$  same as  $[NT = green \text{ then } WA = red]$

Only need to consider assignments to a single variable at each node

$\Rightarrow b = d$  and there are  $d^n$  leaves

Depth-first search for CSPs with single-variable assignments is called **backtracking** search

Backtracking search is the basic uninformed algorithm for CSPs

Can solve  $n$ -queens for  $n \approx 25$

# Backtracking search



```
function BACKTRACKING-SEARCH(csp) returns solution/failure
```

```
  return RECURSIVE-BACKTRACKING({ }, csp)
```

```
function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
```

```
if assignment is complete then return assignment
```

```
var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
```

```
for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
```

```
  if value is consistent with assignment given CONSTRAINTS[csp] then
```

```
    add {var = value} to assignment
```

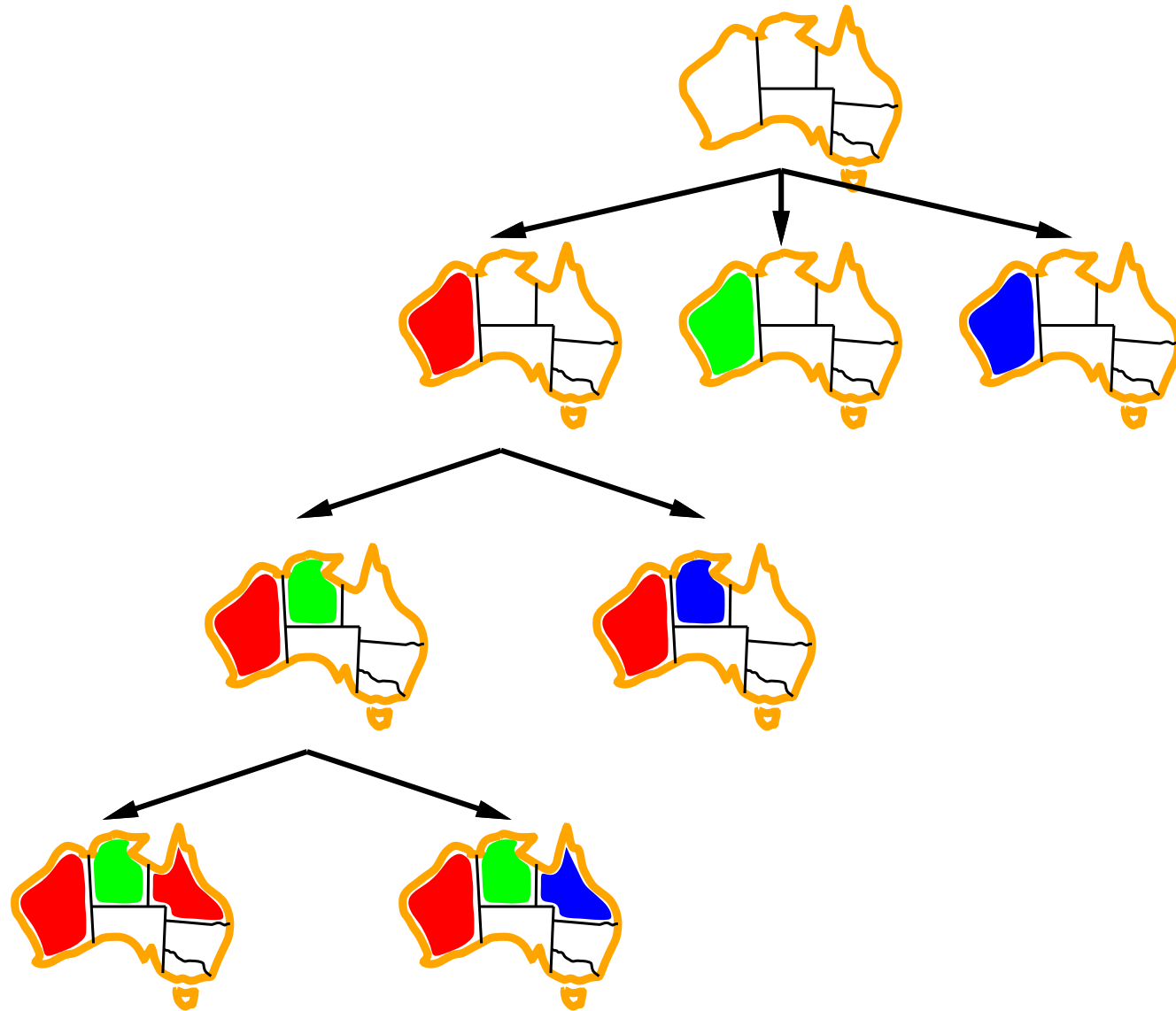
```
    result ← RECURSIVE-BACKTRACKING(assignment, csp)
```

```
    if result ≠ failure then return result
```

```
    remove {var = value} from assignment
```

```
return failure
```

