Introduction to Artificial Intelligence

Constraint Satisfaction Problems

Dr. Tapas Kumar Mishra
SRM University, AP

Introduction

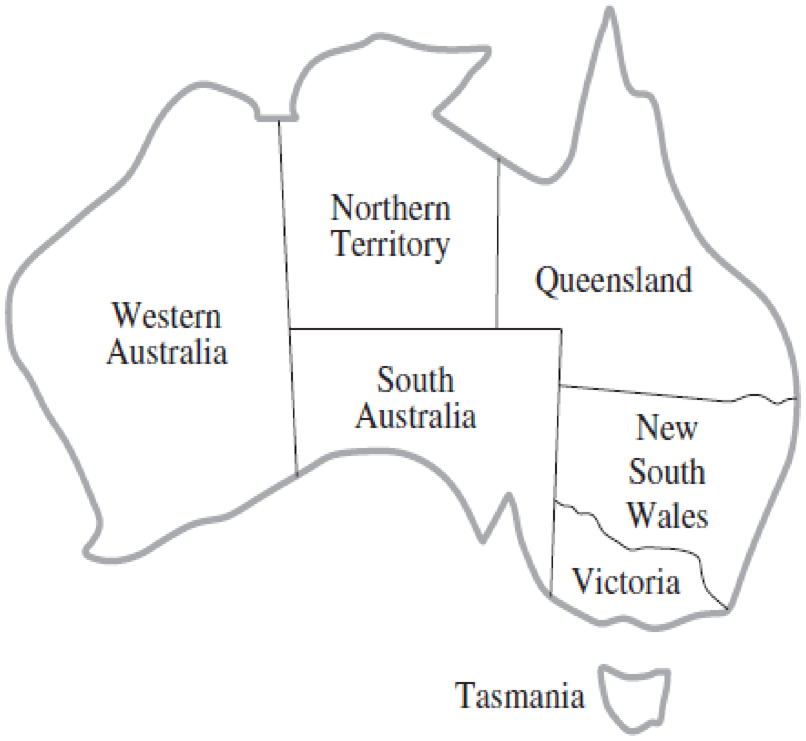
We use a factored representation for each state: a set of variables, each of which has a value. A problem is solved when each variable has a value that satisfies all the constraints on the variable. A problem described this way is called a constraint satisfaction problem.

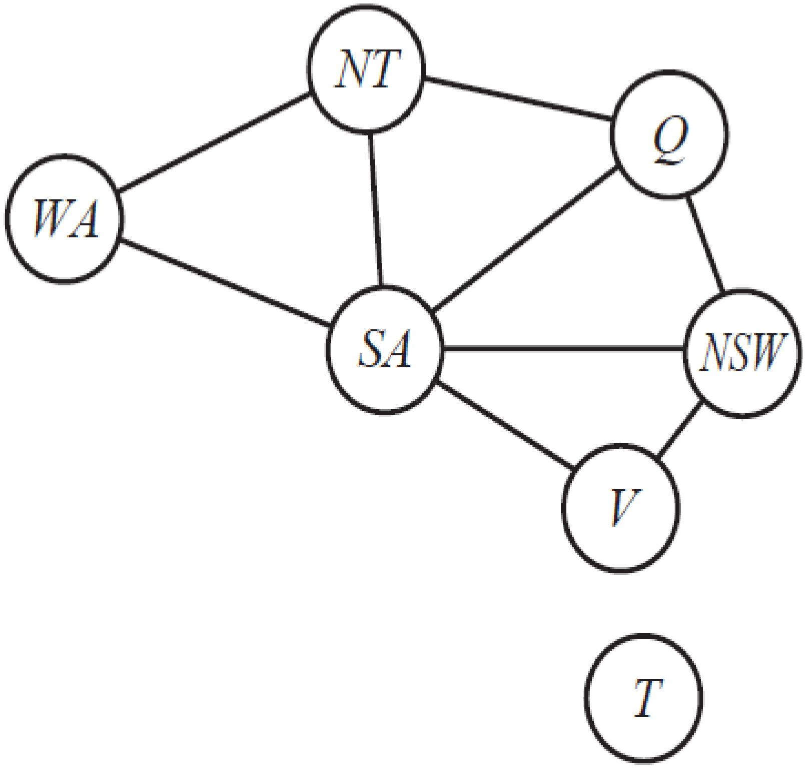
It has three components, X,D, and C:

- X is a set of variables, $\{X_1, \ldots, Xn\}$.
- D is a set of domains, $\{D_1, \ldots, Dn\}$, one for each variable.
- C is a set of constraints that specify allowable combinations of values.

Example problem: Graph coloring

Fig: 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.





- X = {WA,NT,Q,NSW, V, SA, T} .
- C = {SA ≠ WA, SA ≠ NT, SA ≠ Q, SA ≠ NSW, SA ≠ V, WA ≠ NT,NT ≠ Q,Q ≠ NSW,NSW ≠ V } .
- D = {red, green, blue}.

Example problem: Job-shop scheduling

Consider the problem of scheduling the assembly of a car. The whole job is composed of tasks, and we can model each task as a variable, where the value of each variable is the time that the task starts, expressed as an integer number of minutes. Constraints can assert that one task must occur before another

- X = {AxleF, AxleB, WheelRFp, WheelLF, WheelRB, WheelLB, NutsRF, NutsLF, NutsRB, NutsLB, CapRF, CapLF, CapRB, CapLB, Inspect} .

