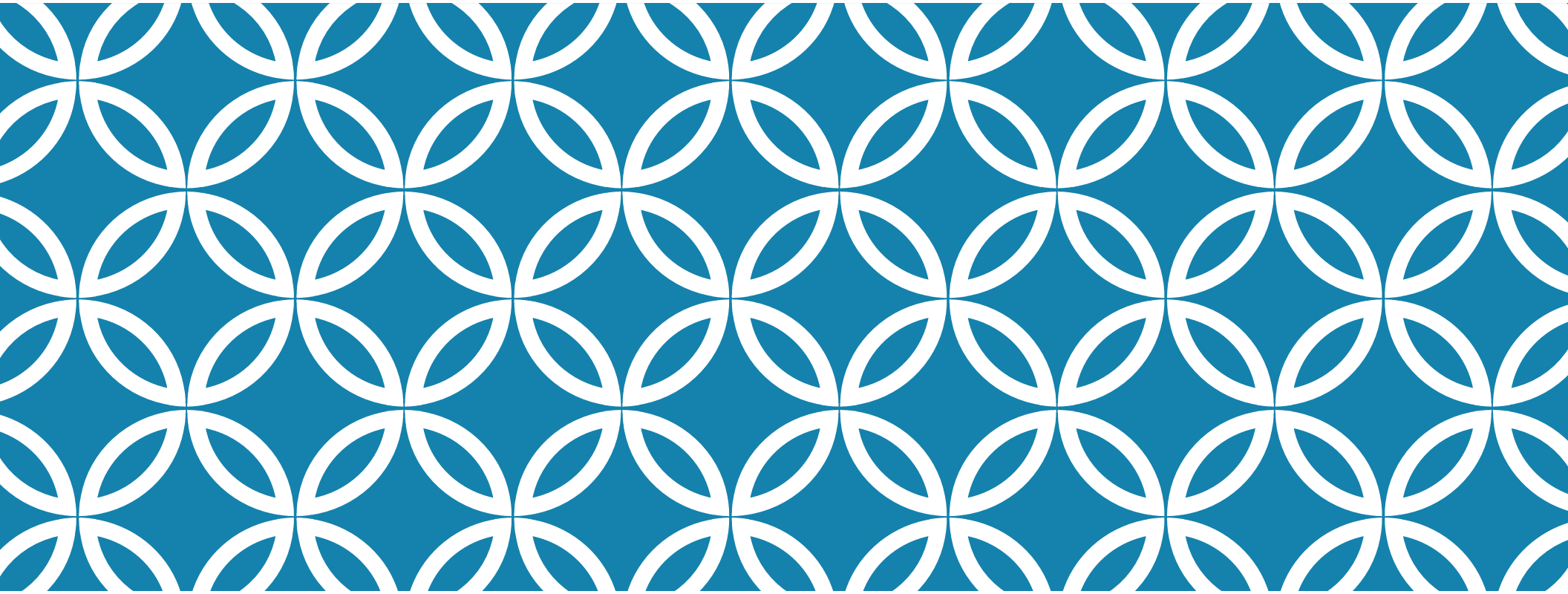


ARTIFICIAL INTELLIGENCE

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

DEFINING CONSTRAINT SATISFACTION PROBLEMS

A constraint satisfaction problem consists of three components, X;D; and C:
X is a set of variables, {X1,…,Xn}.
D is a set of domains, {D1,…,Dn}, one for each variable.
C is a set of constraints that specify allowable combinations of values.

A domain, Di, consists of a set of allowable values, {v1,…,vk}, for variable Xi.

Each constraint Cj consists of a pair {scope;rel}, where

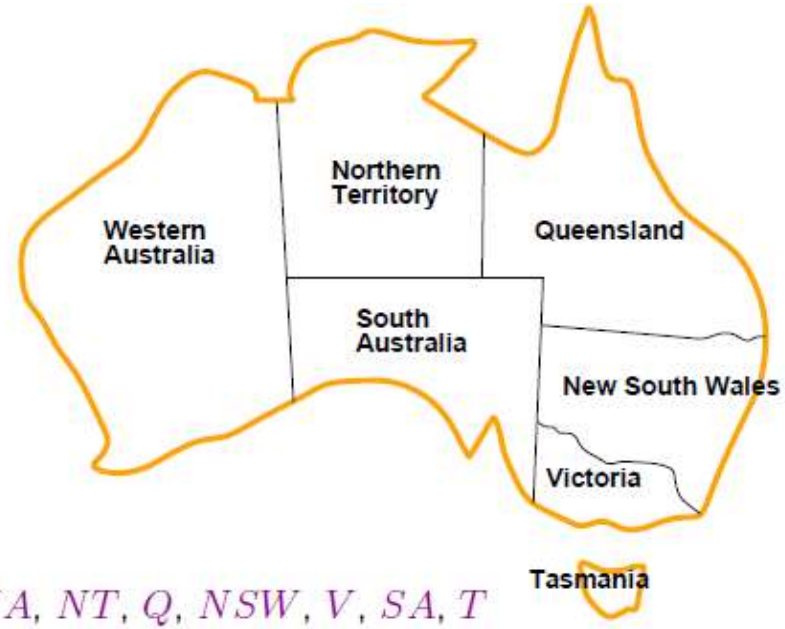
- scope is a tuple of variables that participate in the constraint and
- rel is a relation that defines the values that those variables can take on.
- A relation can be represented as an explicit set of all tuples of values that satisfy the constraint, (X1;X2);{(3;1);(3;2);(2;1)}
- or as a function that can compute whether a tuple is a member of the relation.
- {(X1;X2);X1 > X2i.}

