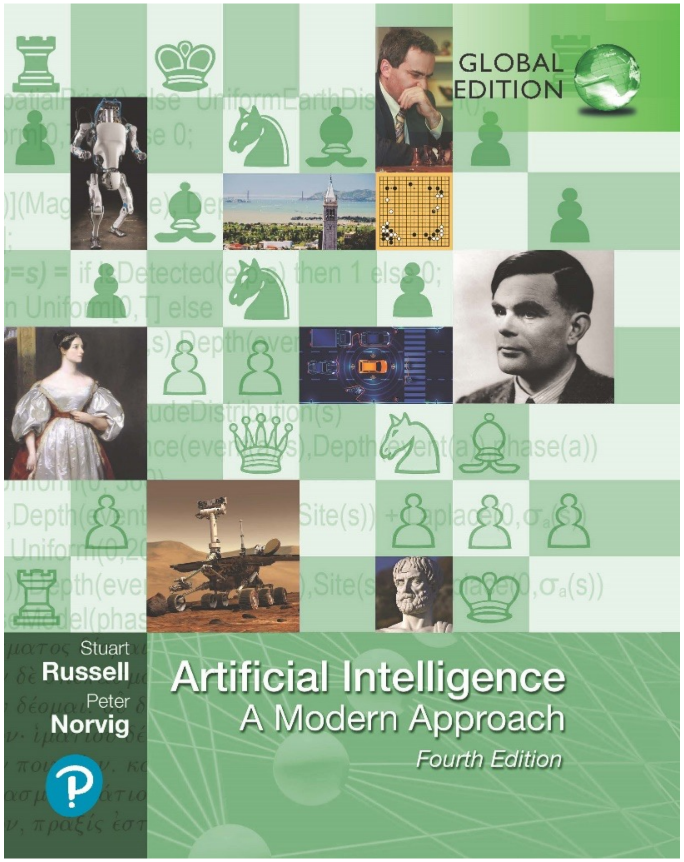
Artificial Intelligence: A Modern Approach

Fourth Edition, Global Edition



Chapter 5

Constraint Satisfaction Problems

Outline

Defining Constraint Satisfaction Problems (CSP)

CSP examples

Backtracking search for CSPs

Local search for CSPs

Problem structure and problem decomposition

Defining Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) consists of three components, *X*, *D*, and *C:*

- *X* is a set of variables, $\{X_1 ..... X_n\}$.
- *D* is a set of domains, $\{D_1, ... , D_n\}$, one for each variable
- *C* is a set of constraints that specify allowable combination of values

CSPs deal with assignments of values to variables.

- A complete assignment is one in which every variable is assigned a value, and a solution to a CSP is a consistent, complete assignment.
- A partial assignment is one that leaves some variables unassigned.
- Partial solution is a partial assignment that is consistent

Constraint satisfaction problems (CSPs)

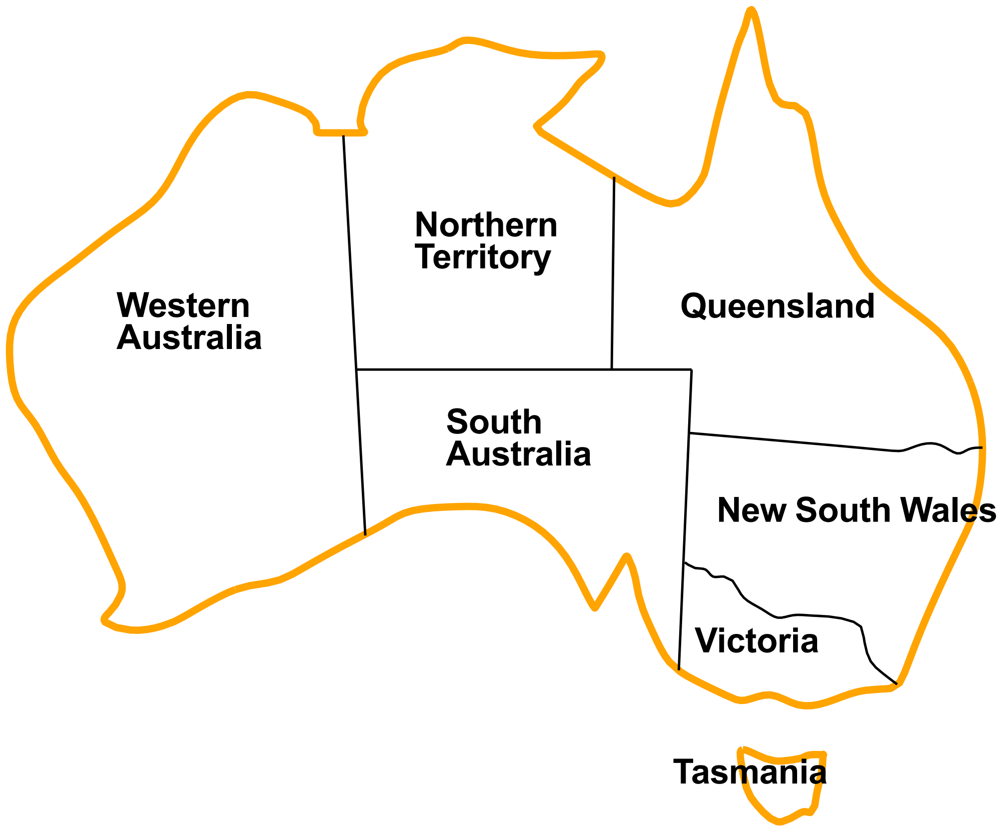
- Standard search problem:

  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

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

Example: Map-Coloring



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

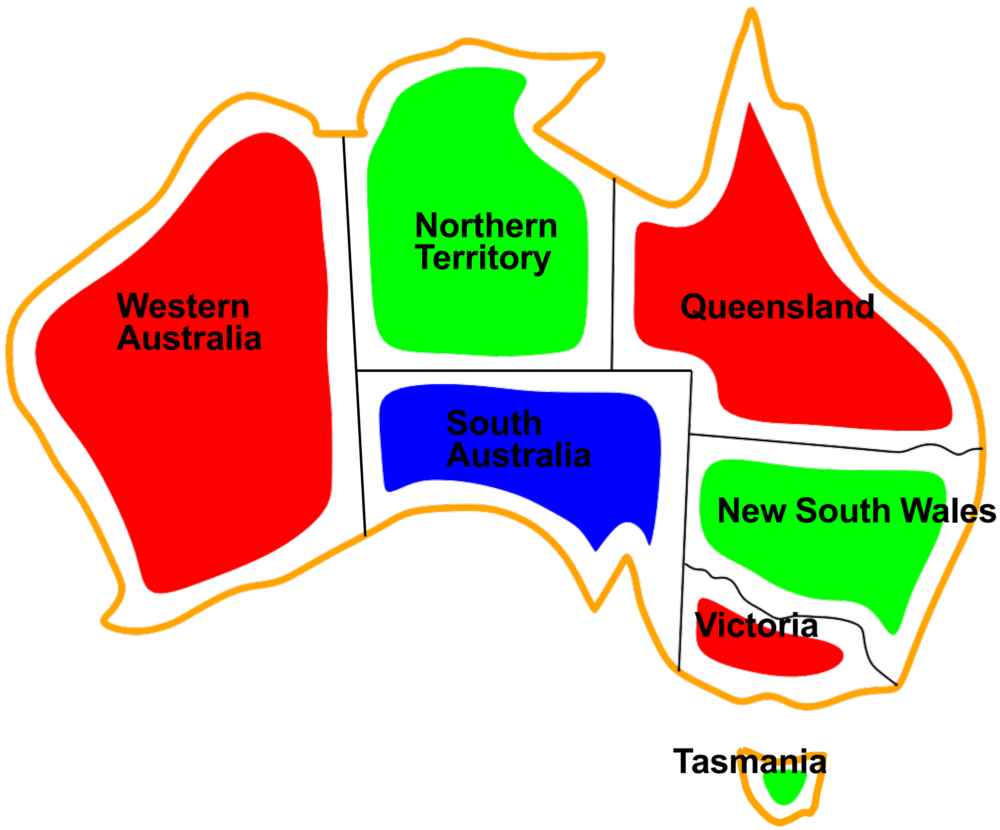
Domains $D_i = \{red, green, blue\}$

Constraints: adjacent regions must have different colors

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

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

Example: Map-Coloring contd.