The value of each variable is the time that the task starts.

Constraint (C) is in format: T1 + d1 ≤ T2 . (PRECEDENCE CONSTRAINTS)

- C= {AxleF + 10 ≤ WheelRF ; AxleF + 10 ≤ WheelLF ; AxleB + 10 ≤ WheelRB; AxleB + 10 ≤ WheelLB ;

WheelRF + 1 ≤ NutsRF ; NutsRF + 2 ≤ CapRF ; WheelLF + 1 ≤ NutsLF ; NutsLF + 2 ≤ CapLF ; WheelRB + 1 ≤ NutsRB; NutsRB + 2 ≤ CapRB; WheelLB + 1 ≤ NutsLB; NutsLB + 2 ≤ CapLB }

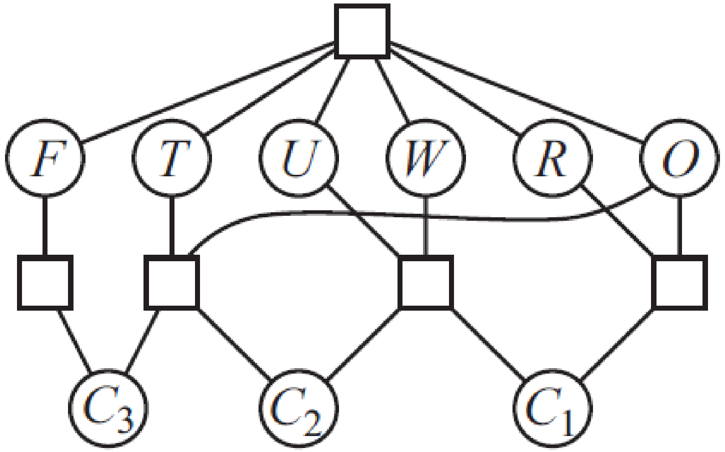
- D = {1, 2, 3, . . . , 27} .

Disjunctive constraint: (AxleF + 10 ≤ AxleB) **or** (AxleB + 10 ≤ AxleF) .

Example problem: cryptarithmetic puzzles.

Each letter in a cryptarithmetic puzzle represents a different digit.
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.

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



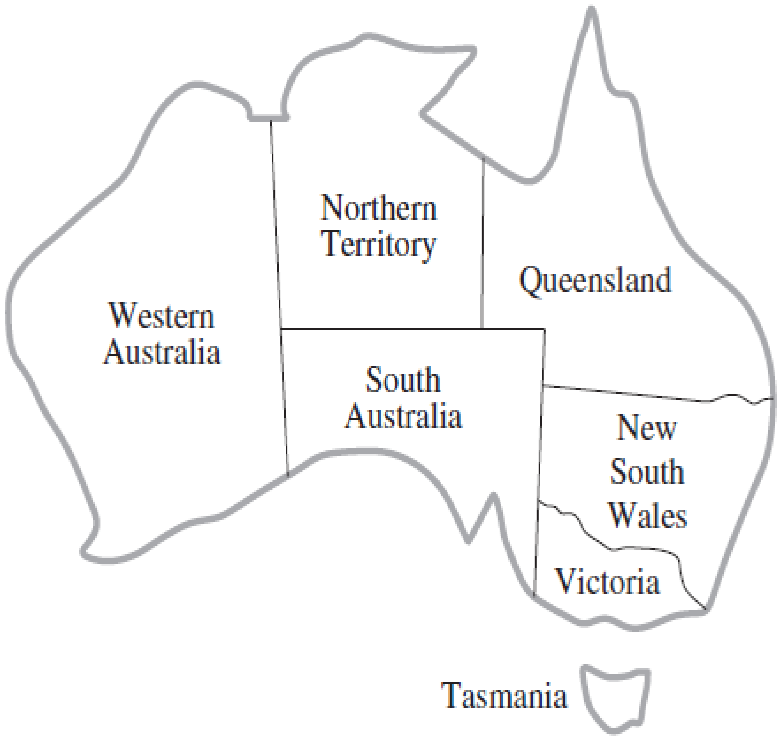
CONSTRAINT={
O + O = R + 10 ▪ C10
C10 +W +W = U + 10 ▪ C100
C100 + T + T = O + 10 ▪ C1000
C1000 = F
}

