PRINCIPLES OF INTELLIGENT SYSTEMS: IMPROVING BACKTRACKING SEARCH*

LECTURE 7

---

*These slides are taken from the Chapter 4b slides of Russell and Norvig's *Artificial Intelligence: A modern approach* (http://aima.eecs.berkeley.edu/slides-pdf/)

Outline

- Variable and value ordering
- Forward-checking
- Arc-consistency
- Problem structure and problem decomposition

Review: Backtracking search

```
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
```

Improving backtracking efficiency

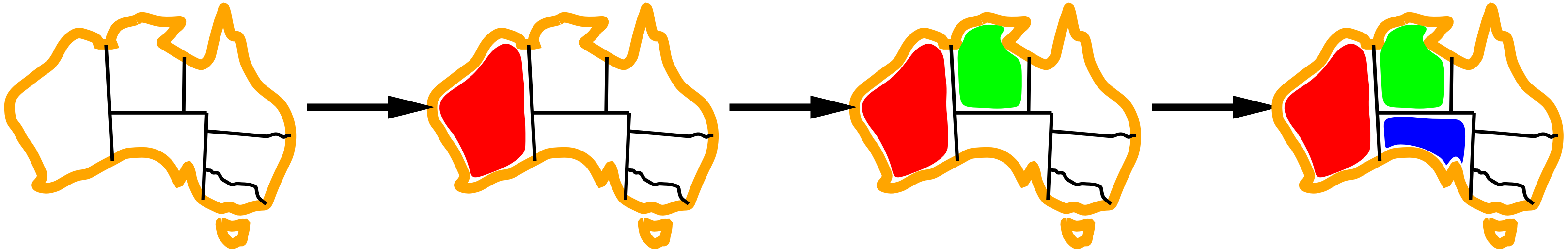
General-purpose methods can give huge gains in speed:

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?

Most constrained variable

Most constrained variable:

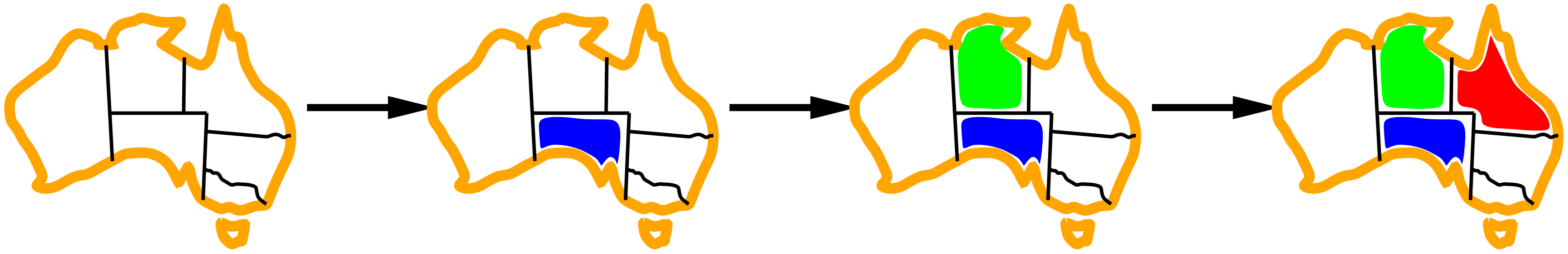
choose the variable with the fewest legal values



