

CONSTRAINT SATISFACTION PROBLEMS

CHAPTER 5

Outline

- ◊ CSP examples
- ◊ Backtracking search for CSPs
- ◊ Problem structure and problem decomposition
- ◊ Local search for CSPs

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

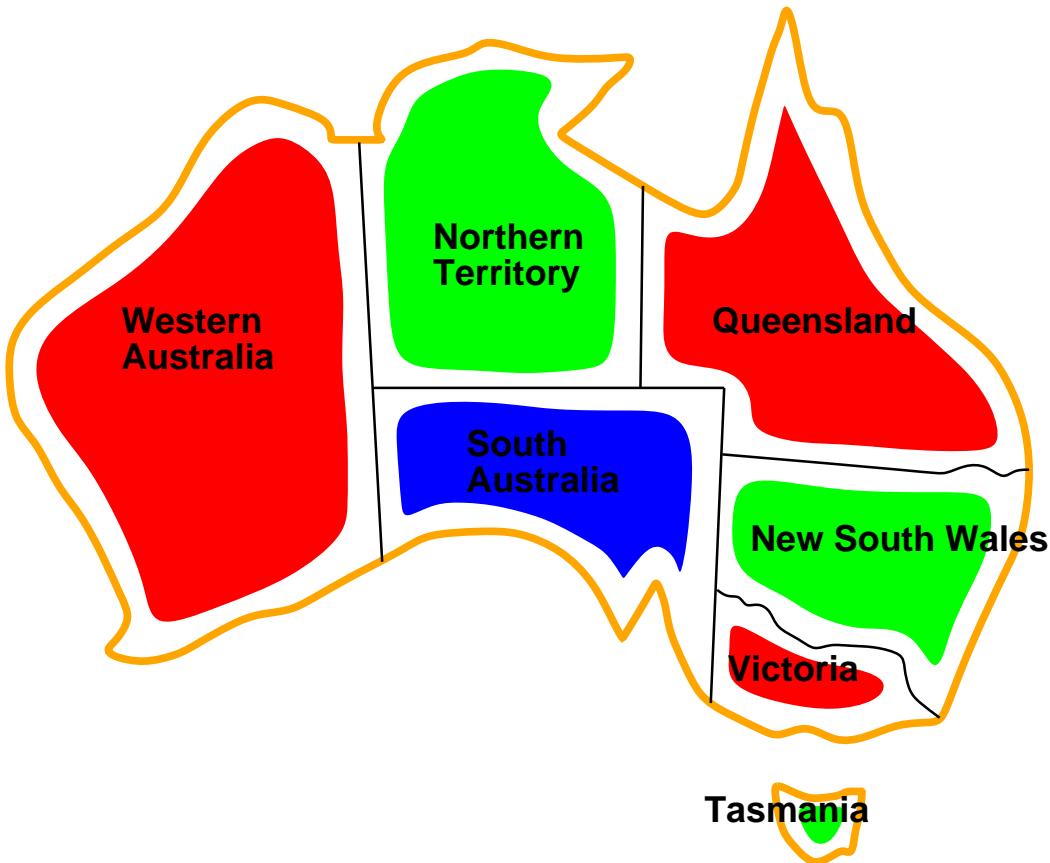
Domains $D_i = \{\text{red, green, blue}\}$

Constraints: adjacent regions must have different colors

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

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

Example: Map-Coloring contd.



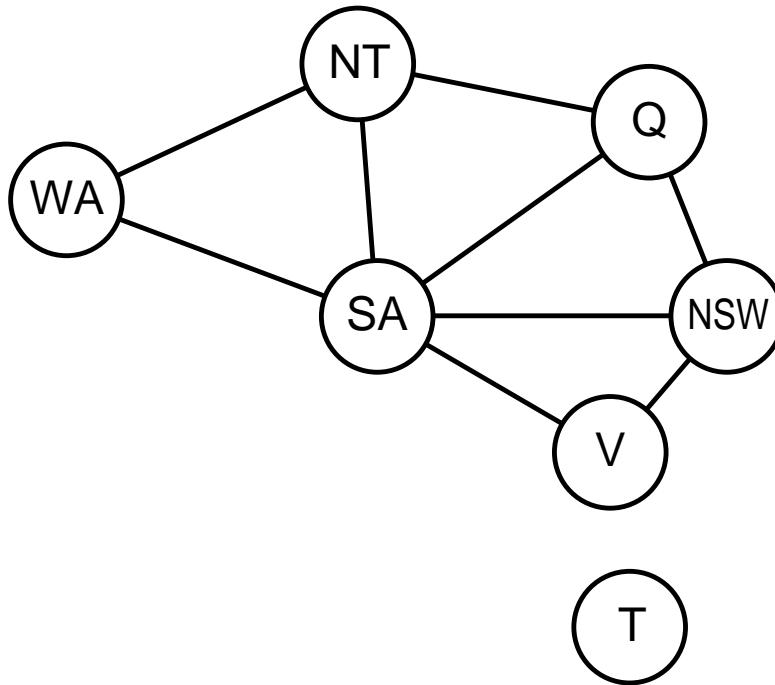
Solutions are assignments satisfying all constraints, e.g.,

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

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!