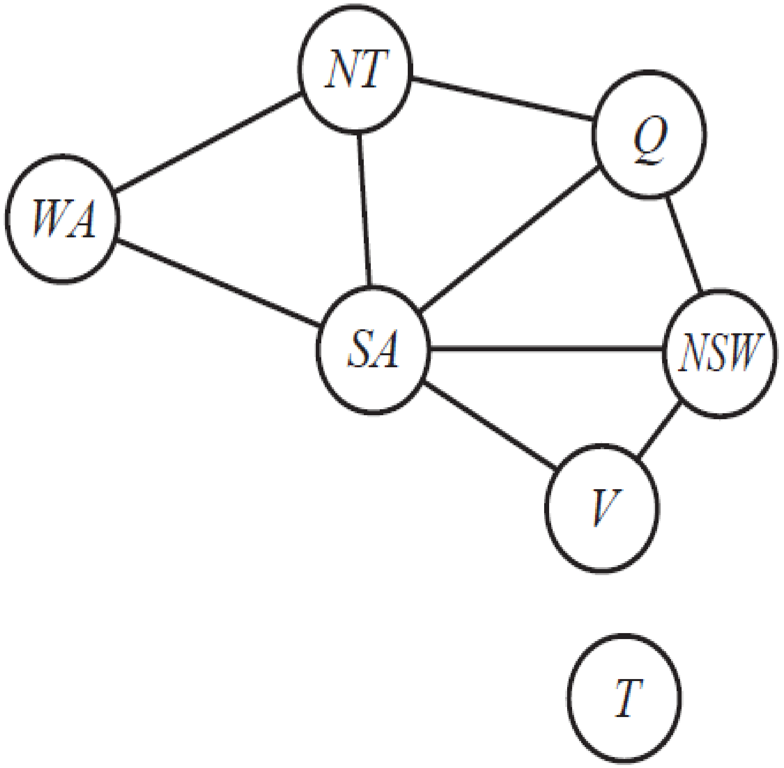
X= {F, O, U, R, T, W, C1, C2, C3}

D= { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

CONSTRAINT PROPAGATION: INFERENCE IN CSPS

Node consistency: A local constraint

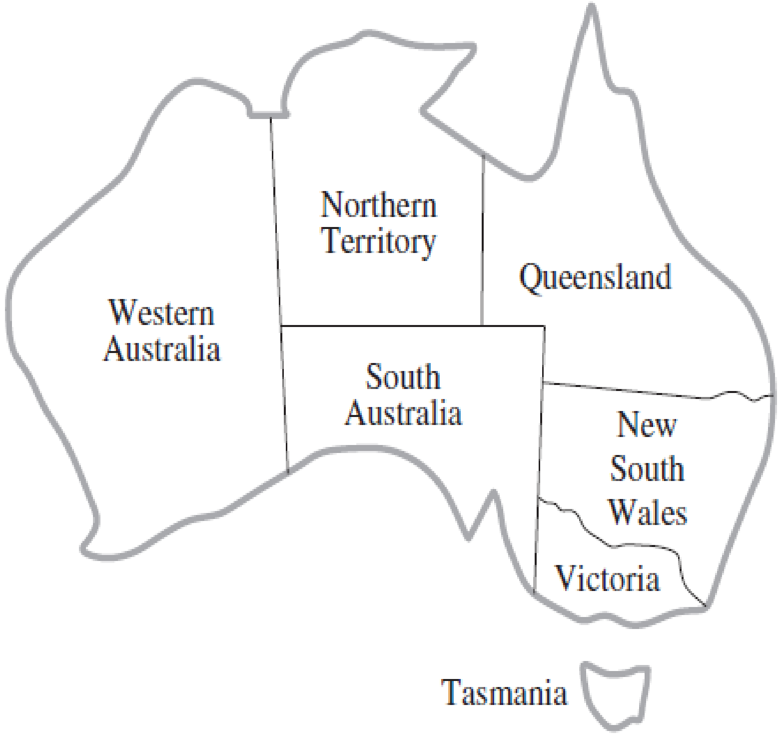


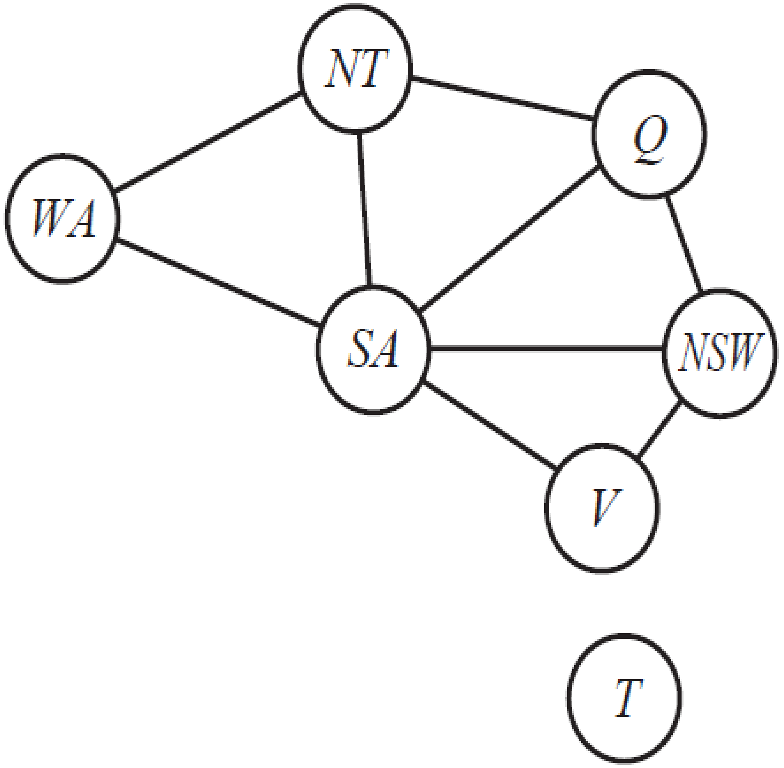


- if all the values in the variable's domain satisfy the variable's unary constraints. For example, in the variant of the Australia map-coloring problem where South Australians dislike green, the variable SA starts with domain {red, green, blue}, and we can make it node consistent by eliminating green, leaving SA with the reduced domain {red, blue}.
- We say that a network is node-consistent if every variable in the network is node-consistent.