CSP

CSPs deal with assignments of values to variables, {Xi = vi;Xj = vj;…} .

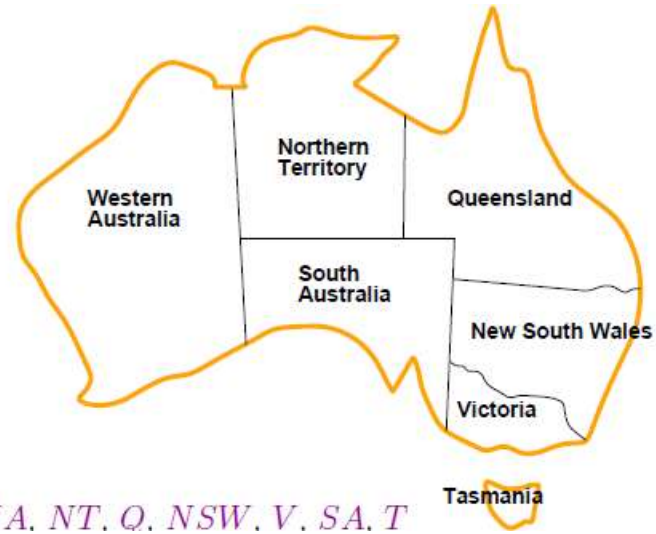
An assignment that does not violate any constraints is called a consistent or legal assignment

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



There are nine constraints:

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

$$C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}.$$

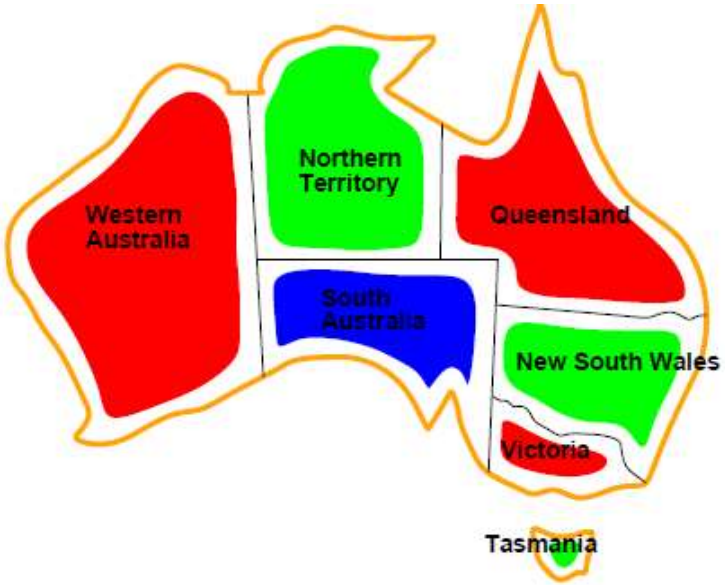
Here we are using abbreviations; $SA \neq WA$