Most constraining variable

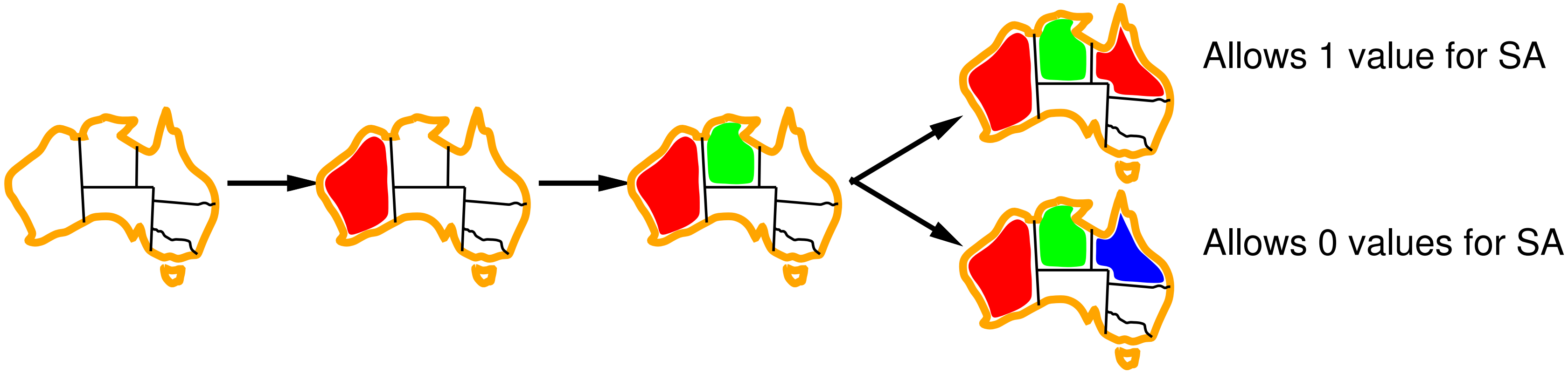
Tie-breaker among most constrained variables

Most constraining variable:
choose the variable with the most constraints on remaining variables



Least constraining value

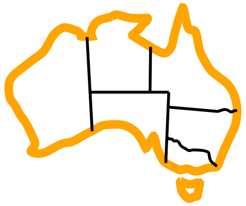
Given a variable, choose the least constraining value:
the one that rules out the fewest values in the remaining variables

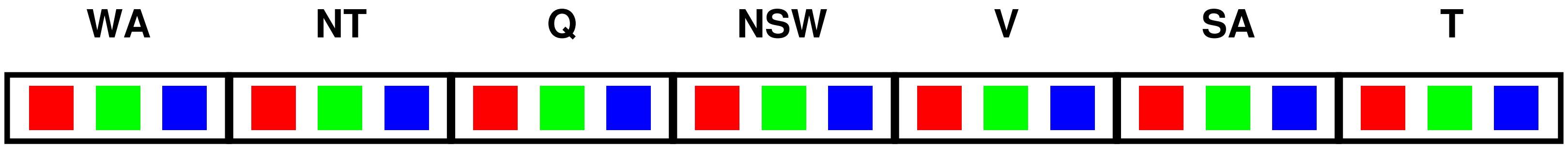


Combining these heuristics makes 1000 queens feasible

Forward checking

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

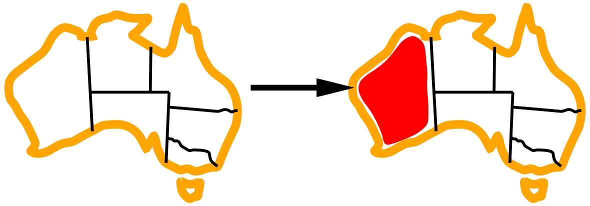




WA	NT	Q	NSW	V	SA	T
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦

Forward checking

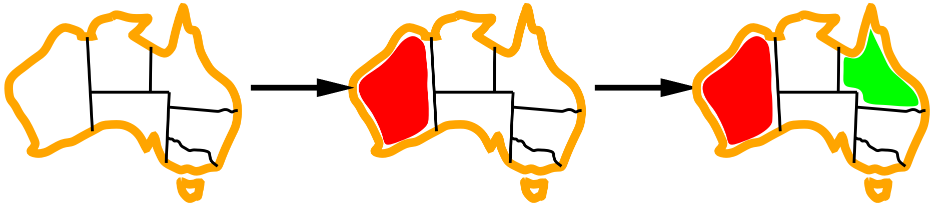
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