CONSTRAINT PROPAGATION: INFERENCE IN CSPS

Arc consistency: A local constraint





- if every value in its domain satisfies the variable's binary constraints. More formally, X_i is arc-consistent with respect to another variable X_j 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 (X_i,X_j).
- A network is arc-consistent if every variable is arc consistent with every other variable.

CONSTRAINT PROPAGATION: INFERENCE IN CSPS

Arc consistency: A local constraint

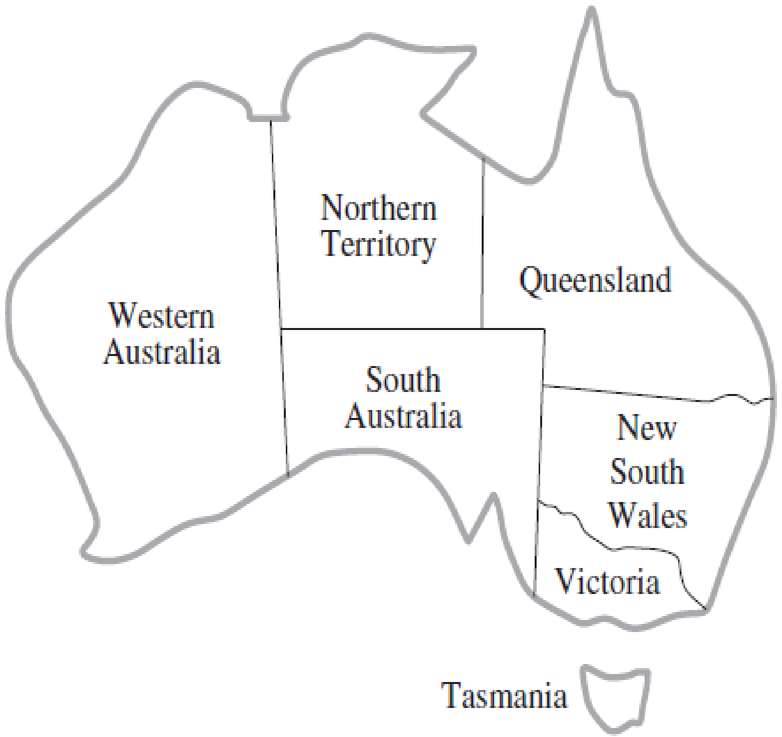
```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components (X, D, C)
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if REVISE(csp, X_i, X_j) then
      if size of D_i = 0 then return false
      for each X_k in X_i.NEIGHBORS - {X_j} do
        add (X_k, X_i) to queue
  return true
```

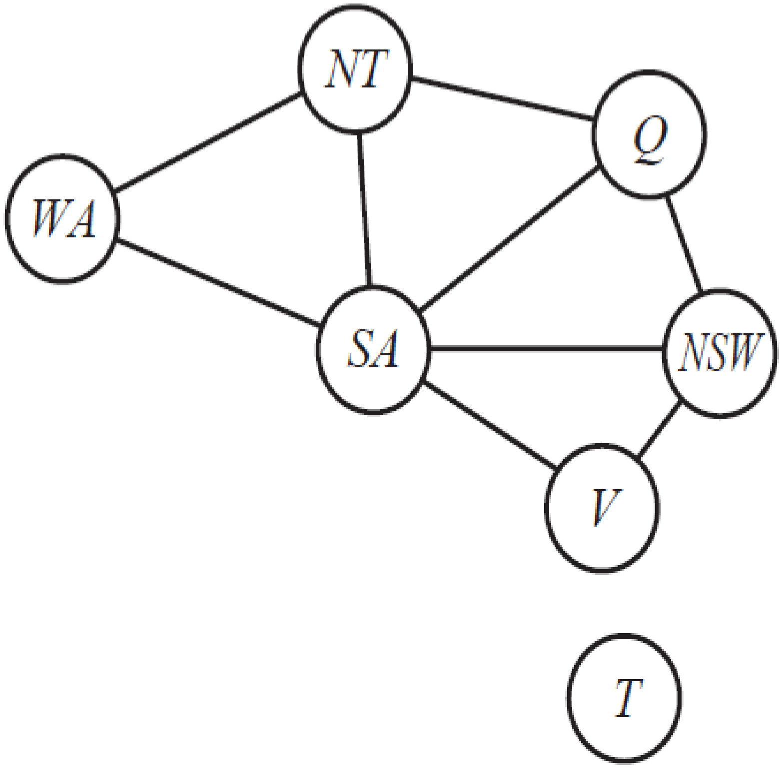
```
function REVISE(csp, X_i, X_j) returns true iff we revise the domain of X_i
  revised ← false
  for each x in D_i do
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j then
      delete x from D_i
      revised ← true
  return revised
```

Fig: The arc-consistency algorithm AC-3.