is a shortcut $\langle (SA, WA), SA \neq WA \rangle,$

Where it can be fully enumerated in turn as

$$\{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}.$$

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

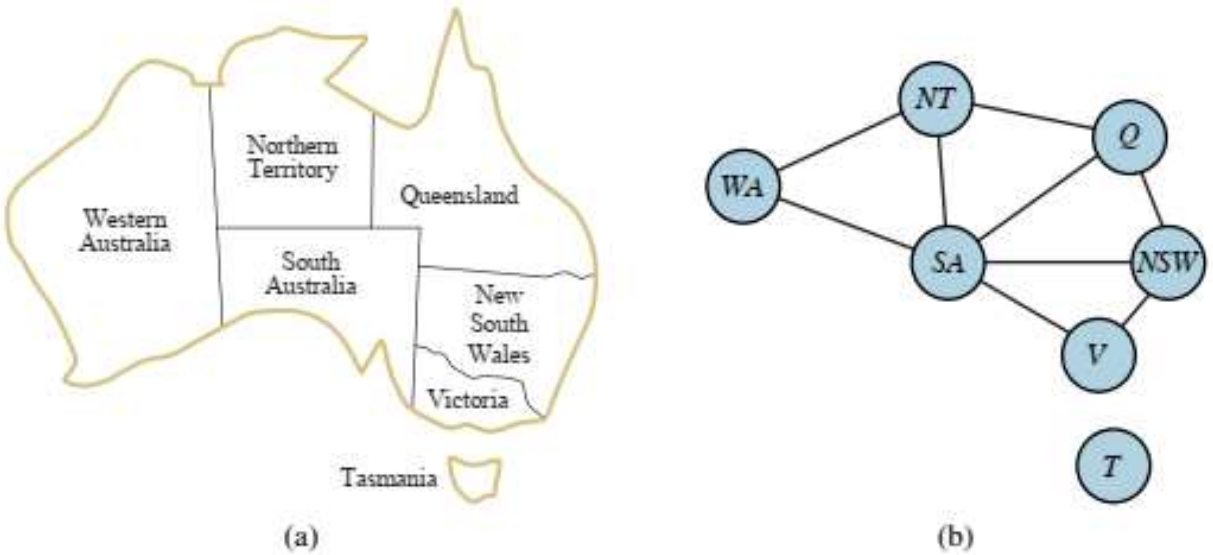
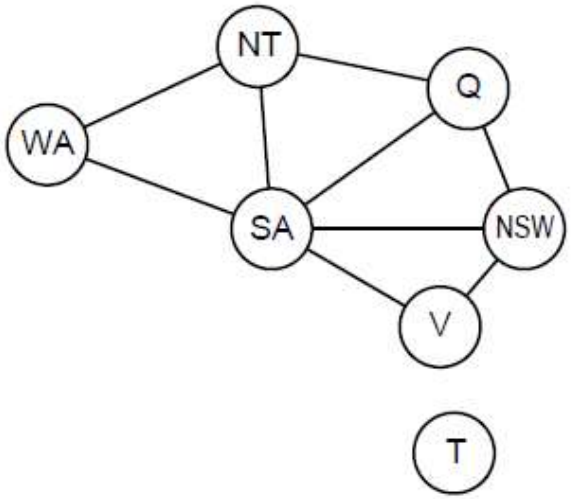


Figure 5.1 (a) The principal states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem (CSP). The goal is to assign colors to each region so that no neighboring regions have the same color. (b) The map-coloring problem represented as a constraint graph.

CONSTRAINT GRAPHS

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!



VARIATIONS ON 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

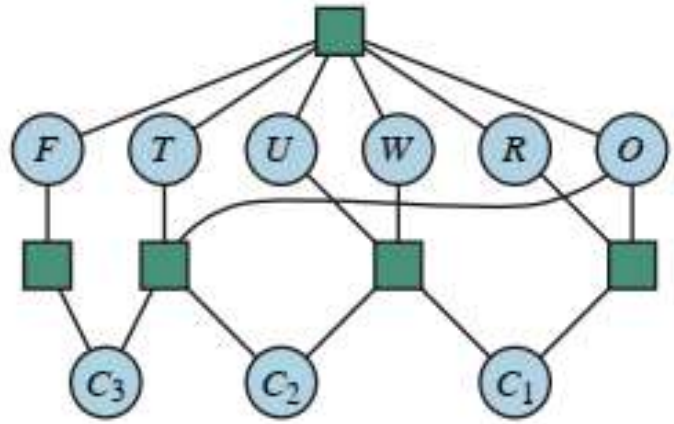
Preferences (soft constraints),

- e.g., red is better than green often by a cost for each variable assignment → constrained optimization problems

CRYPTARITHMETIC

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

(a)

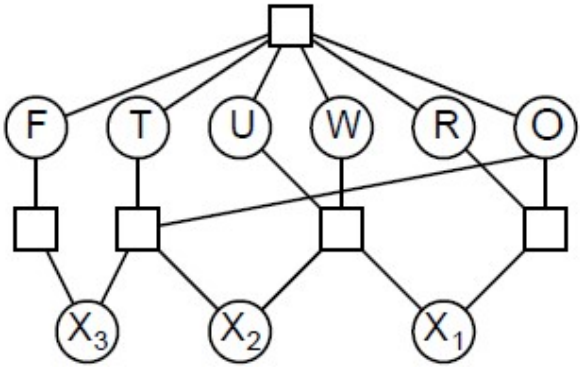


(b)

Figure 5.2 (a) A cryptarithmetic problem. Each letter stands for a distinct digit; the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, with the added restriction that no leading zeroes are allowed. (b) The constraint hypergraph for the cryptarithmetic problem, showing the *Alldiff* constraint (square box at the top) as well as the column addition constraints (four square boxes in the middle). The variables C_1, C_2, and C_3 represent the carry digits for the three columns from right to left.

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.

CONSTRAINT PROPAGATION: INFERENCE IS CSP

An atomic state-space search algorithm makes progress in only one way:

- by expanding a node to visit the successors.