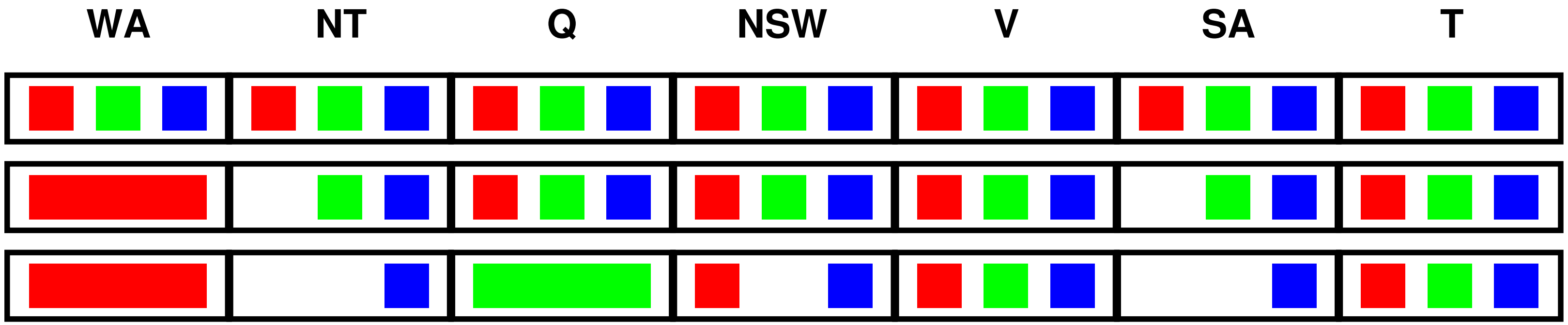


WA	NT	Q	NSW	V	SA	T
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦
🟥	🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟩 🟦	🟥 🟩 🟦

Forward checking

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

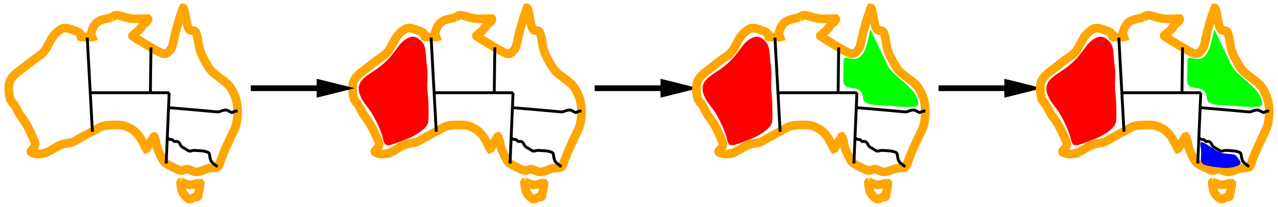




WA	NT	Q	NSW	V	SA	T
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦
🟥	🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟩 🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥 🟦	🟥 🟩 🟦	🟦	🟥 🟩 🟦

Forward checking

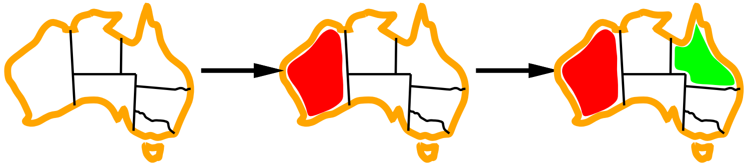
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦
🟥	🟦	🟩	🟥	🟦		🟥🟩🟦

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
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦
🟥	🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟩 🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥 🟦	🟥 🟩 🟦	🟦	🟥 🟩 🟦

NT and SA cannot both be blue!

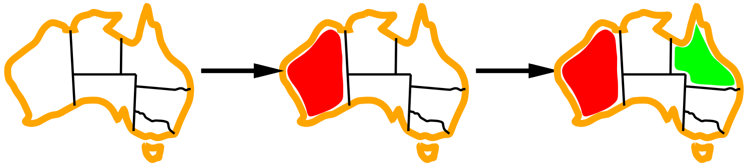
Constraint propagation repeatedly enforces constraints locally