# Backtracking search example



# Improving backtracking efficiency



backtracking is uninformed  
make it more informed

**General-purpose** methods can give huge gains in speed:

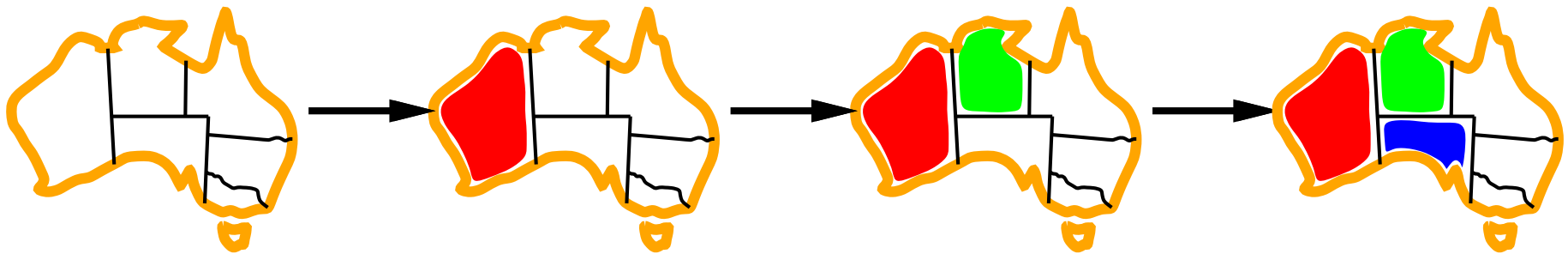
1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we take advantage of problem structure?



# Minimum remaining values



Minimum remaining values (MRV):  
choose the variable with the fewest legal values



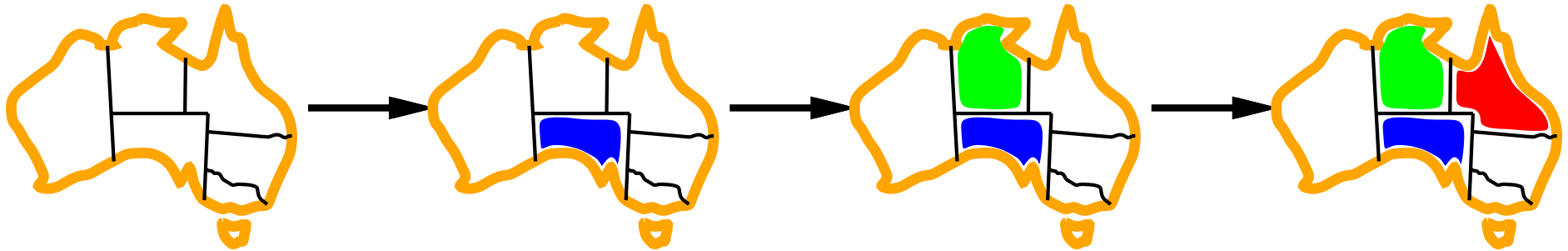
# Degree heuristic



Tie-breaker among MRV variables

Degree heuristic:

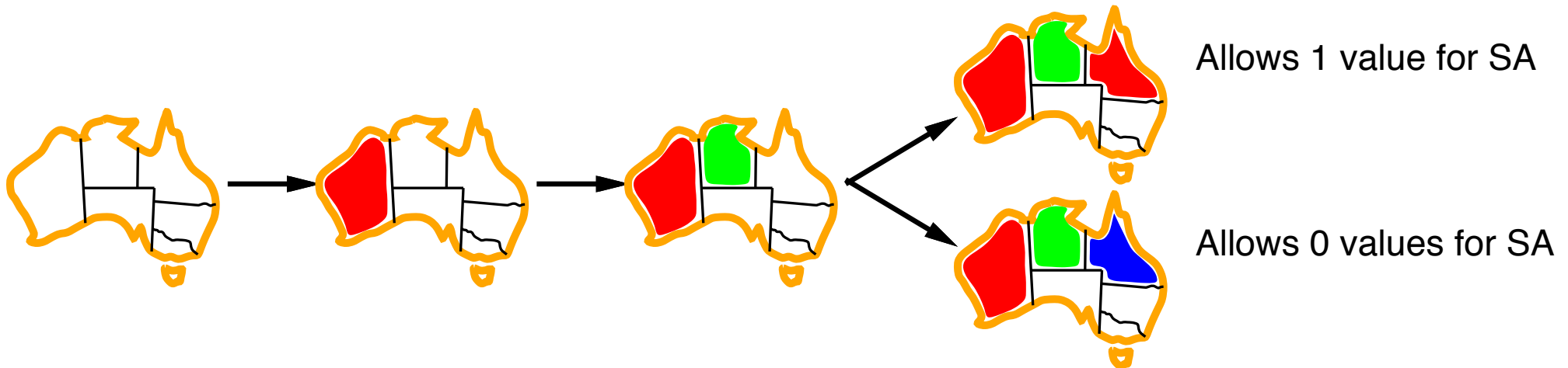
choose the variable with the most constraints on remaining variables