A CSP algorithm has choices. It can generate successors by

- choosing a new variable assignment,
- or it can do a specific type of inference called constraint propagation

constraint propagation:

- using the constraints to reduce the number of legal values for a variable, which in turn can reduce the legal values for another variable, and so on.

CONSTRAINT PROPAGATION: 1. NODE CONSISTENCY

A single variable (corresponding to a node in the CSP graph) is node-consistent if all the values in the variable's domain satisfy the variable's unary constraints.

For example, in the variant of the Australia map-coloring problem where South Australians dislike green, the variable SA starts with domain

{red;green;blue}

, and we can make it node consistent by eliminating green, leaving SA with the reduced domain {red;blue}

A graph is node-consistent if every variable in the graph is node-consistent.

CONSTRAINT PROPAGATION: 2. ARC CONSISTENCY

A variable is arc-consistent if every value in its domain satisfies the variable's binary constraints.

Xi is arc-consistent with respect to another variable Xj if for every value in the current domain Di there is some value in the domain Dj that satisfies the binary constraint on the arc (Xi;Xj).

A graph is arc-consistent if every variable is arc-consistent with every other variable.

$$Y = X^2, \qquad \langle (X, Y), \{(0,0), (1,1), (2,4), (3,9)\} \rangle .$$

To make X arc-consistent with respect to Y, we reduce X's domain to {0;1;2;3}.

- If we also make Y arc-consistent with respect to X, then Y's domain becomes {0;1;4;9}.

CONSTRAINT PROPAGATION: 3. PATH CONSISTENCY

Path consistency tightens the binary constraints by using implicit constraints that are inferred by looking at triples of variables.

A two-variable set {*Xi;Xj*} is path-consistent with respect to a third variable *Xm* if,

- For every assignment {Xi = a;Xj = b} consistent with the constraints (if any) on {Xi;Xj},
- there is an assignment to Xm that satisfies the constraints on {Xi;Xm} and {Xm;Xj}.

The name refers to the overall consistency of the path from *Xi to Xj* with *Xm* in the middle

CONSTRAINT PROPAGATION: 4. K- CONSISTENCY

A CSP is strongly ***k-consistent*** if it is *k-consistent* and is also *(k-1)-consistent*, *(k-2)-consistent*, … all the way down to *1-consistent*.

SUDOKU

A Sudoku board consists of 81 squares,

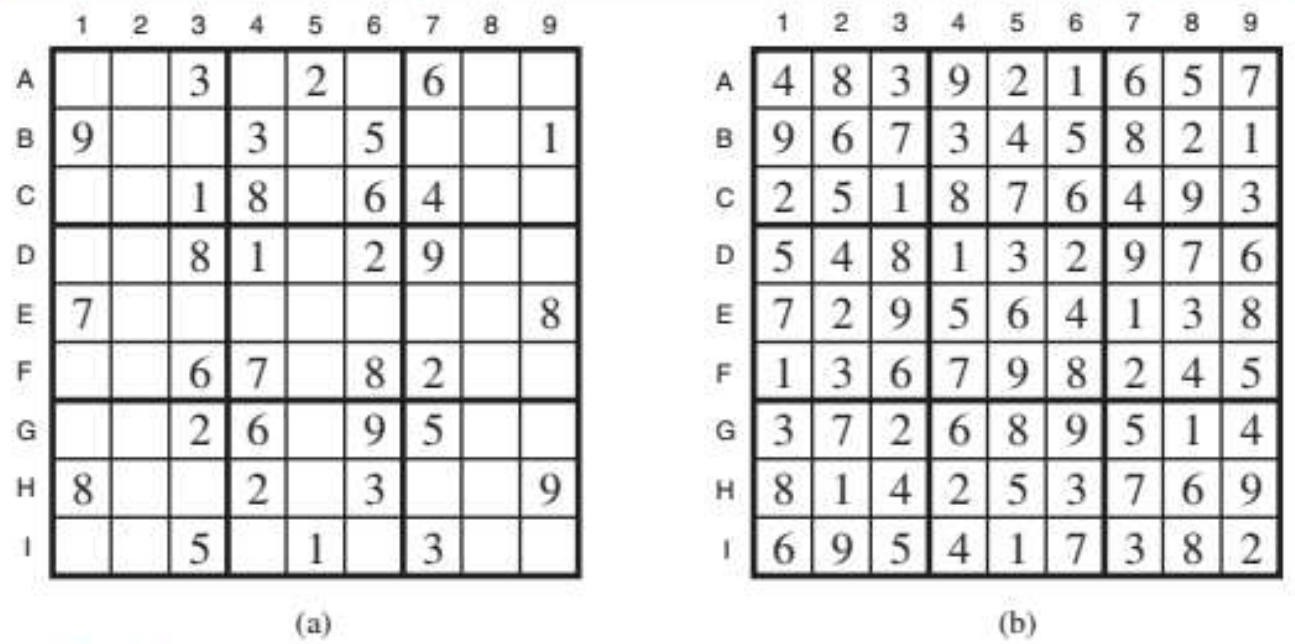
- some of which are initially filled with digits from 1 to 9.