CONSTRAINT PROPAGATION: INFERENCE IN CSPS

Path consistency: A local constraint

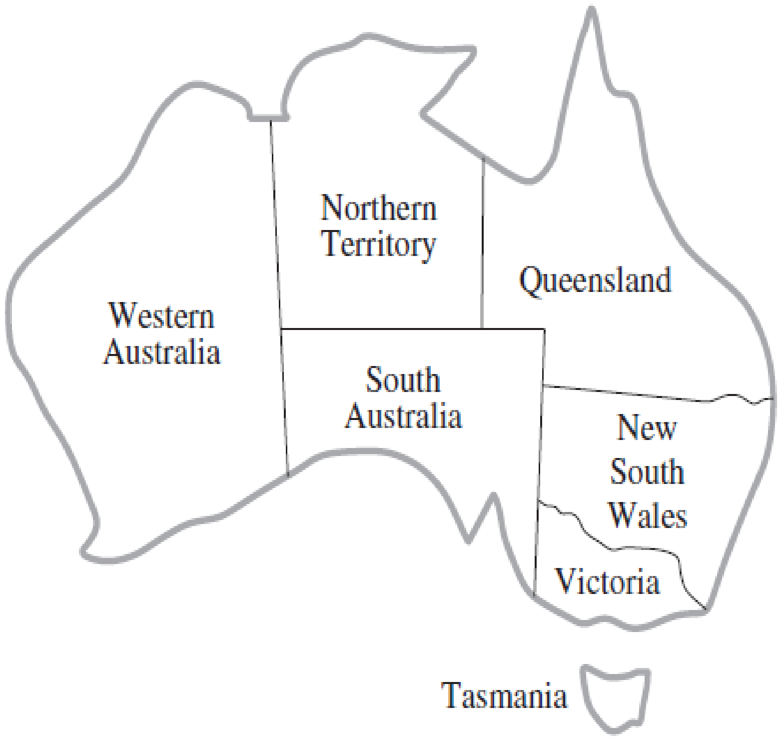


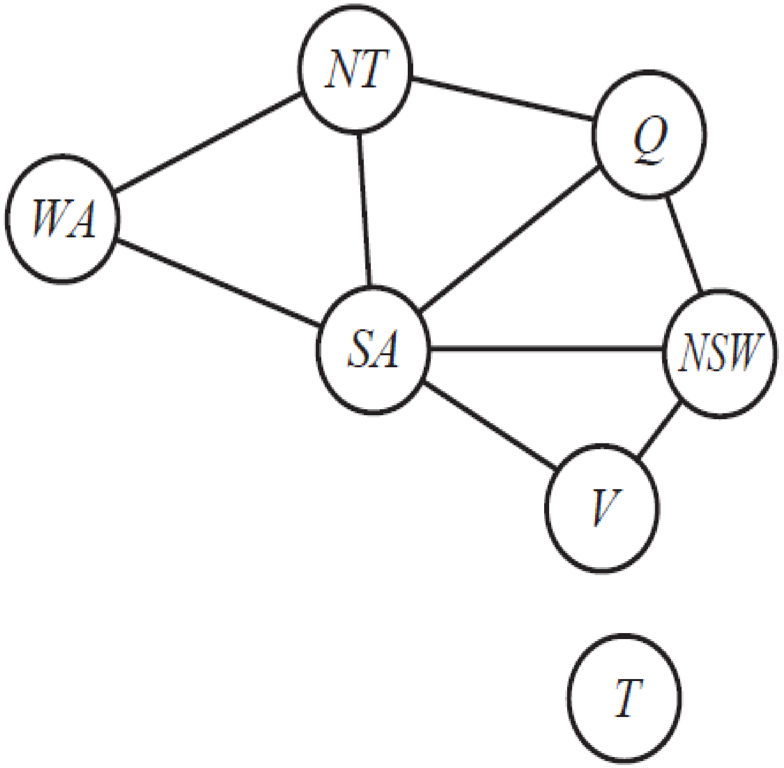


Path consistency tightens the binary constraints by using implicit constraints that are inferred by looking at triples of variables.
A two-variable set {Xi,Xj} is path-consistent with respect to a third variable Xm if, for every assignment {Xi = a,Xj = b} consistent with the constraints on {Xi,Xj}, there is an assignment to Xm that satisfies the constraints on {Xi,Xm} and {Xm,Xj}.

