ARTIFICIAL INTELLIGENCE

Constraint Satisfaction Problem

CONSTRAINT SATISFACTION PROBLEMS (CSPS)

Standard search problem:

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

CSP:

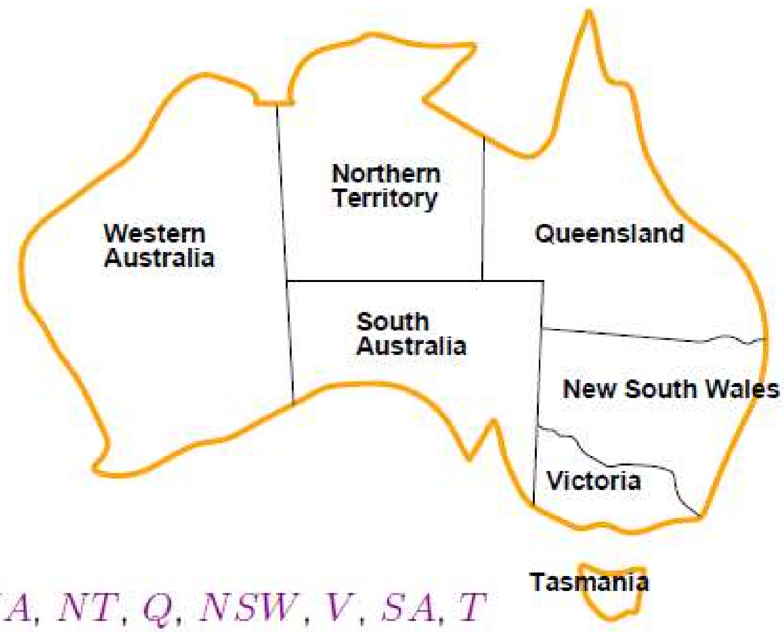
- state is defined by variables X_i with values from domain D_i
- goal test is a set of constraints specifying allowable combinations of values for subsets of variables

Simple example of a formal representation language

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

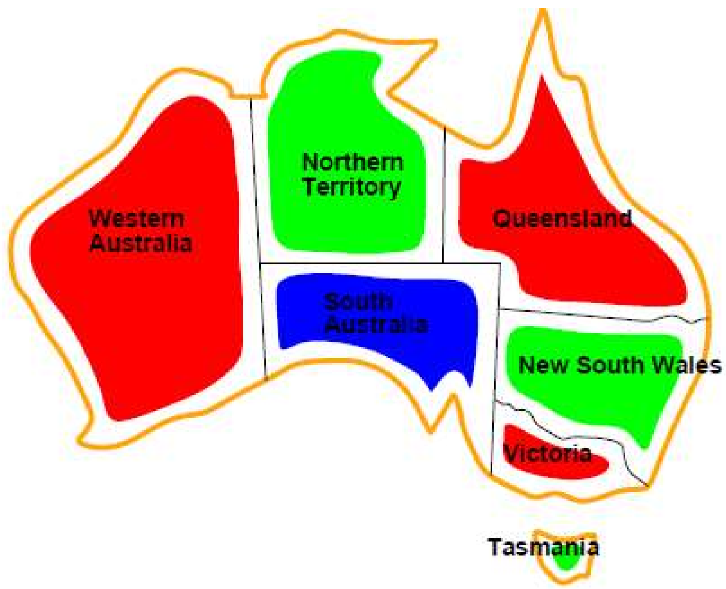
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

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

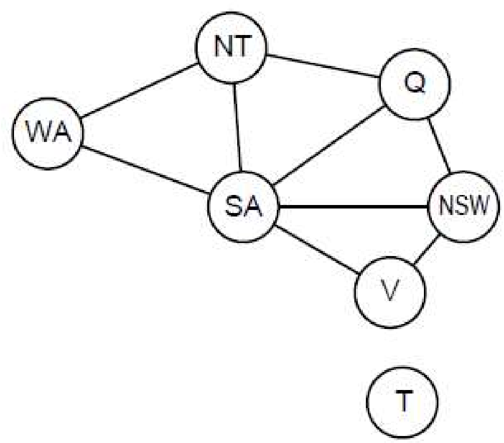


Solutions are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

CONSTRAINT GRAPHS

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 CSP

Discrete variables

- finite domains; size d; O(dn) complete assignments

- Infinite domains (integers, strings, etc.)