# Least constraining value

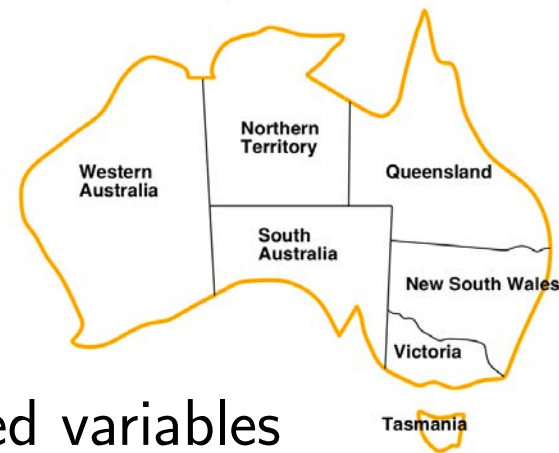


Given a variable, choose the least constraining value:  
the one that rules out the fewest values in the remaining variables

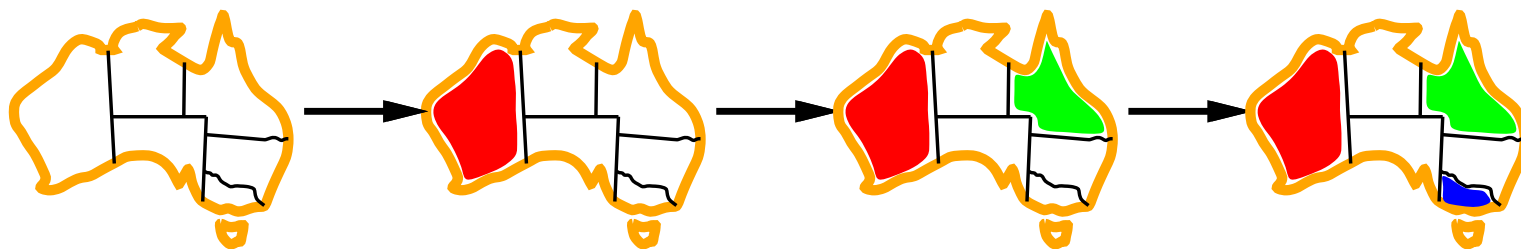


Combining these heuristics makes 1000 queens feasible

# Forward checking



Idea: Keep track of remaining legal values for unassigned variables  
 Terminate search when any variable has no legal values

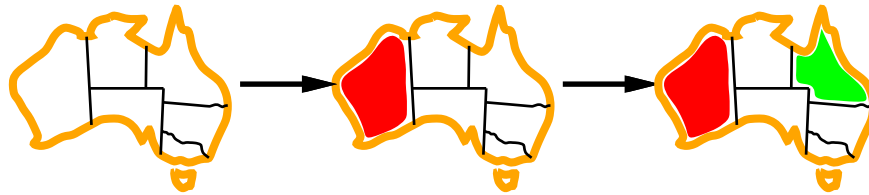


WA	NT	Q	NSW	V	SA	T
Red, Green, Blue	Red, Green, Blue	Red, Green, Blue	Red, Green, Blue	Red, Green, Blue	Red, Green, Blue	Red, Green, Blue
Red	Green, Blue	Red, Green, Blue	Red, Green, Blue	Red, Green, Blue	Green, Blue	Red, Green, Blue
Red	Blue	Green	Red, Blue	Red, Green, Blue	Blue	Red, Green, Blue
Red	Blue	Green	Red	Blue		Red, Green, Blue

# Constraint propagation



Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



WA	NT	Q	NSW	V	SA	T
Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Red Green Blue
Red	Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Green Blue	Red Green Blue
Red	Blue	Green	Red Blue	Red Green Blue	Blue	Red Green Blue

*NT* and *SA* cannot both be blue!