CONSTRAINT PROPAGATION: INFERENCE IN CSPS

K-consistency: A local constraint

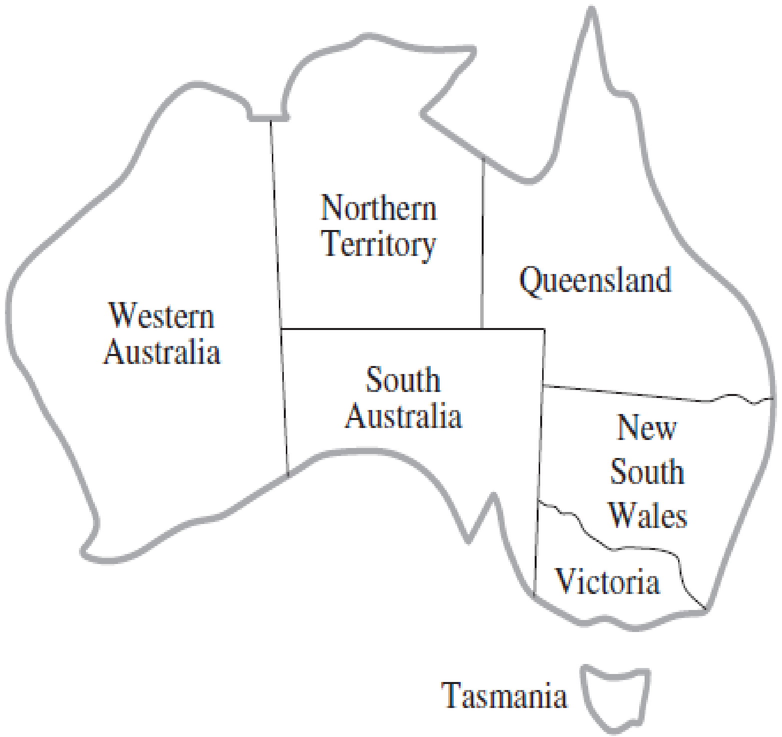


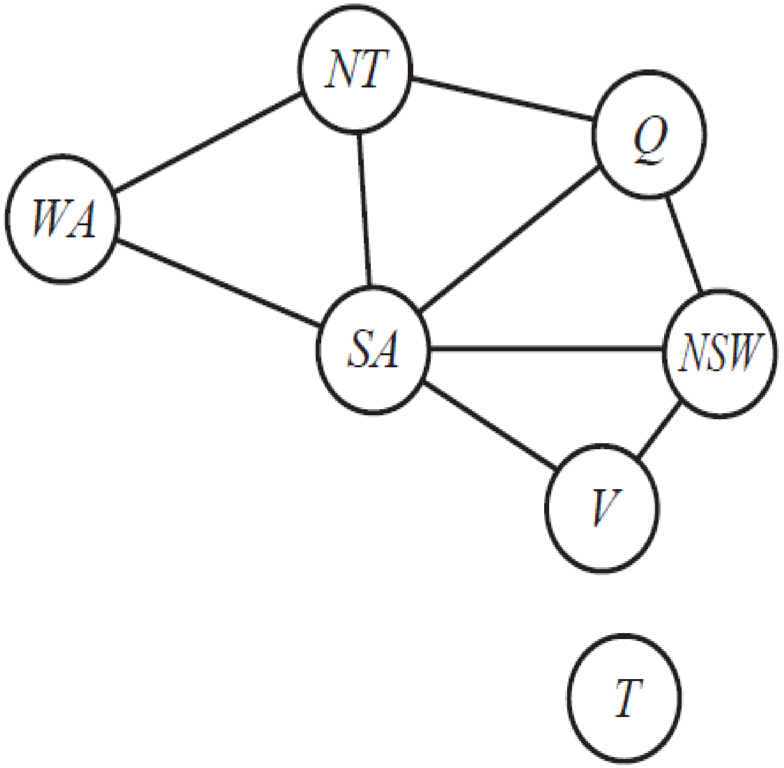


A CSP is **k-consistent** if, for any set of k − 1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to any k^{th} variable.

CONSTRAINT PROPAGATION: INFERENCE IN CSPS

bounds propagation:
A Global constraint





D1 = [0, 165] and D2 = [0, 385] .

F1 + F2 = 420. (Additional constraint). ➔ D1 = [35, 165] and D2 = [255, 385] .

We say that a CSP is **bounds consistent** if for every variable X, and for both the lower- bound and upper-bound values of X, there exists some value of Y that satisfies the constraint between X and Y for every variable Y

CSP: Example

A Sudoku puzzle

	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)

Fig: (a) A Sudoku puzzle and (b) its solution.

BACKTRACKING SEARCH FOR CSPs

A Sudoku puzzle

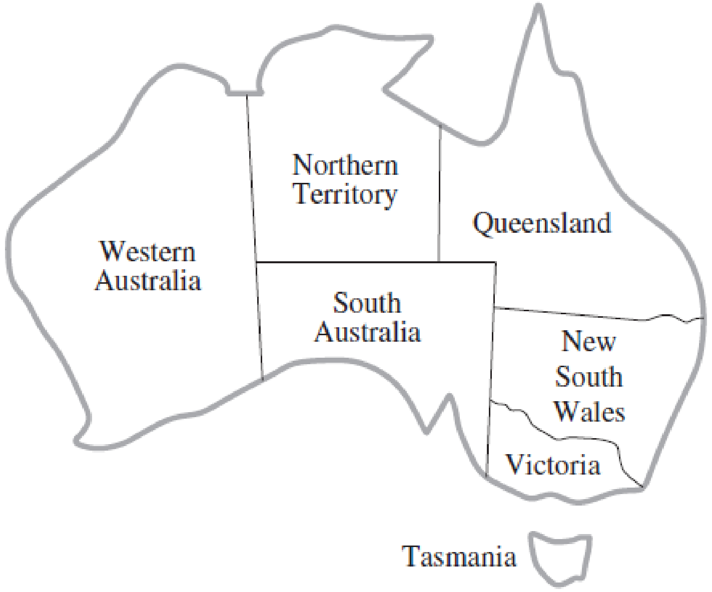
	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