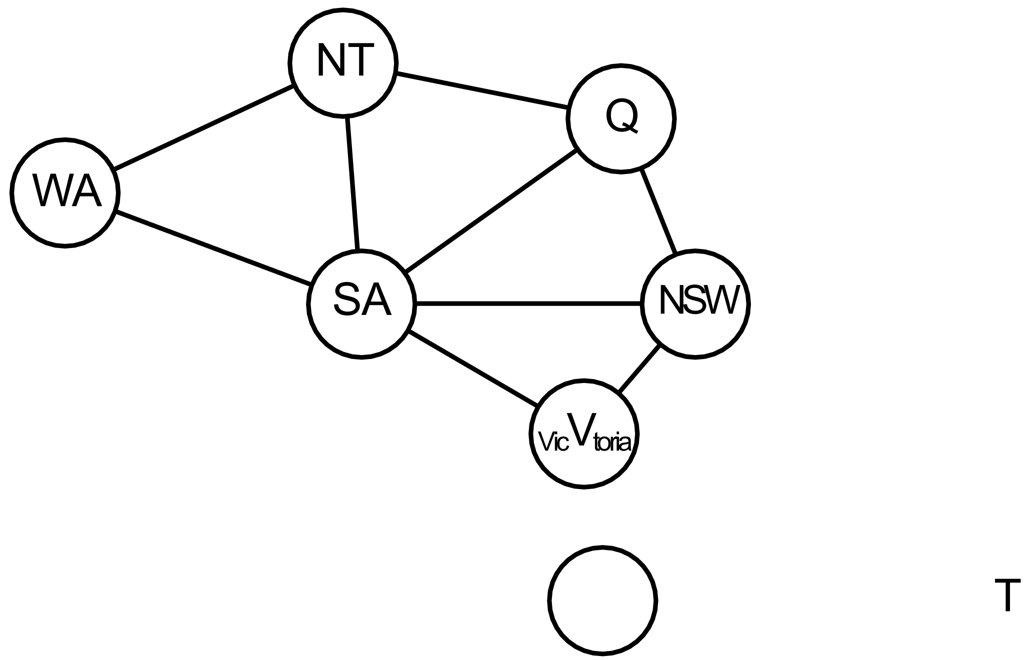


Solutions are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

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., $SA \neq green$

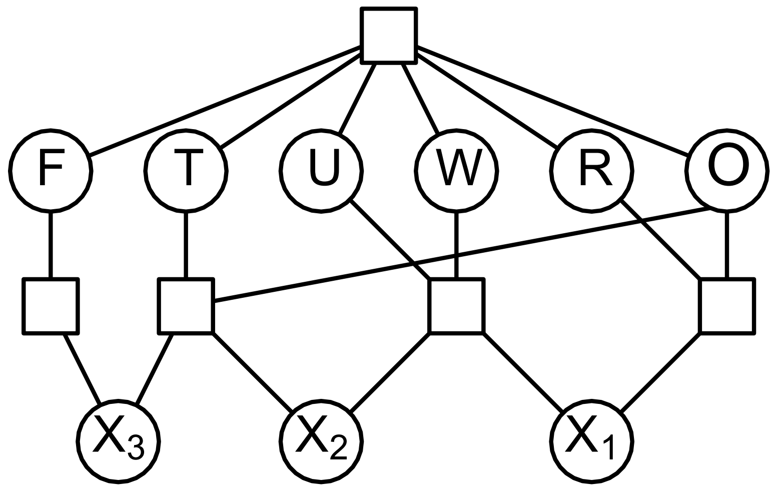
Binary constraints involve pairs of variables,
e.g., $SA \neq WA$

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

Example: Cryptarithmetic

```
    T  W  O
 +  T  W  O
 ----------
 F  O  U  R
```



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.

Real-world CSPs

Assignment problems
    e.g., who teaches what class

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

Hardware configuration

Transportation scheduling

Factory scheduling Floor

planning

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 any unassigned variable that does not conflict with current assignment.

$\Rightarrow$ fail if no legal assignments (not fixable!)

Goal test: the current assignment is complete

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 any unassigned variable that does not conflict with current assignment.

$\Rightarrow$ 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
   $\Rightarrow$ use depth-first search
3) Path is irrelevant, so can also use complete-state formulation
4) $b = (n-k)d$ at depth k, hence $n!d^n$ leaves!!!!

Backtracking search

Variable assignments are commutative, i.e.,

[$WA = red$ then $NT = green$] same as [$NT = green$ 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

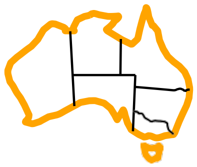
  CSPs Can solve n-queens for $n \approx 25$

Backtracking search

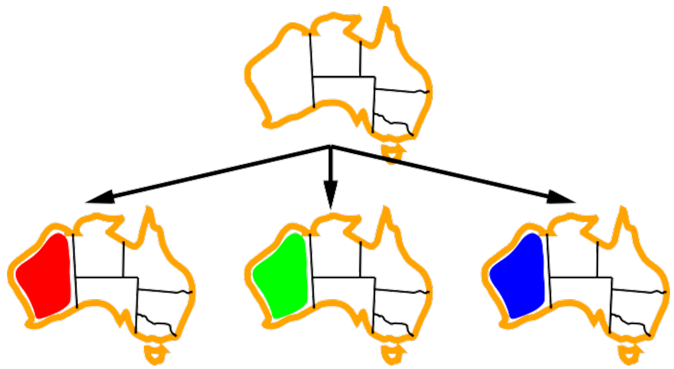
```
function Backtracking-Search(csp) returns solution/failure
    return Recursive-Backtracking({ }, csp)

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
```