Continuous variables

VARIETIES OF CONSTRAINTS

Unary constraints involve a single variable,

- e.g., SA != green

Binary constraints involve pairs of variables,

- e.g., SA != WA

Higher-order constraints involve 3 or more variables,

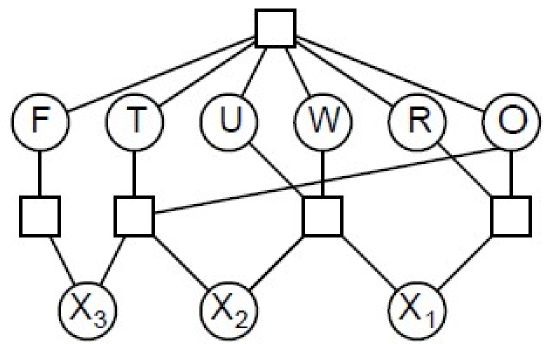
- e.g., cryptarithmetic column constraints

Preferences (soft constraints),

- e.g., red is better than green often 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 X1 X2 X3

Domains: (0; 1; 2; 3; 4; 5; 6; 7; 8; 9)

Constraints

alldiff(F; T;U;W;R;O)

$O + O = R + 10\ X_1$, 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

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, f g
- 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

This is the same for all CSPs!

Every solution appears at depth n with n variables use depth-first search

Path is irrelevant, so can also use complete-state formulation

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

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

Backtracking search is the basic uninformed algorithm for CSPs

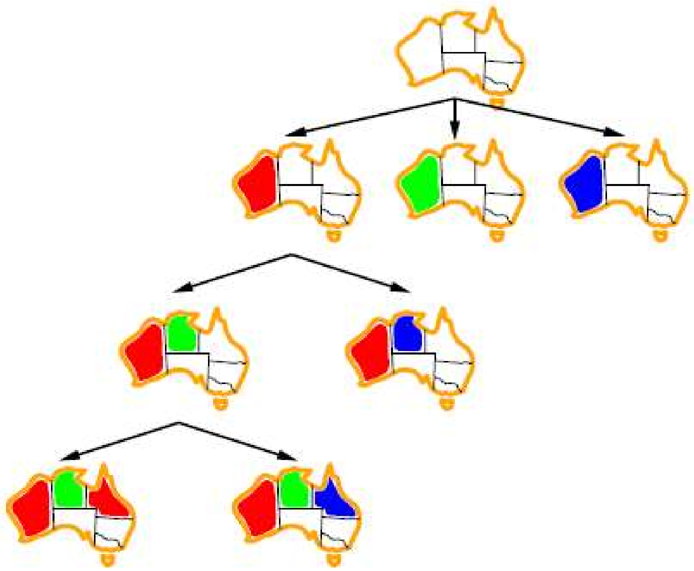
Can solve n-queens for $n \sim 25$

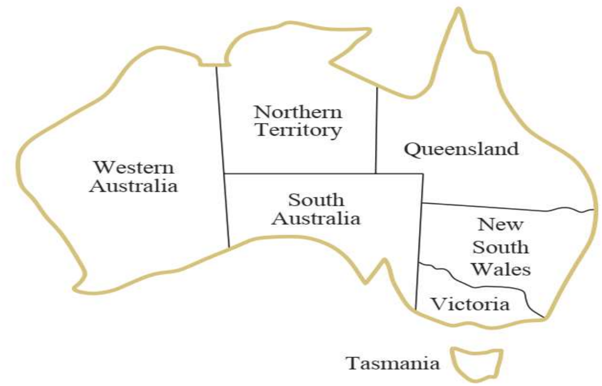
BACKTRACKING SEARCH

```
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



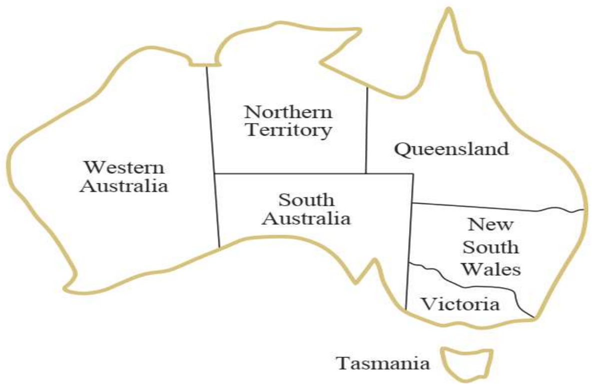


IMPROVING BACKTRACKING EFFICIENCY