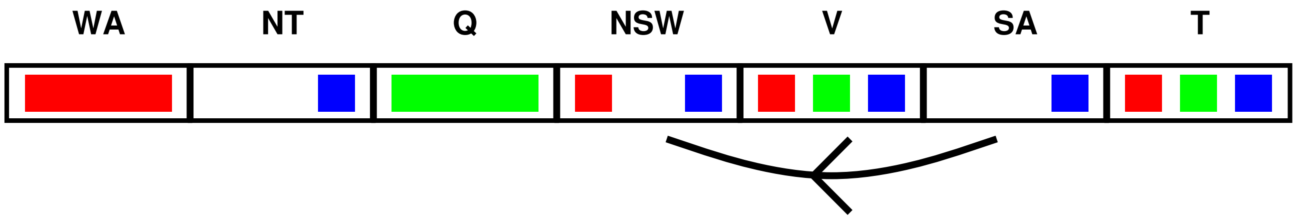
Arc consistency

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





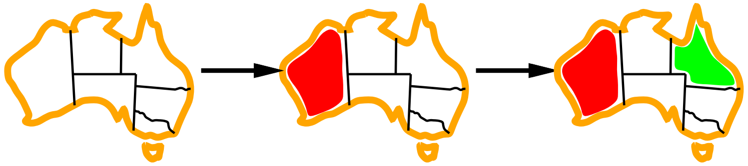
WA	NT	Q	NSW	V	SA	T
🟥	🟦	🟩	🟥 🟦	🟥 🟩 🟦	🟦	🟥 🟩 🟦

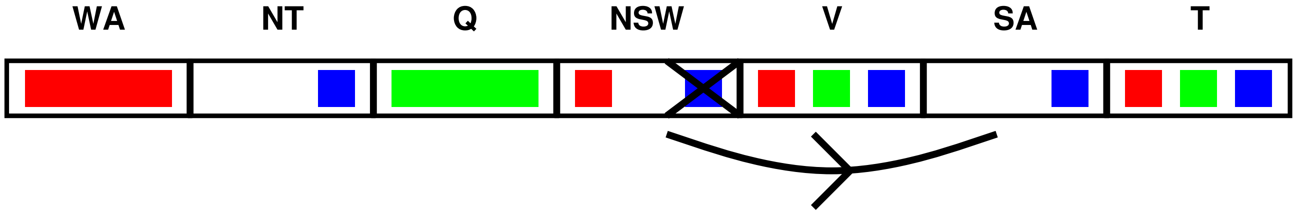
Arc consistency

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



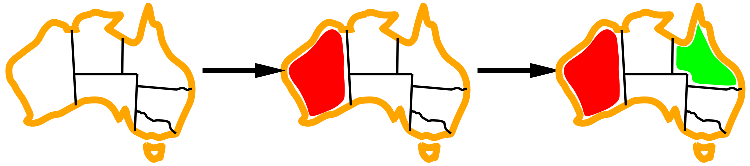


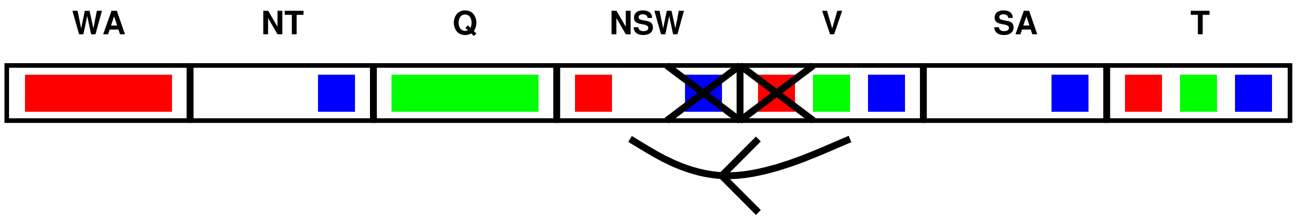
Arc consistency

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





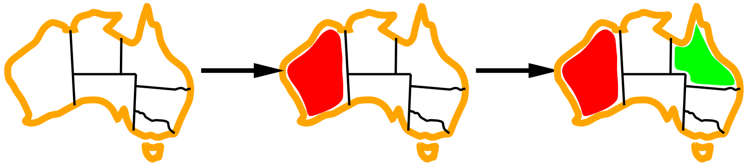
If X loses a value, neighbors of X need to be rechecked