Constraint propagation repeatedly enforces constraints locally

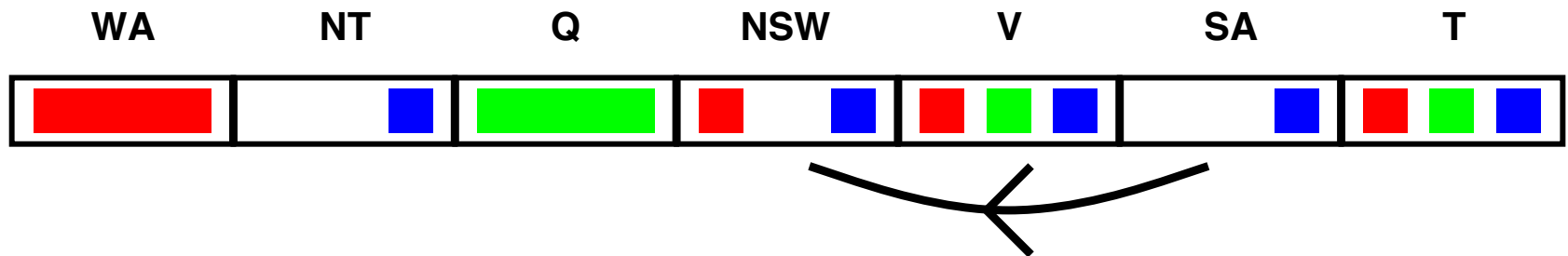
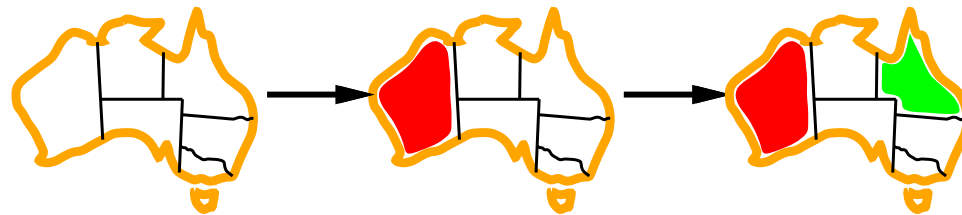
# Arc consistency



Simplest form of propagation makes each arc consistent

$X \rightarrow Y$  is consistent iff

for **every** value  $x$  of  $X$  there is **some** allowed  $y$

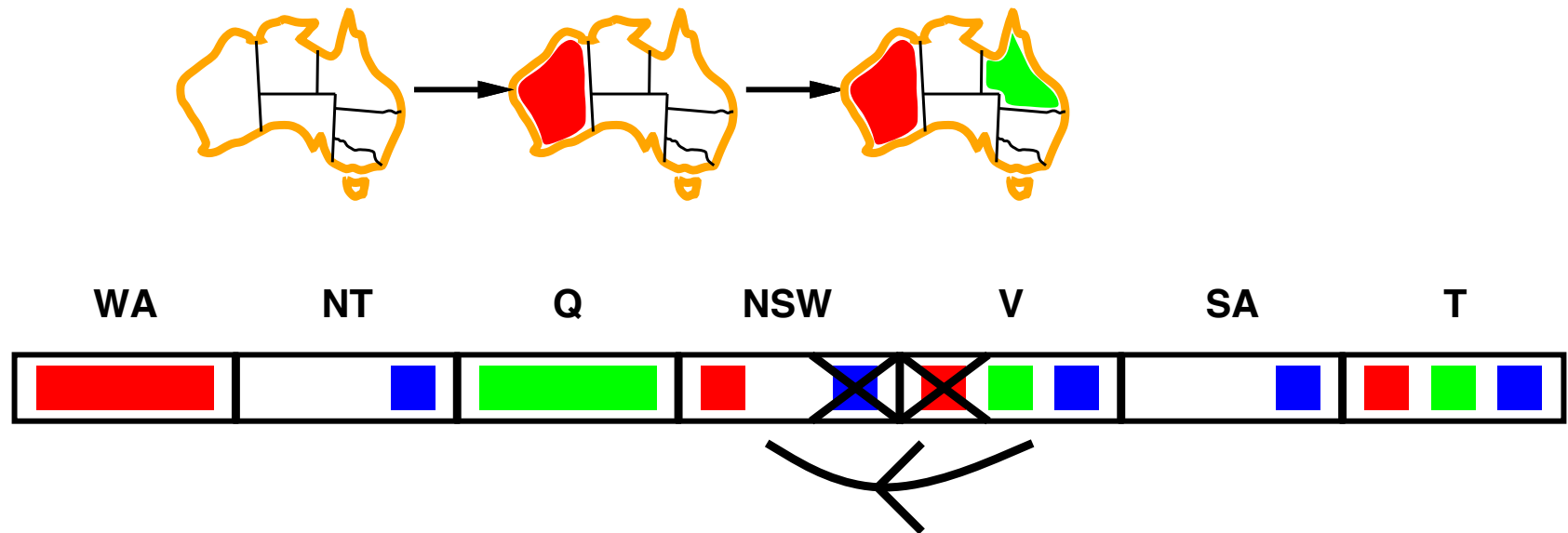


# Arc consistency



Simplest form of propagation makes each arc consistent

$X \rightarrow Y$  is consistent iff  
for **every** value  $x$  of  $X$  there is **some** allowed  $y$