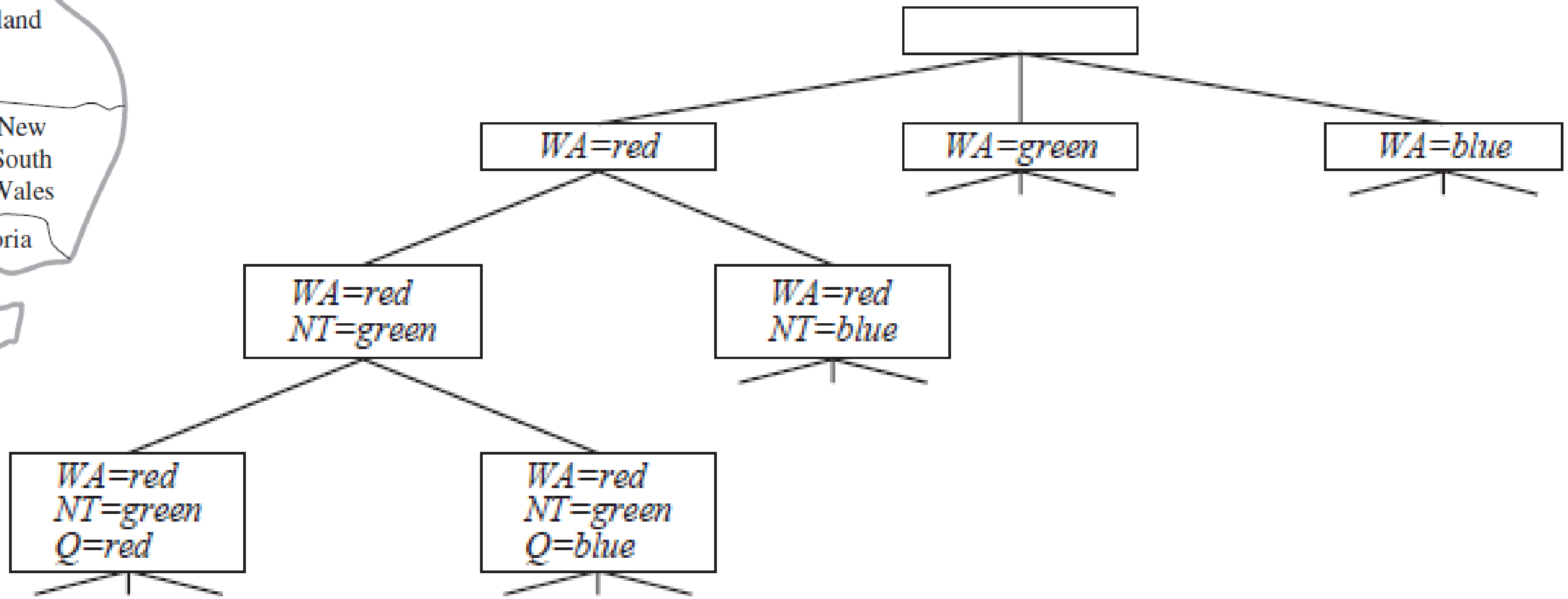
function BACKTRACKING-SEARCH(*csp*) **returns** a solution, or failure
    return BACKTRACK({ }, *csp*)

function BACKTRACK(*assignment*, *csp*) **returns** a solution, or failure
    if *assignment* is complete **then return** *assignment*
    var ← SELECT-UNASSIGNED-VARIABLE(*csp*)
    for each *value* **in** ORDER-DOMAIN-VALUES(*var*, *assignment*, *csp*) **do**
        if *value* is consistent with *assignment* **then**
            add {*var* = *value*} to *assignment*
            inferences ← INFERENCE(*csp*, *var*, *value*)
            if *inferences* ≠ *failure* **then**
                add *inferences* to *assignment*
                result ← BACKTRACK(*assignment*, *csp*)
                if *result* ≠ *failure* **then**
                    return *result*
        remove {*var* = *value*} and *inferences* from *assignment*
    return *failure*

BACKTRACKING SEARCH FOR CSPs

A Map coloring problem





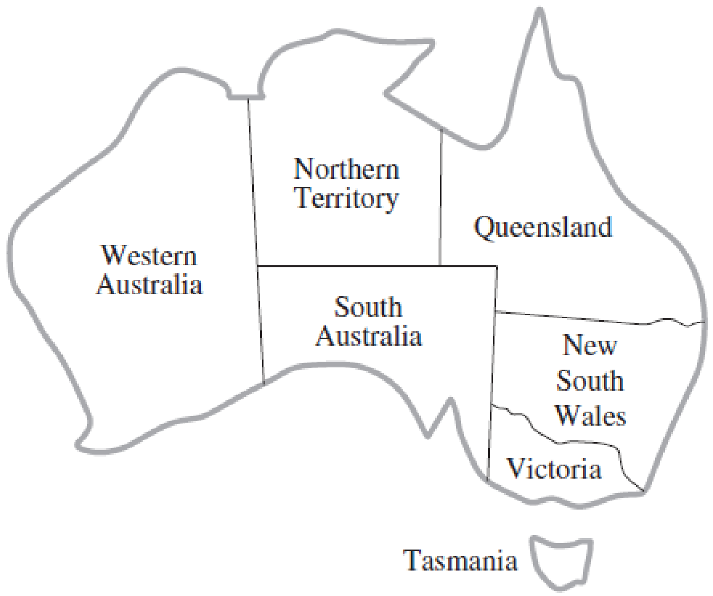
Variable and value ordering: var ←SELECT-UNASSIGNED-VARIABLE(csp) .