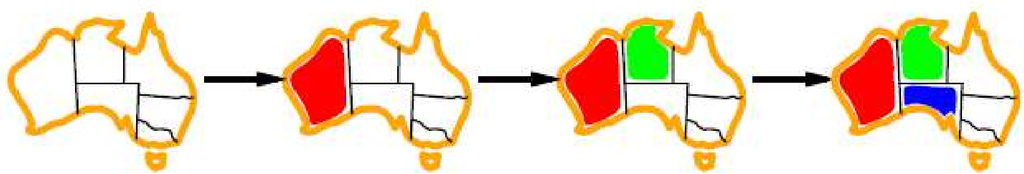
General-purpose methods can give huge gains in speed:

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

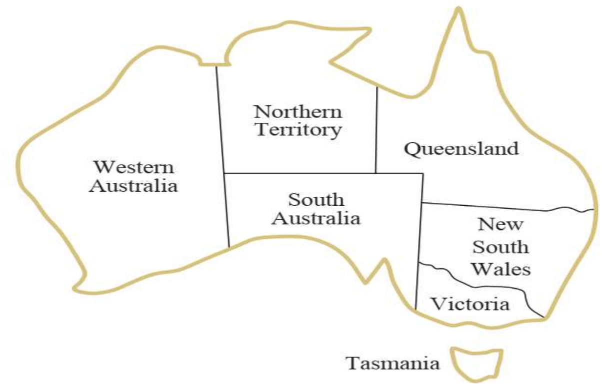
1. MINIMUM REMAINING VALUES



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

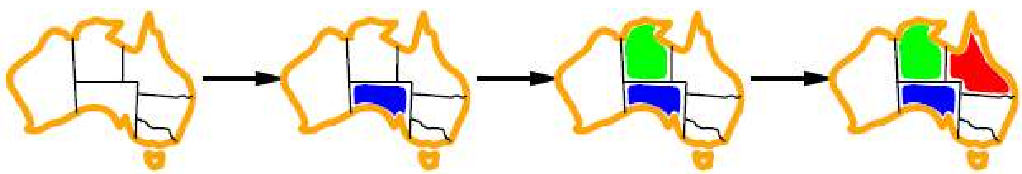


2. DEGREE HEURISTIC

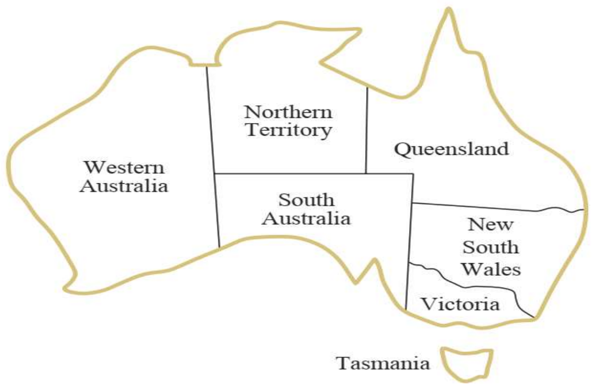


Tie-breaker among MRV variables

Degree heuristic: choose the variable with the most constraints on remaining variables

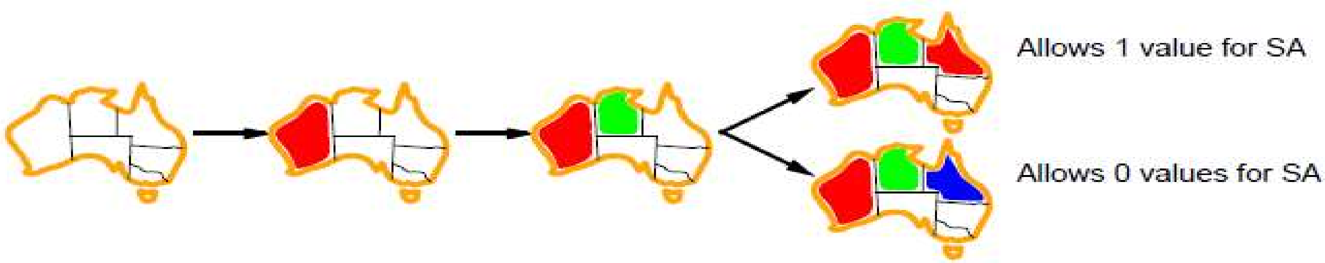


3. LEAST CONSTRAINING VALUE

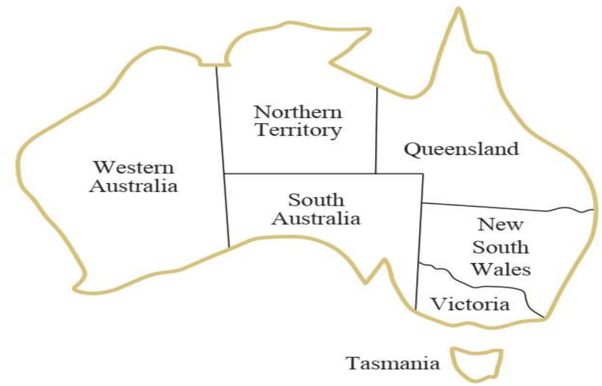


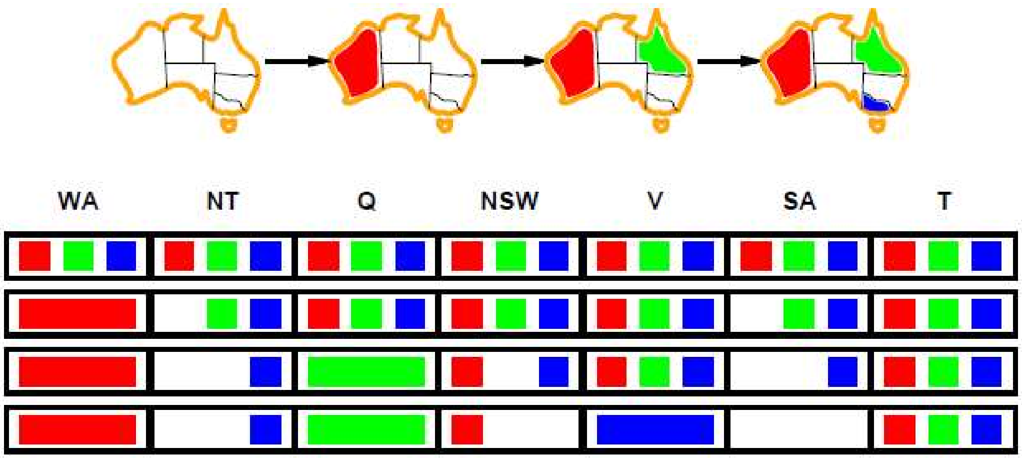