If  $X$  loses a value, neighbors of  $X$  need to be rechecked

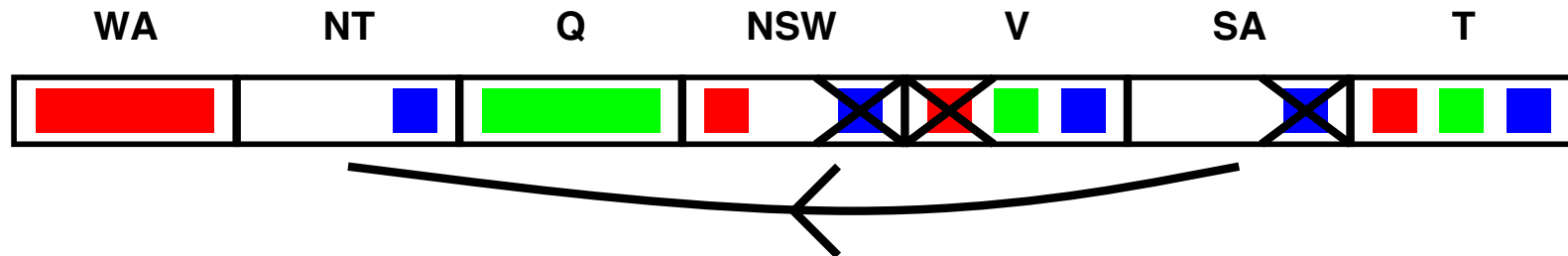
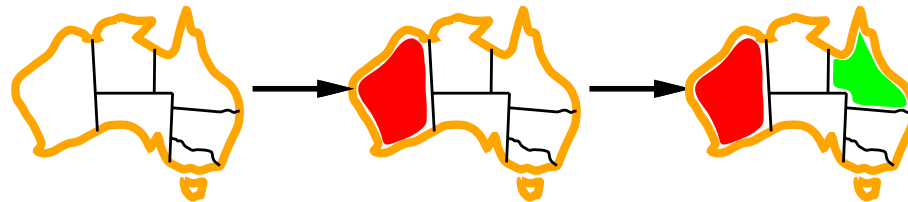
# Arc consistency



Simplest form of propagation makes each arc **consistent**

$X \rightarrow Y$  is consistent iff

for **every** value  $x$  of  $X$  there is **some** allowed  $y$



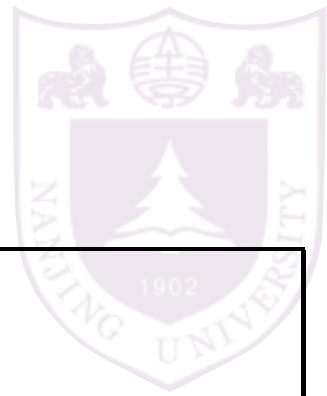
If  $X$  loses a value, neighbors of  $X$  need to be rechecked

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment



# Arc consistency



**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 succeeds

*removed*  $\leftarrow$  *false*

**for each**  $x$  **in** DOMAIN[ $X_i$ ] **do**

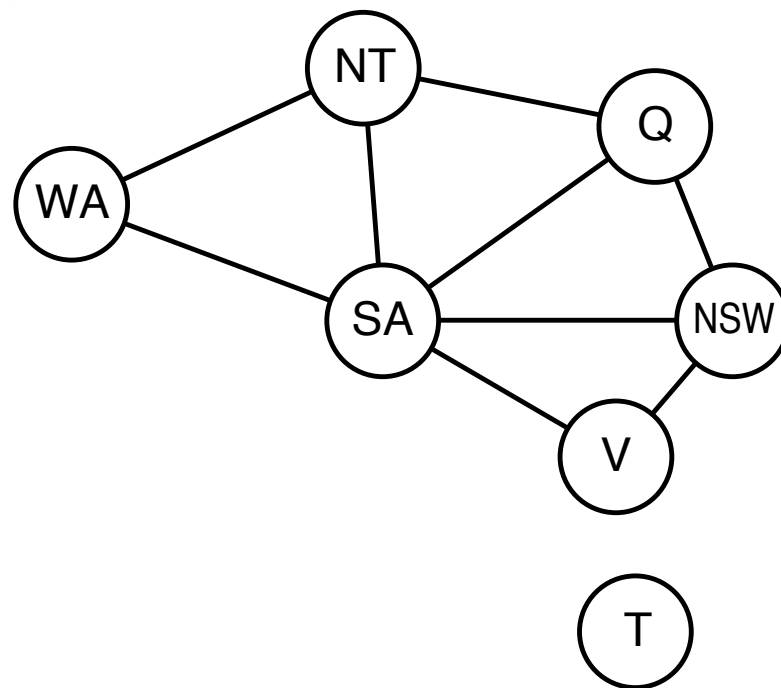
**if** no value  $y$  in DOMAIN[ $X_j$ ] allows  $(x, y)$  to satisfy the constraint  $X_i \leftrightarrow X_j$