The puzzle is to fill in all the remaining squares

- such that no digit appears twice in any row, column, or 3×3 box

A row, column, or box is called a unit

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



	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

Figure 5.4 (a) A Sudoku puzzle and (b) its solution.

Variables?
Domain?

Constraints?

SUDOKU

$$
\begin{aligned}
&Alldiff(A1, A2, A3, A4, A5, A6, A7, A8, A9)\\
&Alldiff(B1, B2, B3, B4, B5, B6, B7, B8, B9)\\
&\ldots\\
&Alldiff(A1, B1, C1, D1, E1, F1, G1, H1, I1)\\
&Alldiff(A2, B2, C2, D2, E2, F2, G2, H2, I2)\\
&\ldots\\
&Alldiff(A1, A2, A3, B1, B2, B3, C1, C2, C3)\\
&Alldiff(A4, A5, A6, B4, B5, B6, C4, C5, C6)\\
&\ldots
\end{aligned}
$$

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

Node consistency?

Arc consistency?

Path consistency?

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

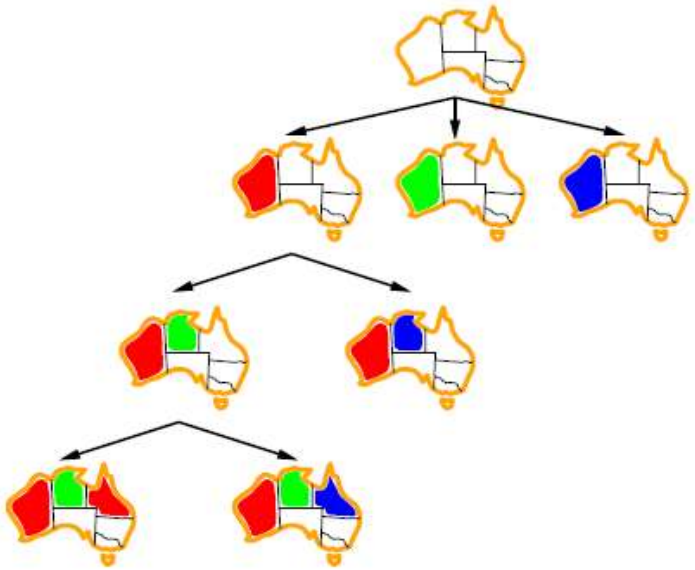
Backtracking search is the basic uninformed algorithm for CSPs

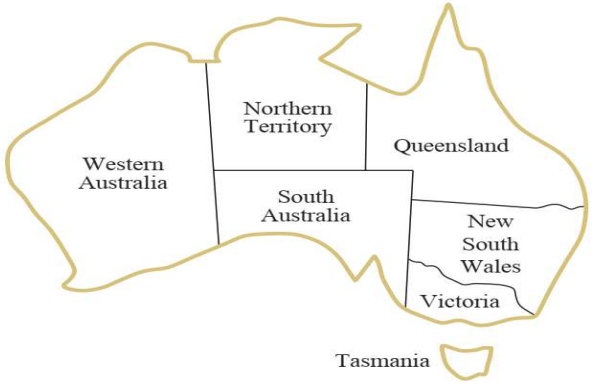
Can solve n-queens for n ~ 25

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



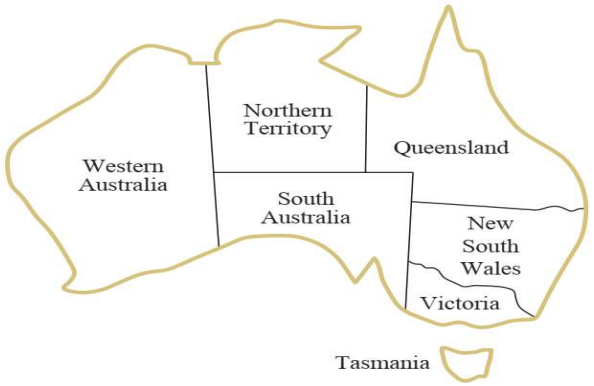


IMPROVING BACKTRACKING EFFICIENCY

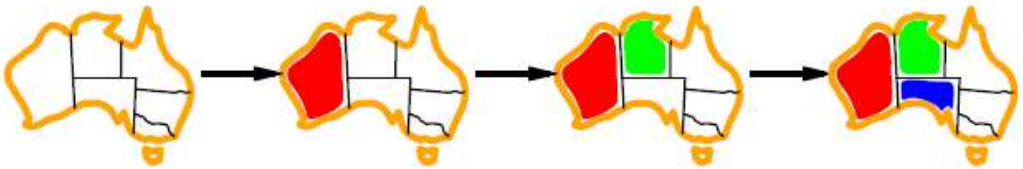
General-purpose methods can give huge gains in speed:

- 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?