Given a variable, choose the least constraining value: the one that rules out the fewest values in the remaining variables.

Combining these heuristics makes 1000 queens feasible



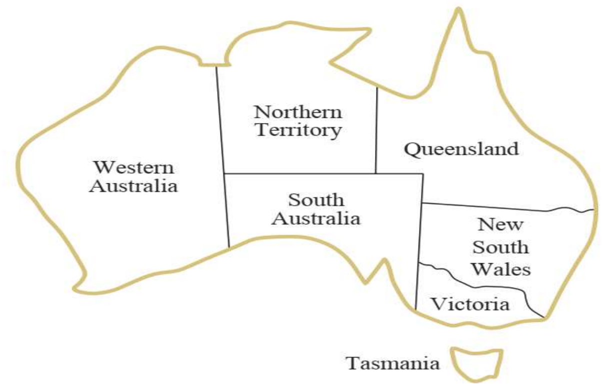
FORWARD CHECKING



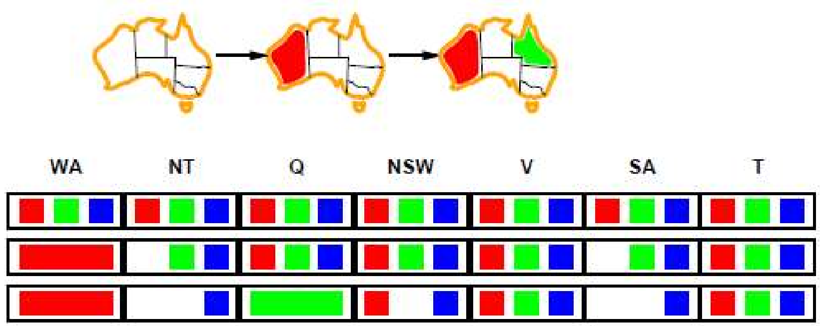


Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

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!

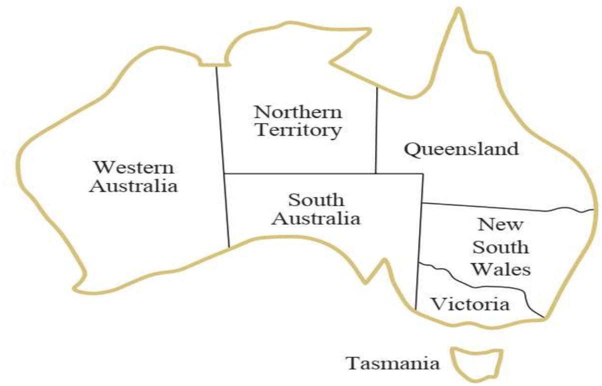
Constraint propagation repeatedly enforces constraints locally

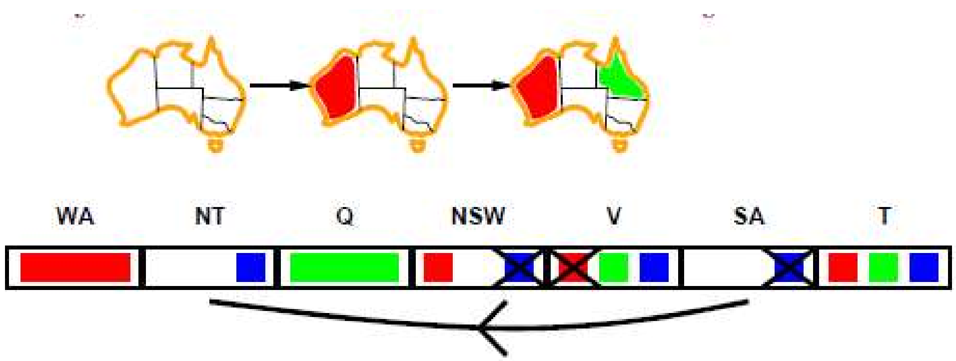
ARC CONSISTENCY

Simplest form of propagation makes each arc consistent

X → 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

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