**then** delete  $x$  from DOMAIN[ $X_i$ ]; *removed*  $\leftarrow$  *true*

**return** *removed*

$O(n^2d^3)$ , can be reduced to  $O(n^2d^2)$  (but detecting **all** is NP-hard)

# Problem Structure



Tasmania and mainland are **independent subproblems**

Identifiable as **connected components** of constraint graph

Suppose each subproblem has  $c$  variables out of  $n$  total

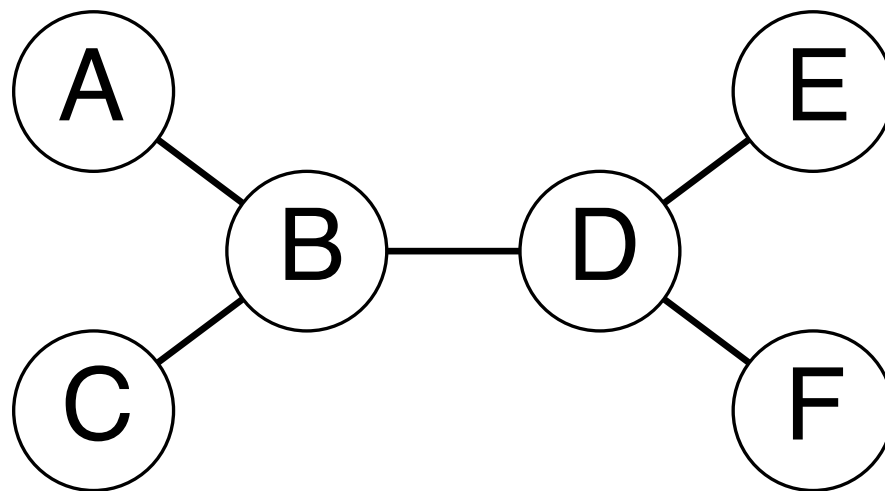
Worst-case solution cost is  $n/c \cdot d^c$ , **linear** in  $n$

E.g.,  $n = 80$ ,  $d = 2$ ,  $c = 20$

$2^{80} = 4$  billion years at 10 million nodes/sec

$4 \cdot 2^{20} = 0.4$  seconds at 10 million nodes/sec

# Tree-structured CSPs



**Theorem:** if the constraint graph has no loops, the CSP can be solved in  $O(n d^2)$  time

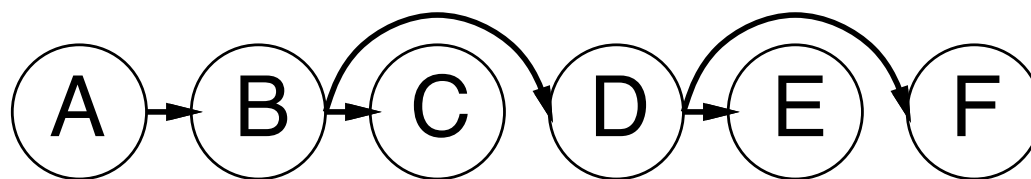
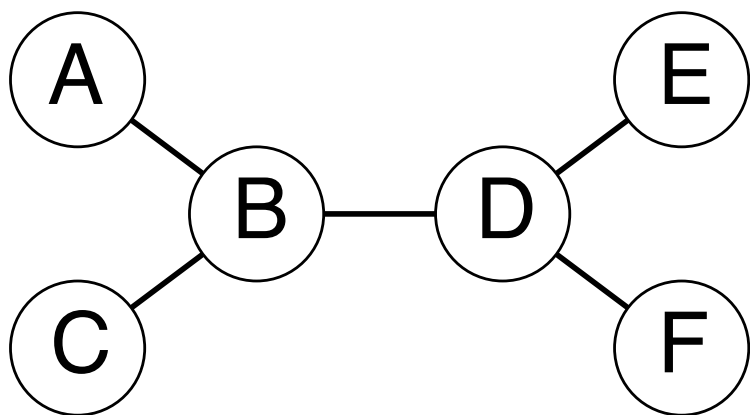
Compare to general CSPs, where worst-case time is  $O(d^n)$

This property also applies to logical and probabilistic reasoning: an important example of the relation between syntactic restrictions and the complexity of reasoning.