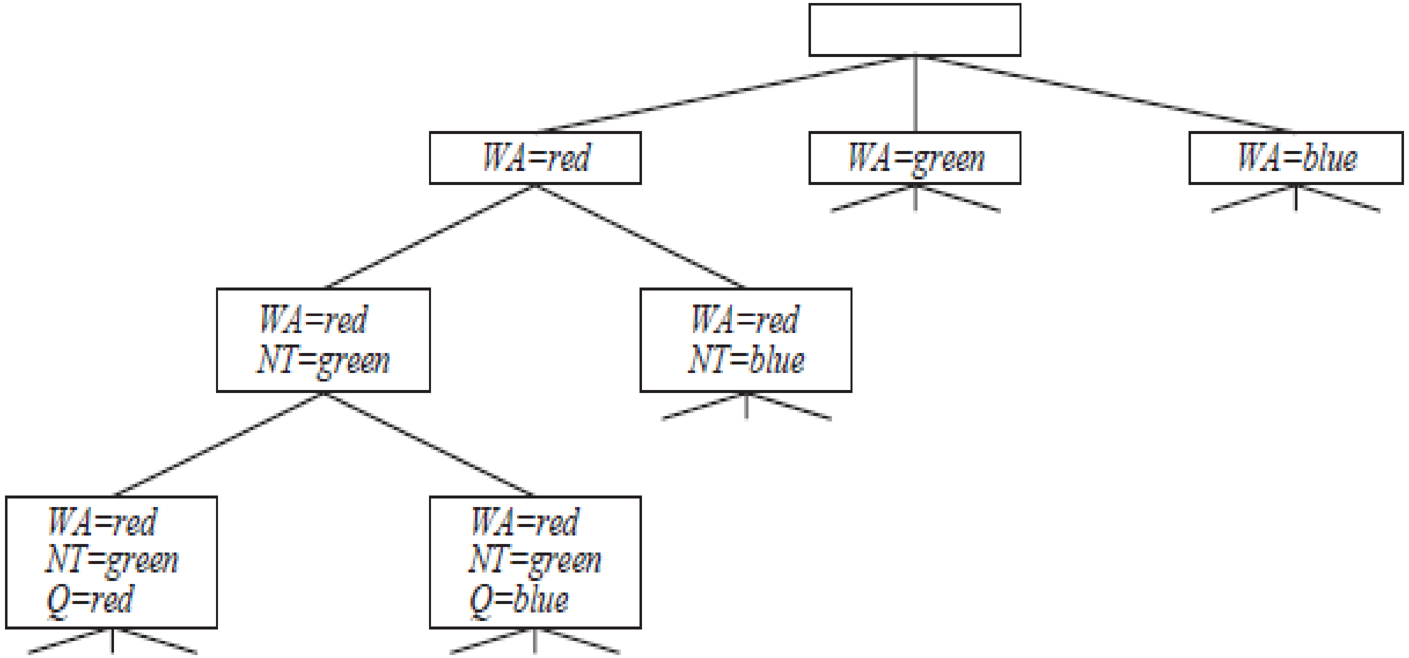
- MINIMUMREMAININGVALUES:
- DEGREE HEURISTIC

BACKTRACKING SEARCH FOR CSPs

Interleaving search and inference: Intelligent backtracking: Looking backward

A Map coloring problem





	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	R	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	R	B	G	R B	R G B	B	R G B
After *V=blue*	R	B	G	R	B		R G B

LOCAL SEARCH FOR CSPs: Topological Sort

A Map coloring problem

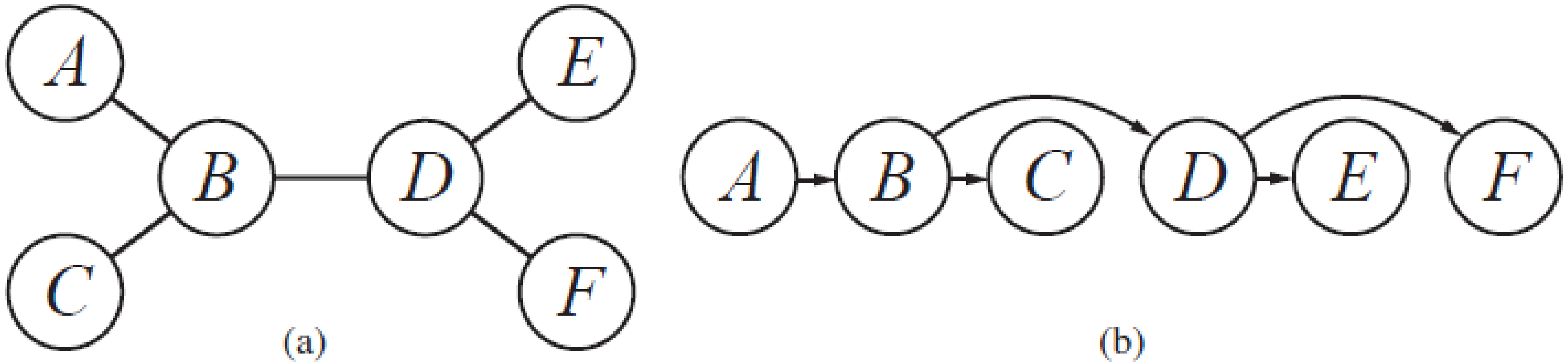


Fig: (a) The constraint graph of a tree-structured CSP. (b) A linear ordering of the variables consistent with the tree with A as the root. This is known as a **topological sort** of the variables.