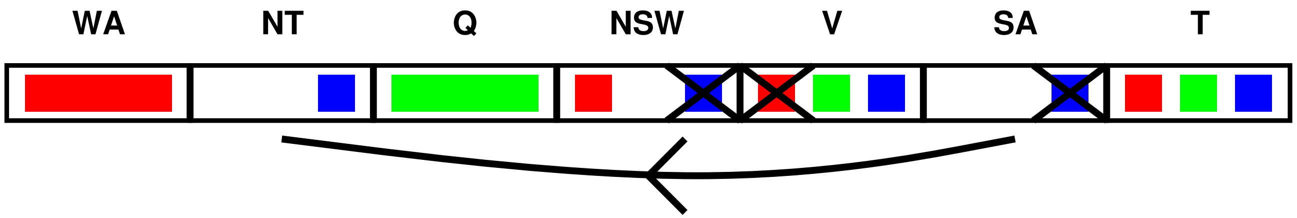
Arc consistency

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 algorithm

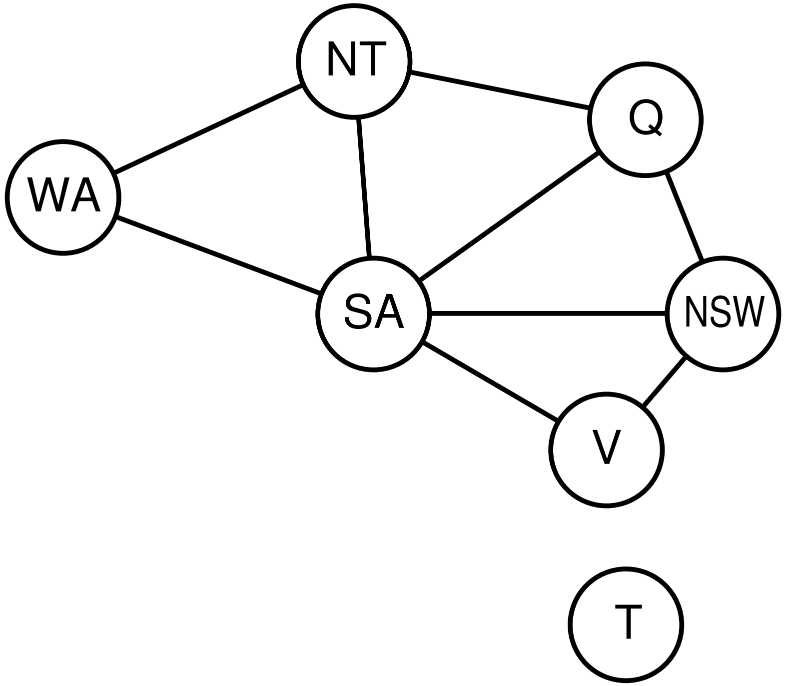
```
function AC-3( $csp$) returns the CSP, possibly with reduced domains
    inputs: $csp$, a binary CSP with variables $\{X_1,\ X_2,\ \ldots,\ X_n\}$
    local variables: $queue$, a queue of arcs, initially all the arcs in $csp$

    while $queue$ is not empty do
        $(X_i,\ X_j) \leftarrow$ 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 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 $removed$
```

$O(n^2 d^3)$, can be reduced to $O(n^2 d^2)$
but cannot detect all failures in poly time!

Problem structure



Tasmania and mainland are independent subproblems

Identifiable as connected components of constraint graph

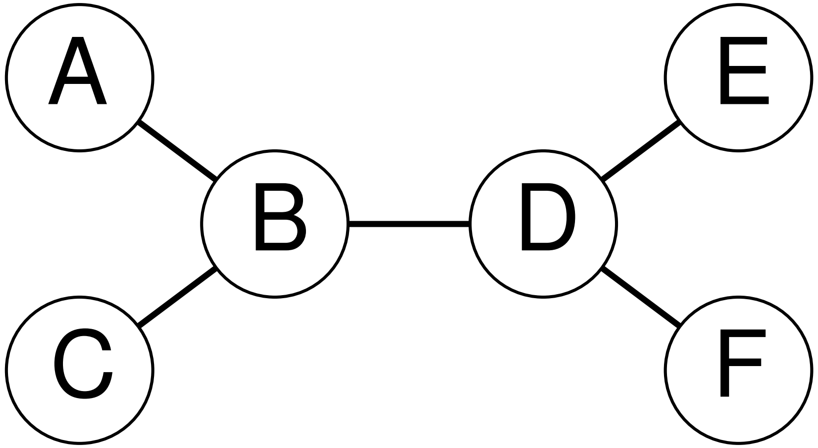
Problem structure contd.