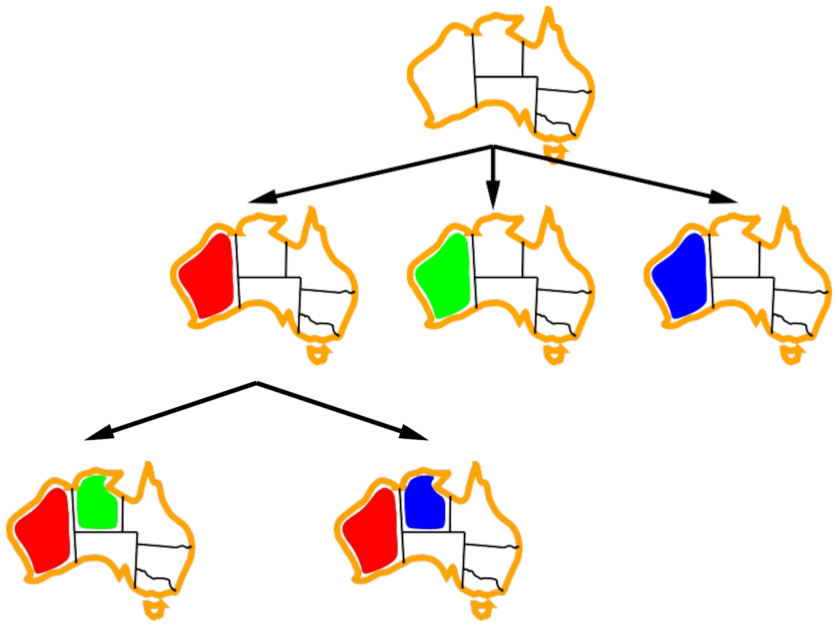
Backtracking example



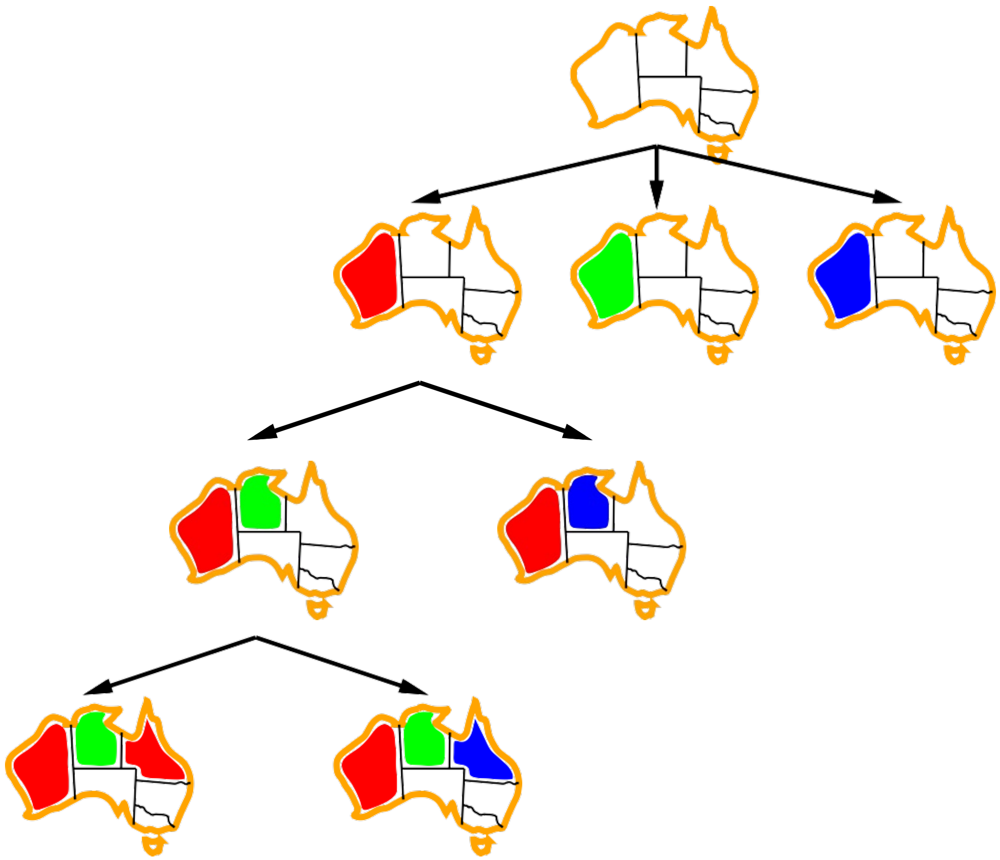
Backtracking example



Backtracking example



Backtracking example



Improving backtracking efficiency

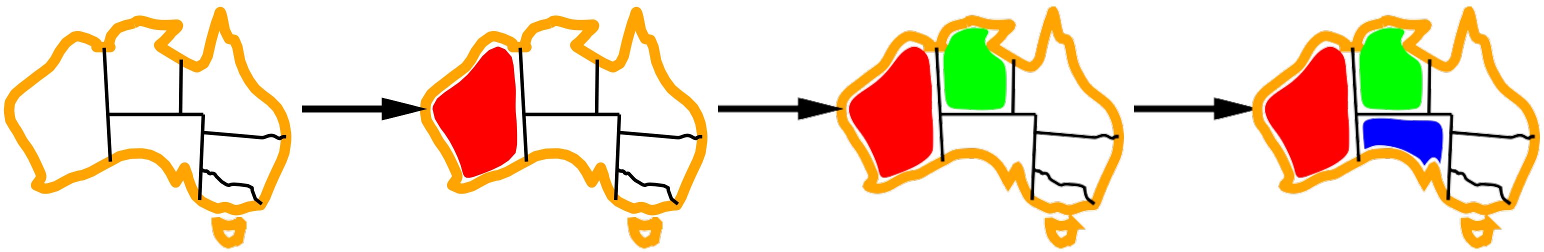
General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?