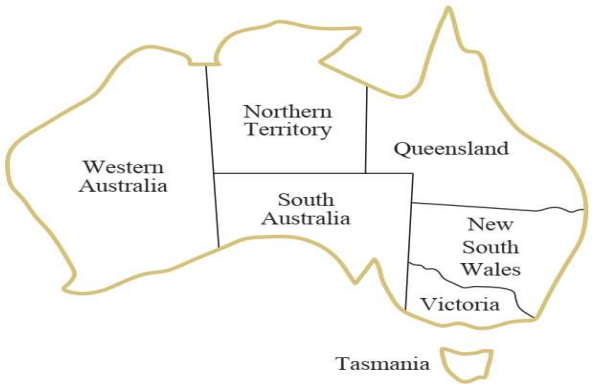
1. MINIMUM REMAINING VALUES



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

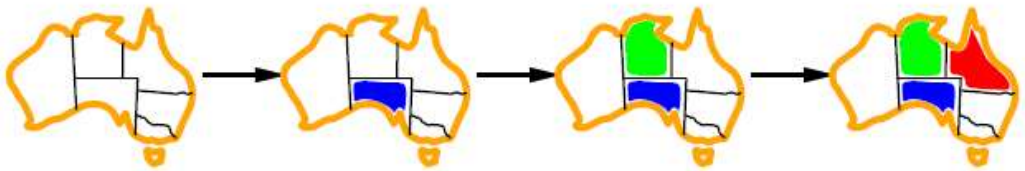


2. DEGREE HEURISTIC

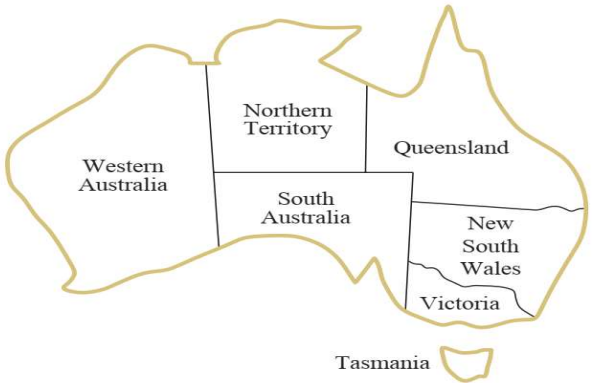


Tie-breaker among MRV variables

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

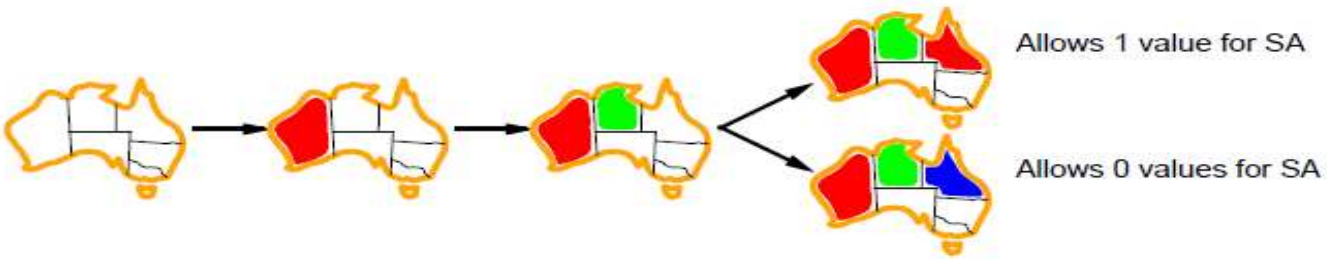


3. LEAST CONSTRAINING VALUE

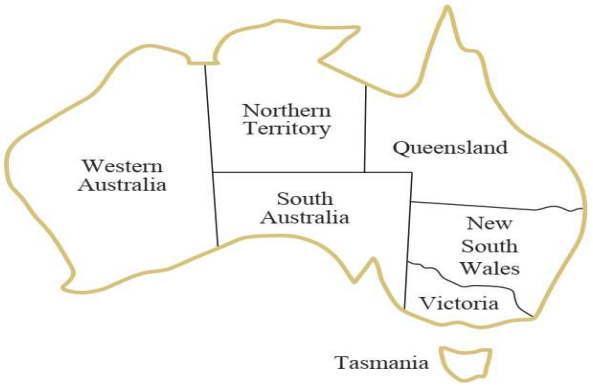


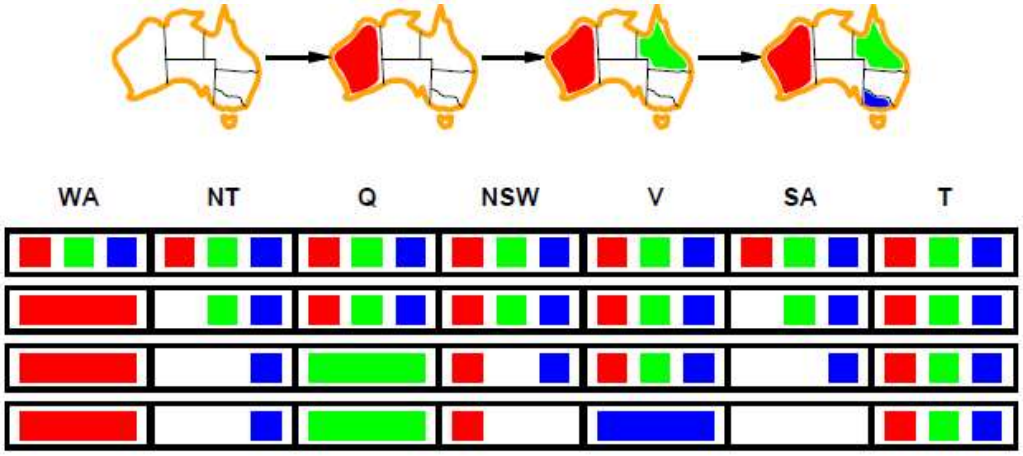