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



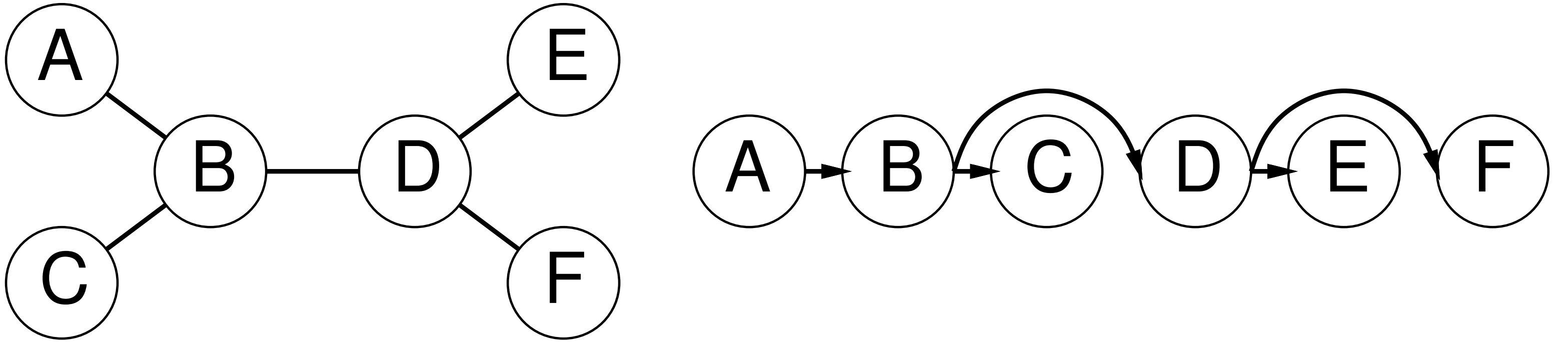
Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\,d^2)$ time

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

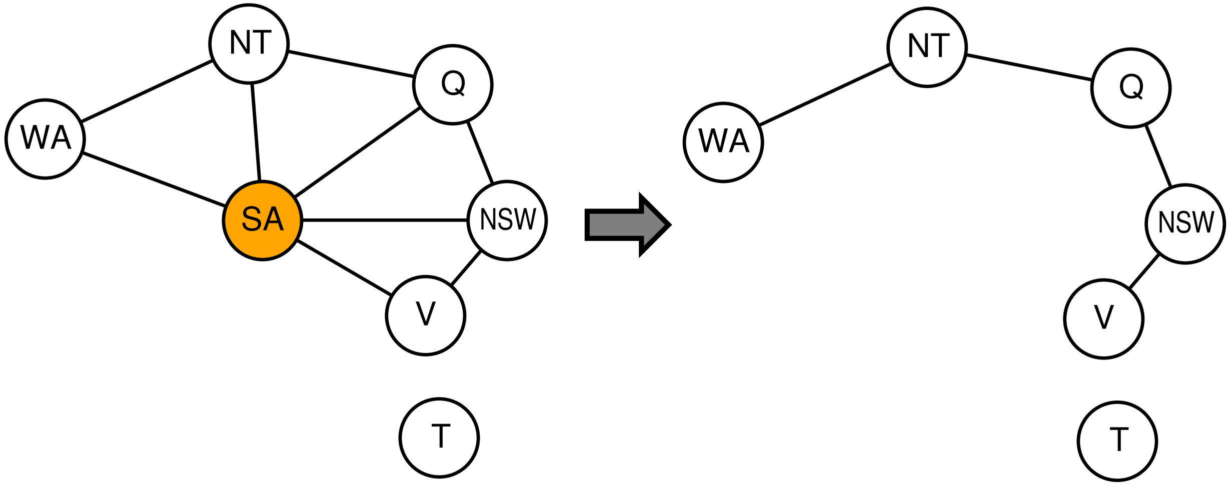


2. For j from n down to 2, apply $\textsc{RemoveInconsistent}(Parent(X_j), X_j)$

3. For j from 1 to n, assign X_j consistently with $Parent(X_j)$

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

Summary

Variable ordering and value selection heuristics help backtracking 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