2. Which value should we assign to the selected variable?
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

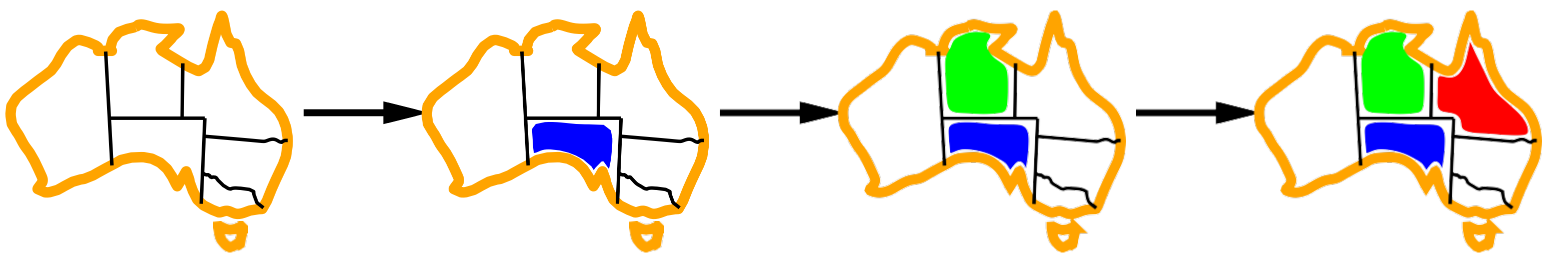


Degree heuristic in Tied Situations

Tie-breaker among MRV variables

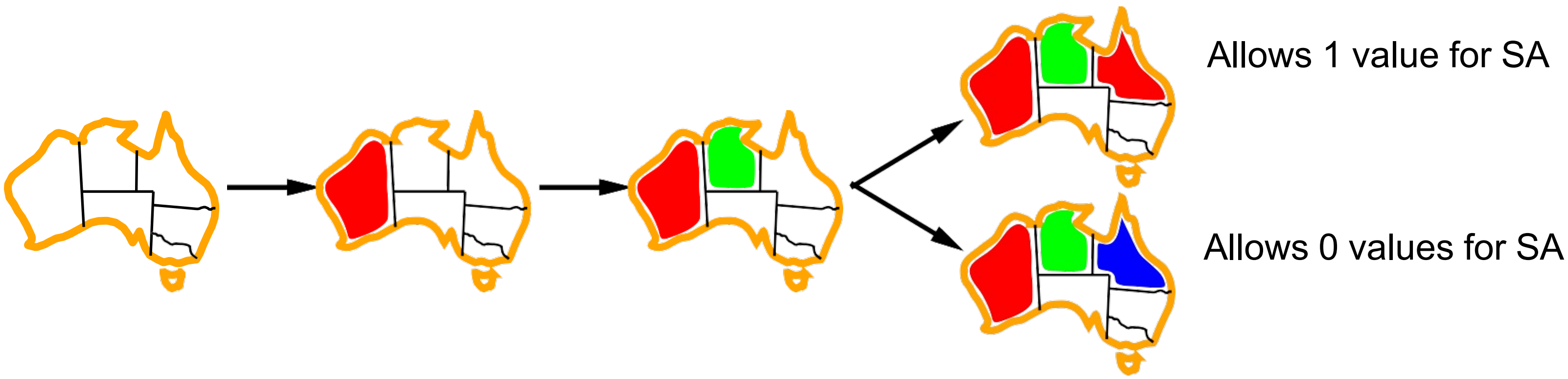
Degree heuristic:

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

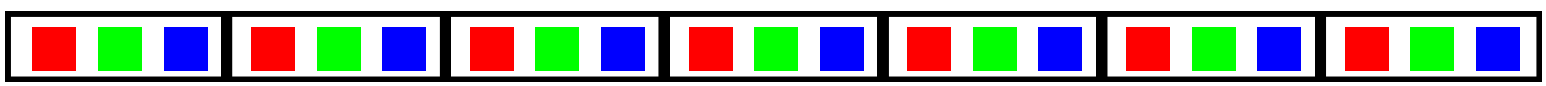


Combining these heuristics makes 1000 queens feasible

Forward checking

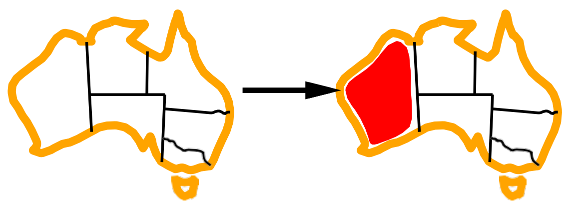
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

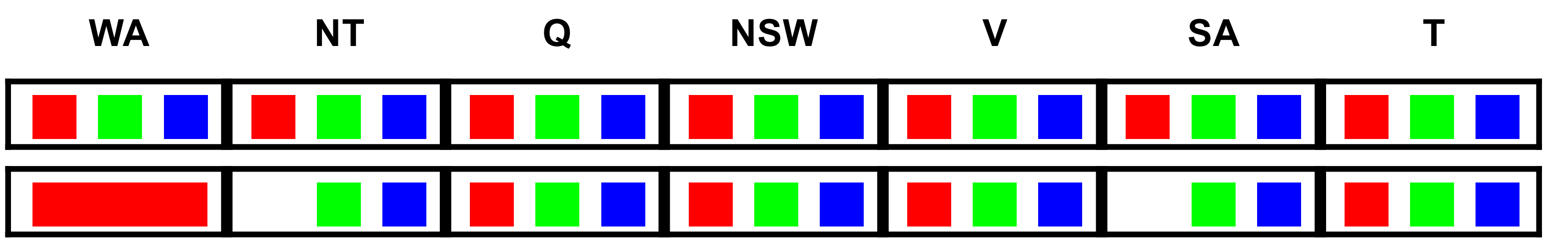
WA NT Q NSW V SA T



Forward checking

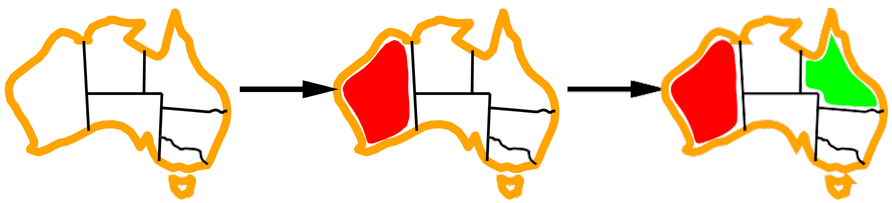
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

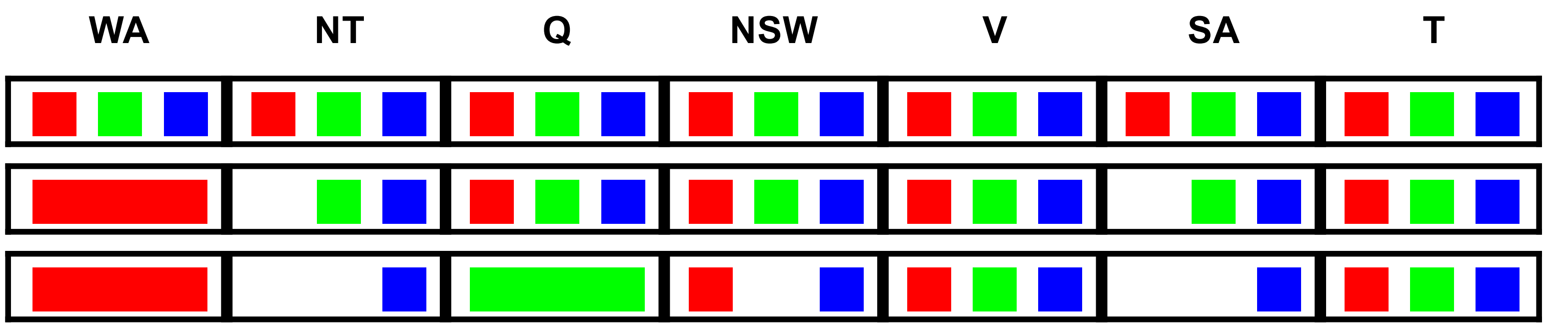




Forward checking

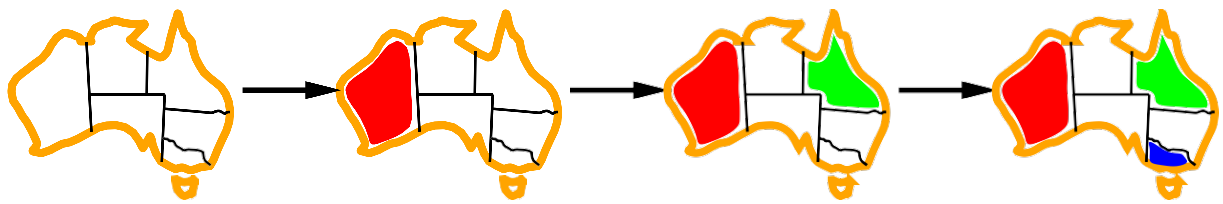
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