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 constraints

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., cryptarithmetic 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

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

Backtracking search

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

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

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 $n \approx 25$

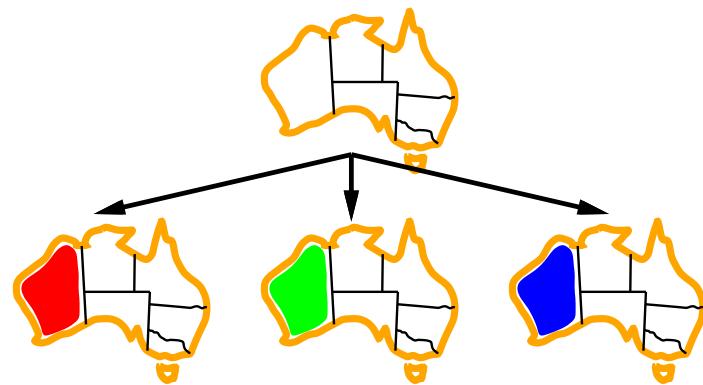
Backtracking search

```
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  $\leftarrow$  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  $\leftarrow$  RECURSIVE-BACKTRACKING(assignment, csp)
            if result  $\neq$  failure then return result
            remove  $\{var = value\}$  from assignment
    return failure
```

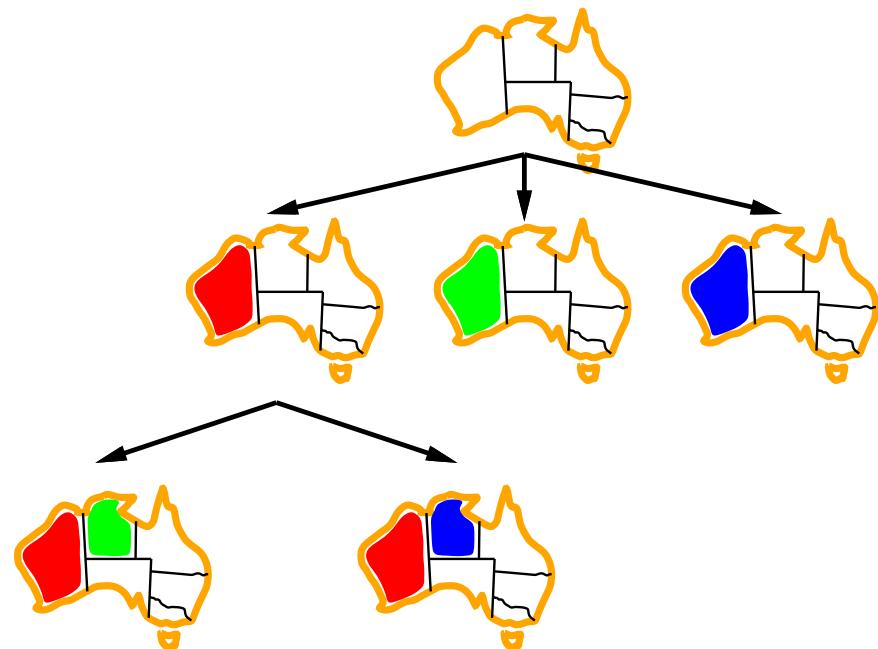
Backtracking example



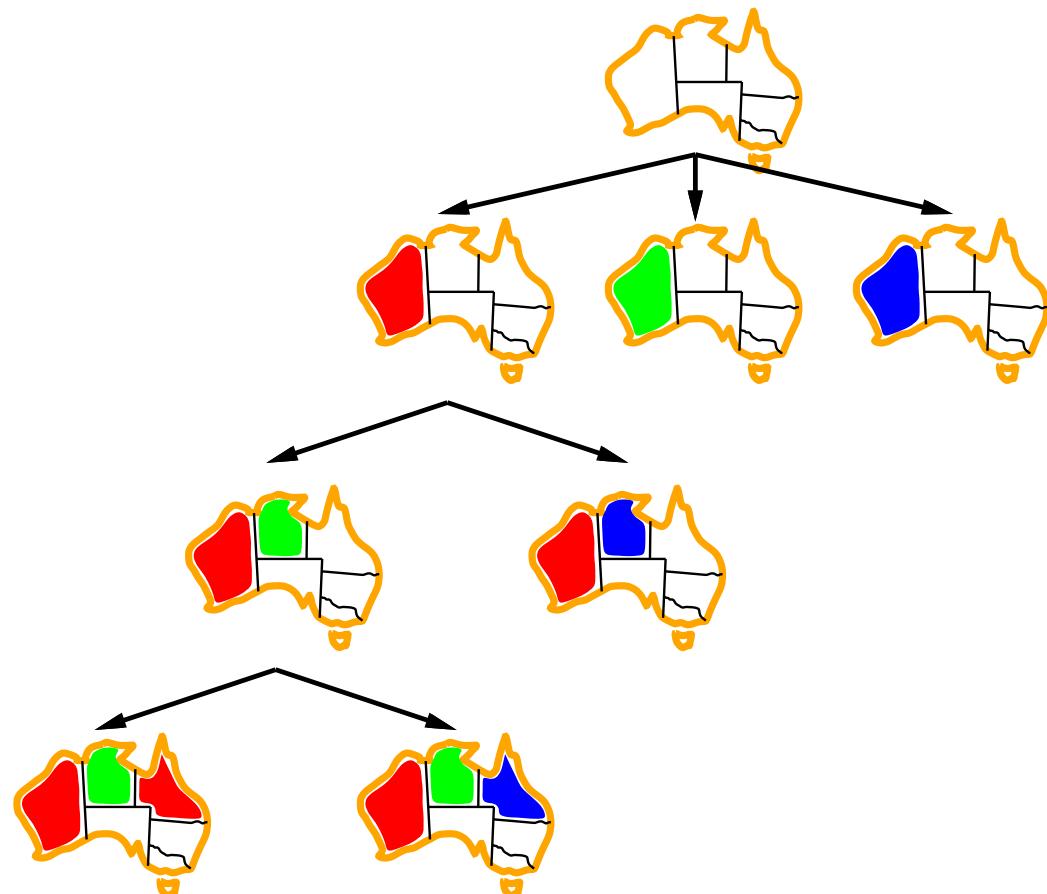
Backtracking example



Backtracking example



Backtracking example



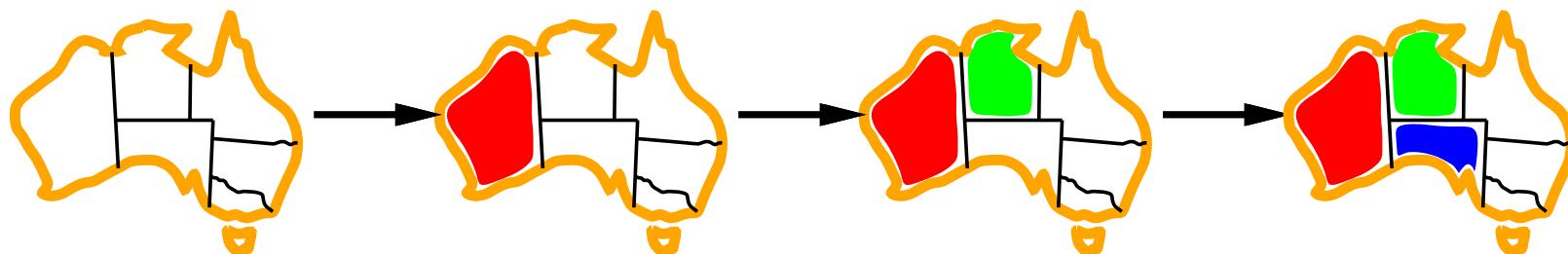
Improving backtracking efficiency

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?