Given a variable, choose the least constraining value: the one that rules out the fewest values in the remaining variables.

Combining these heuristics makes 1000 queens feasible



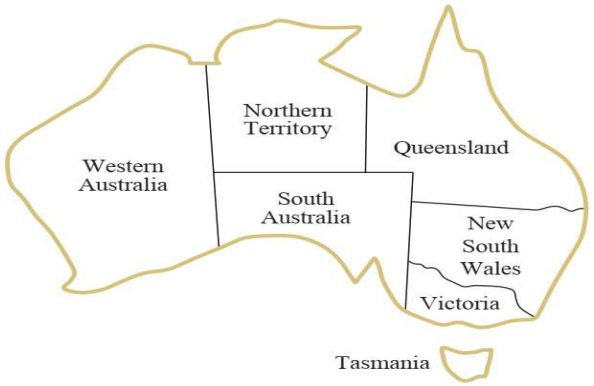
FORWARD CHECKING



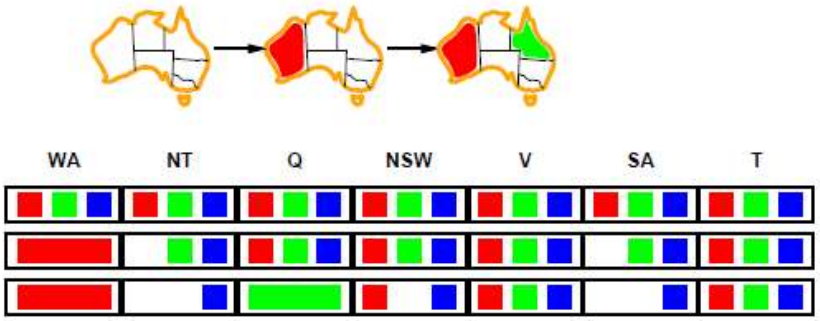


Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

CONSTRAINT PROPAGATION



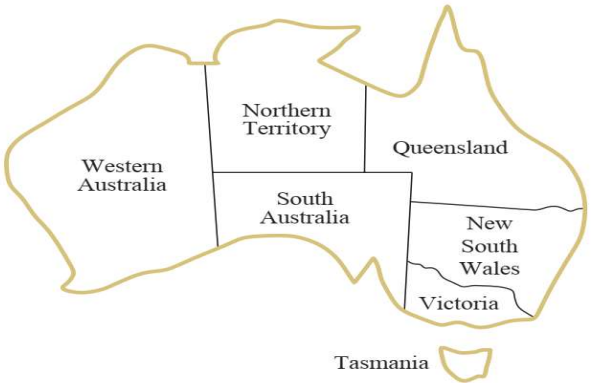
Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



NT and SA cannot both be blue!