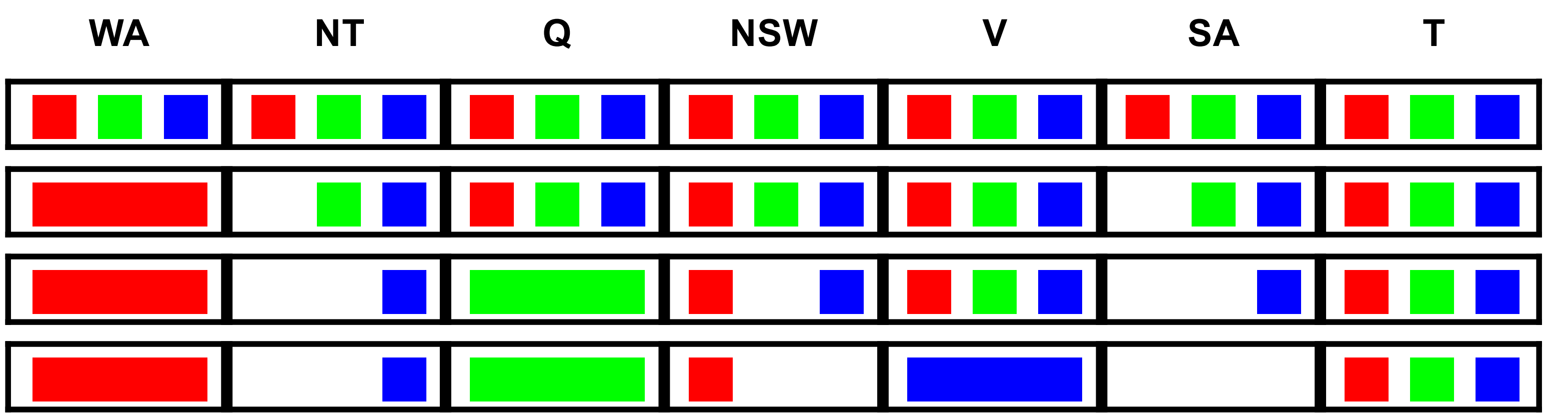




Forward checking

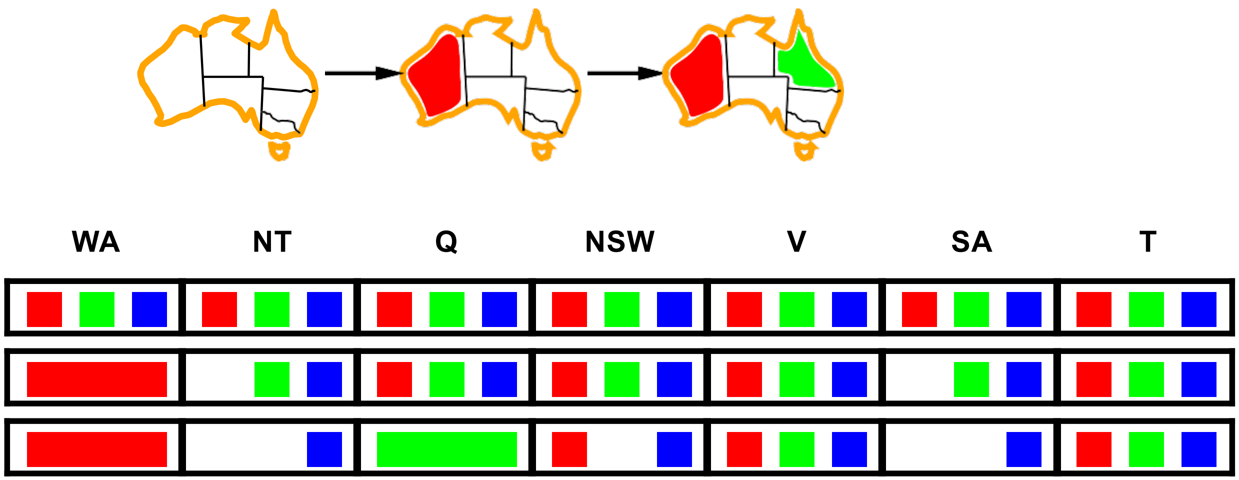
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values





Constraint propagation

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



NT and SA cannot both be blue!