Minimum remaining values

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

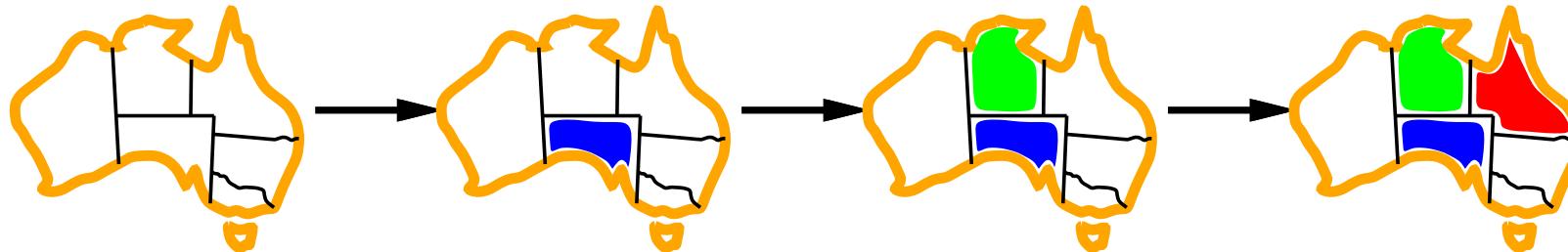


Degree heuristic

Tie-breaker among MRV variables

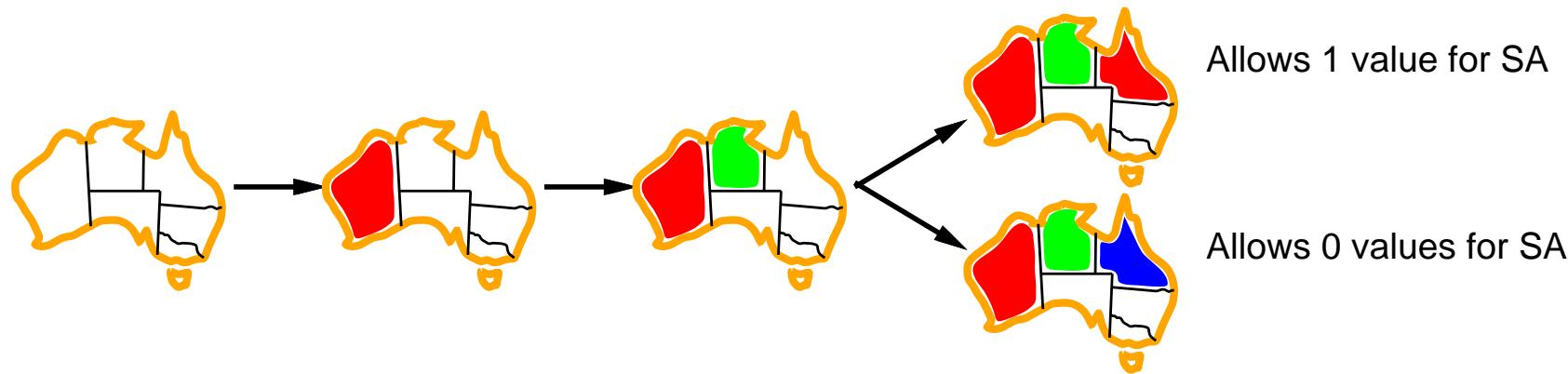
Degree heuristic:

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

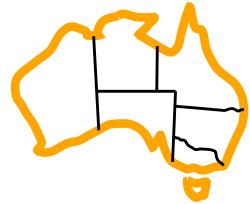


Combining these heuristics makes 1000 queens feasible

Forward checking

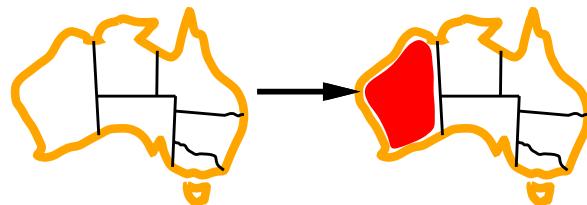
Idea: Keep track of remaining legal values for unassigned variables

Terminate search when any variable has no legal values



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
Red		Green	Blue	Red	Green	Blue

Forward checking

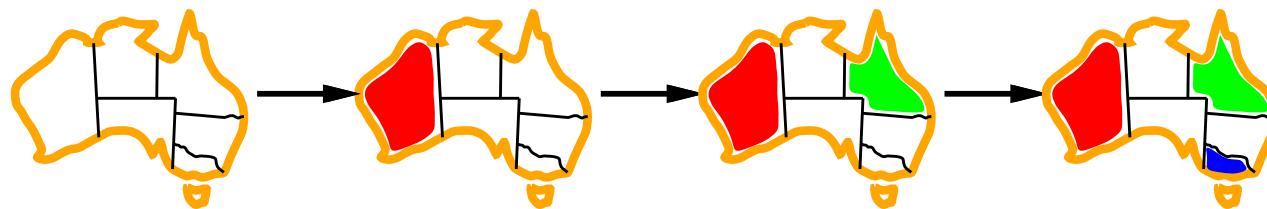
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



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

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
Red		Green	Blue	Red	Green	Blue
Red		Blue	Green	Red	Blue	Red
Red		Green		Red	Blue	Red

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