Constraint propagation repeatedly enforces constraints locally

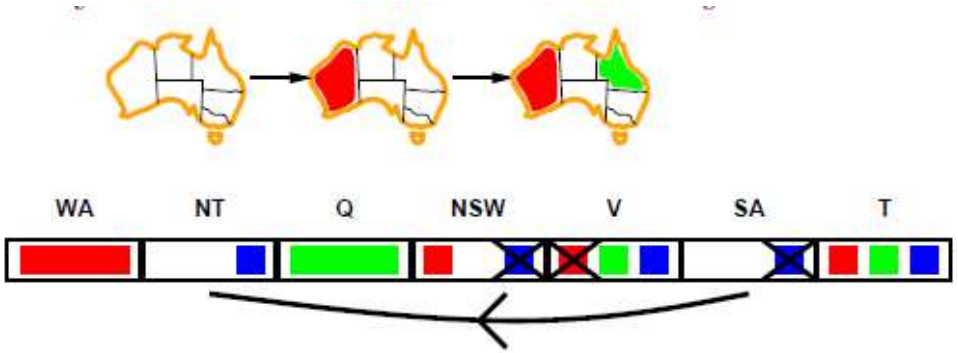
ARC CONSISTENCY



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

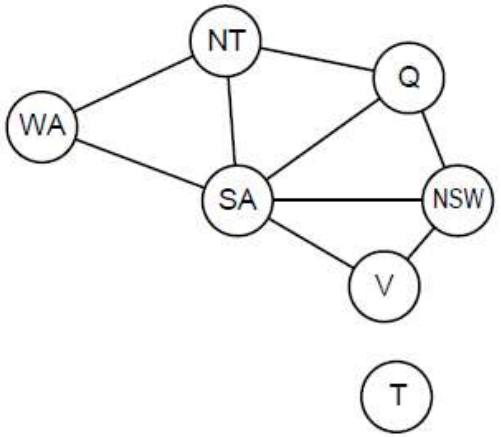
Can be run as a preprocessor or after each assignment

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, ..., 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) ← 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 succeeds
  removed ← 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 ↔ X_j
      then delete x from DOMAIN[X_i]; removed ← true
  return removed
```

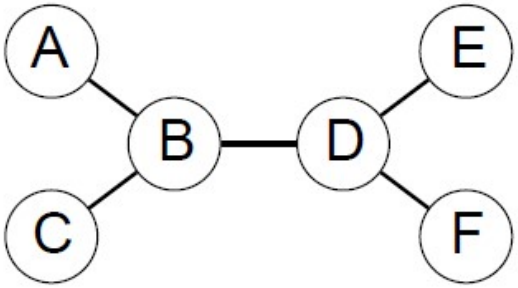
$O(n^2d^3)$, can be reduced to $O(n^2d^2)$ (but detecting all is NP-hard)

PROBLEM STRUCTURE



Tasmania and mainland are independent subproblems
Identifiable as connected components of constraint graph

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 $O(d^n)$

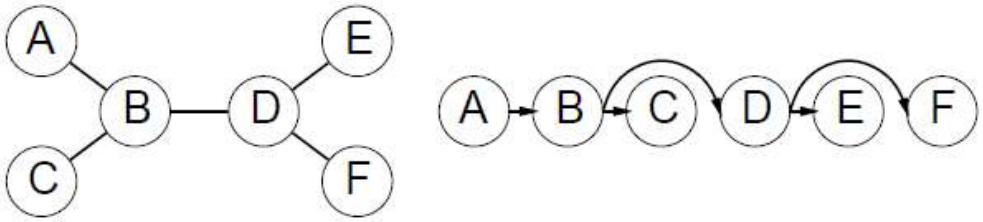
This property also applies to logical and probabilistic reasoning: an important example of the relation between syntactic restrictions and the complexity of reasoning.

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.

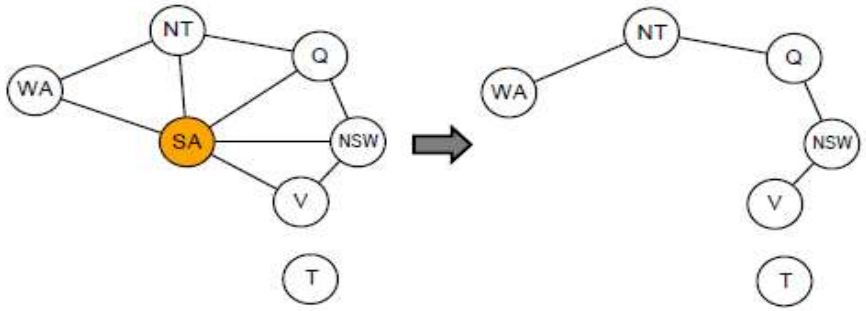
2. For j from n down to 2, apply Remove Inconsistent(Parent(Xj);Xj)

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



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 → 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 ($44 = 256$ states)

Operators: move queen in column

Goal test: no attacks

Evaluation: h(n) = number of attacks

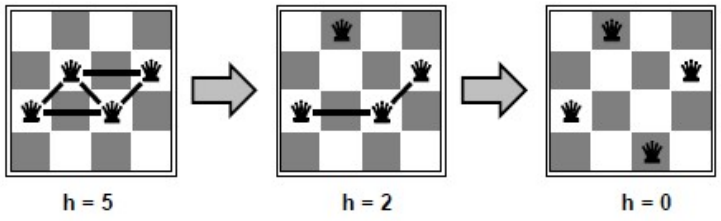


Figure 5.8 A two-step solution using min-conflicts for an 8-queens problem. At each stage, a queen is chosen for reassignment in its column. The number of conflicts (in this case, the number of attacking queens) is shown in each square. The algorithm moves the queen to the min-conflicts square, breaking ties randomly.

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