# Algorithm for tree-structured CSPs



1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering

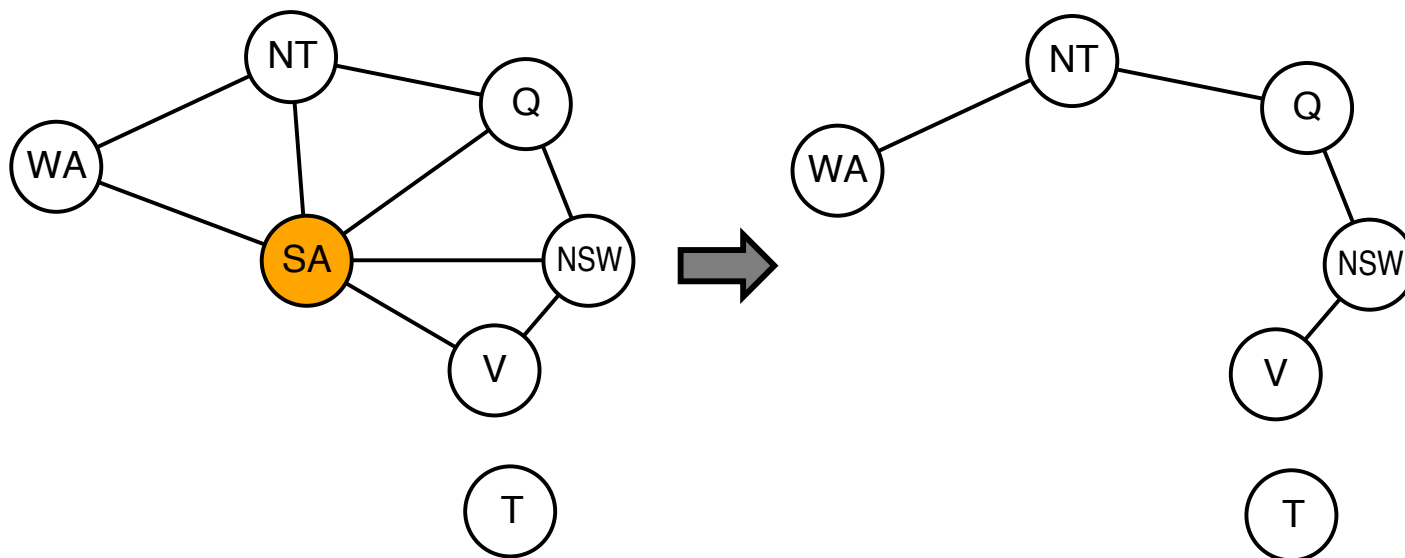


2. For  $j$  from  $n$  down to  $2$ , apply  $\text{REMOVEINCONSISTENT}(\text{Parent}(X_j), X_j)$
3. For  $j$  from  $1$  to  $n$ , assign  $X_j$  consistently with  $\text{Parent}(X_j)$

# Nearly tree-structured CSPs



Conditioning: instantiate a variable, prune its neighbors' domains



Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

Cutset size  $c \Rightarrow$  runtime  $O(d^c \cdot (n - c)d^2)$ , very fast for small  $c$

# Iterative algorithms for CSPs



Hill-climbing, simulated annealing typically work with “complete” states, i.e., all variables assigned

To apply to CSPs:

- allow states with unsatisfied constraints

- operators **reassign** variable values

Variable selection: randomly select any conflicted variable

Value selection by **min-conflicts** heuristic:

- choose value that violates the fewest constraints

- i.e., hillclimb with  $h(n) =$  total number of violated constraints

# Example: 4-Queens

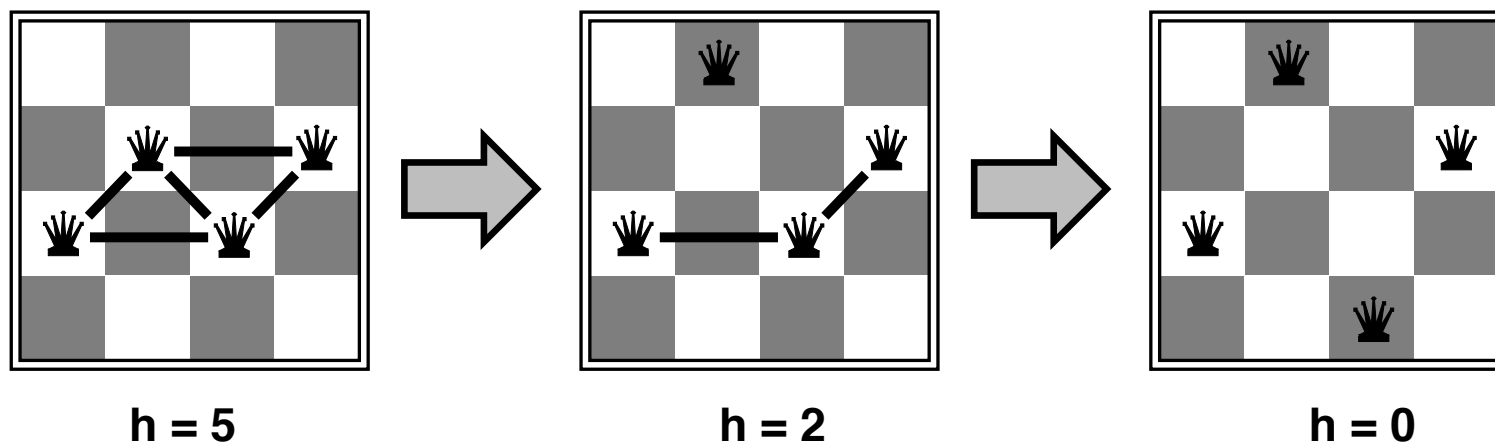


States: 4 queens in 4 columns ( $4^4 = 256$  states)

Operators: move queen in column

Goal test: no attacks

Evaluation:  $h(n) =$  number of attacks



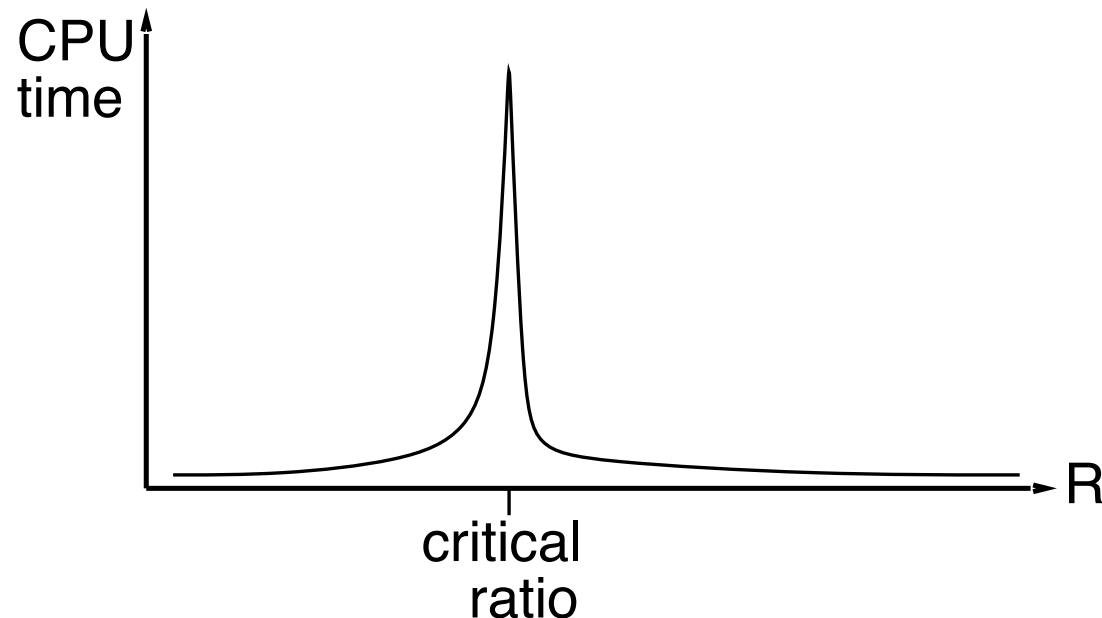


# Performance of min-conflicts

Given random initial state, can solve  $n$ -queens in almost constant time for arbitrary  $n$  with high probability (e.g.,  $n = 10,000,000$ )

The same appears to be true for any randomly-generated CSP **except** in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$





# Summary



CSPs are a special kind of problem:

- states defined by values of a fixed set of variables

- goal test defined by **constraints** on variable values

Backtracking = depth-first search with one variable assigned per node

Variable ordering and value selection heuristics help significantly

Forward checking prevents assignments that guarantee later failure

Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies

The CSP representation allows analysis of problem structure

Tree-structured CSPs can be solved in linear time

Iterative min-conflicts is usually effective in practice