Constraint propagation repeatedly enforces constraints locally

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

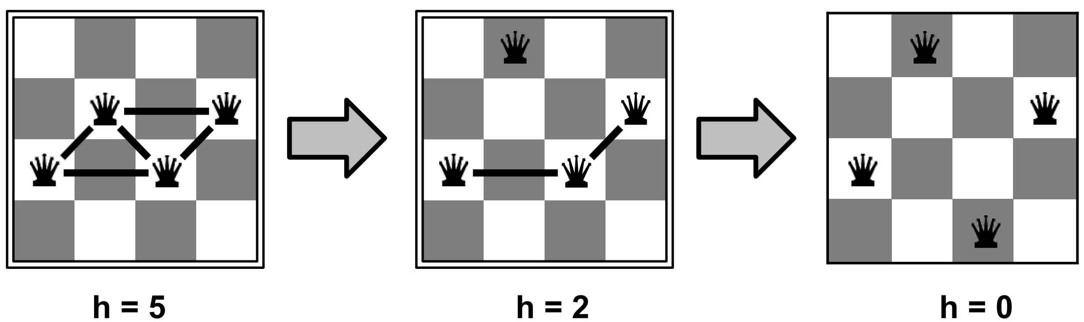
Example: 4-Queens

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

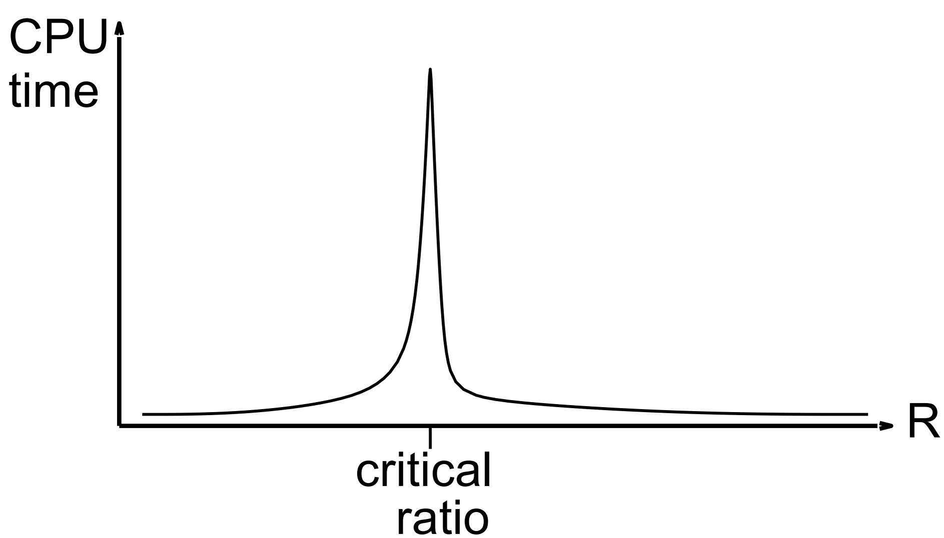


Performance of min-conflicts

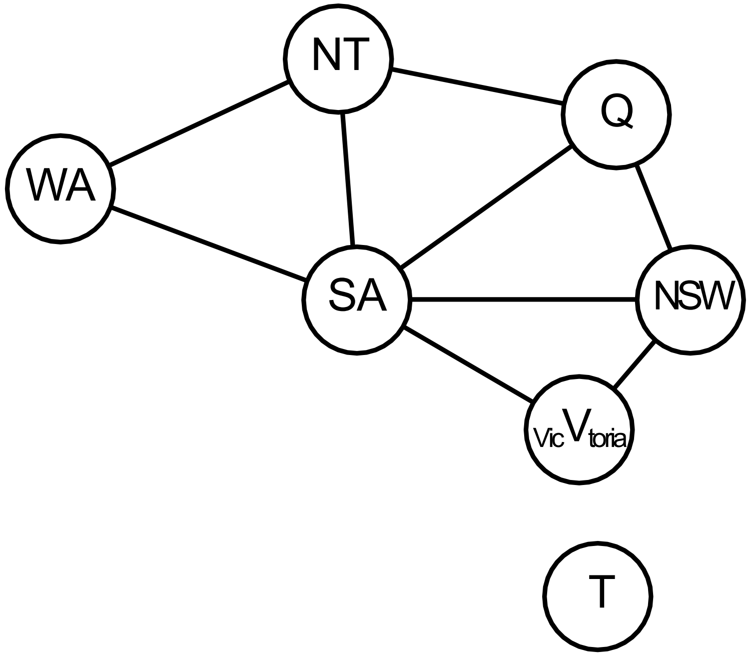
Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., $n = 10{,}000{,}000$)

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

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



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., $n = 80$ variables, $d = 2$ domain size, $c = 20$ variables in each subproblem

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

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

Pearson

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

Local search using the min-conflicts heuristic has also been applied to constraint satisfaction problems with great success

The CSP representation allows analysis of problem structure

Tree-structured CSPs can be solved in linear time