CONSTRAINT SATISFACTION PROBLEMS

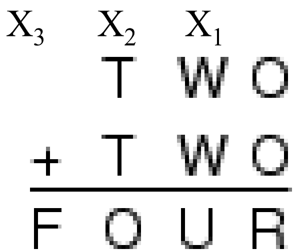
Constraint Satisfaction Problems

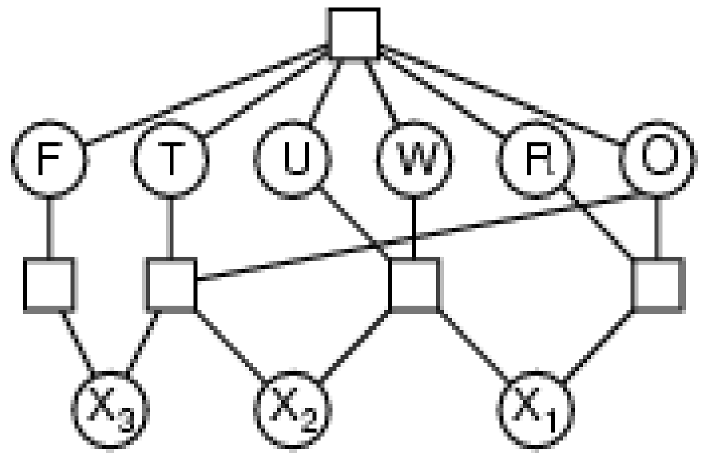
A CSP consists of:

- *Finite set of variables* $X_1, X_2, \ldots, X_n$
- *Nonempty* ***domain*** *of possible values* for each variable $D_1, D_2, \ldots D_n$ *where* $D_i = \{v_1, \ldots, v_k\}$
- *Finite set of constraints* $C_1, C_2, \ldots, C_m$
  - Each *constraint* C_i limits the values that variables can take, e.g., $X_1 \neq X_2$ A *state* is defined as an *assignment* of values to some or all variables.

- **A *consistent assignment* does not violate the constraints.**

Example: Cryptarithmetic



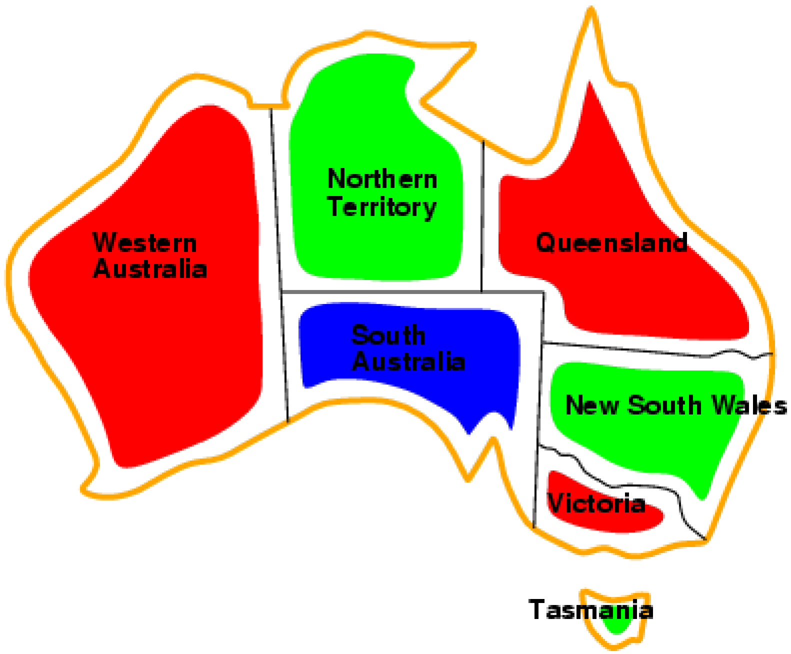


- **Variables**: ***F T U W R O, X_1 X_2 X_3***
- **Domain**: ***{0,1,2,3,4,5,6,7,8,9}***
- **Constraints**:
  - *Alldiff (F,T,U,W,R,O)*
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F$, $T \neq 0$, $F \neq 0$

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 ≠ NT
    - —So (WA,NT) must be in {(red,green),(red,blue),(green,red), …}

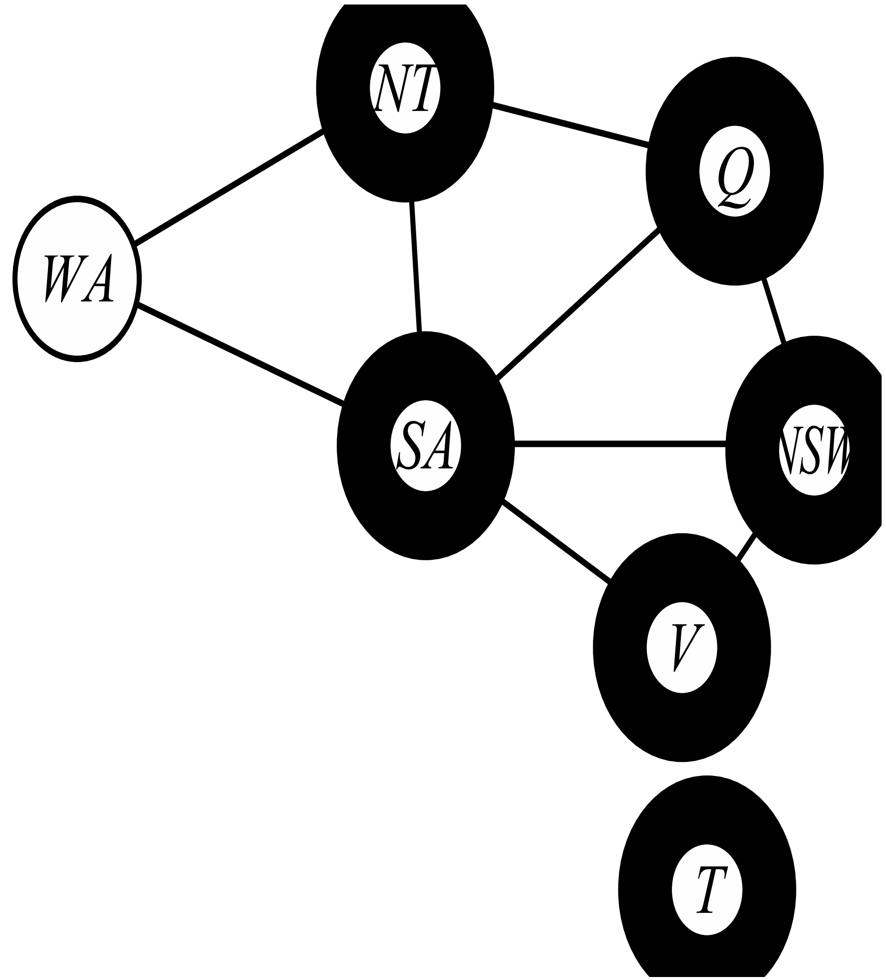
Example: Map-coloring



Solutions are complete and consistent assignments,

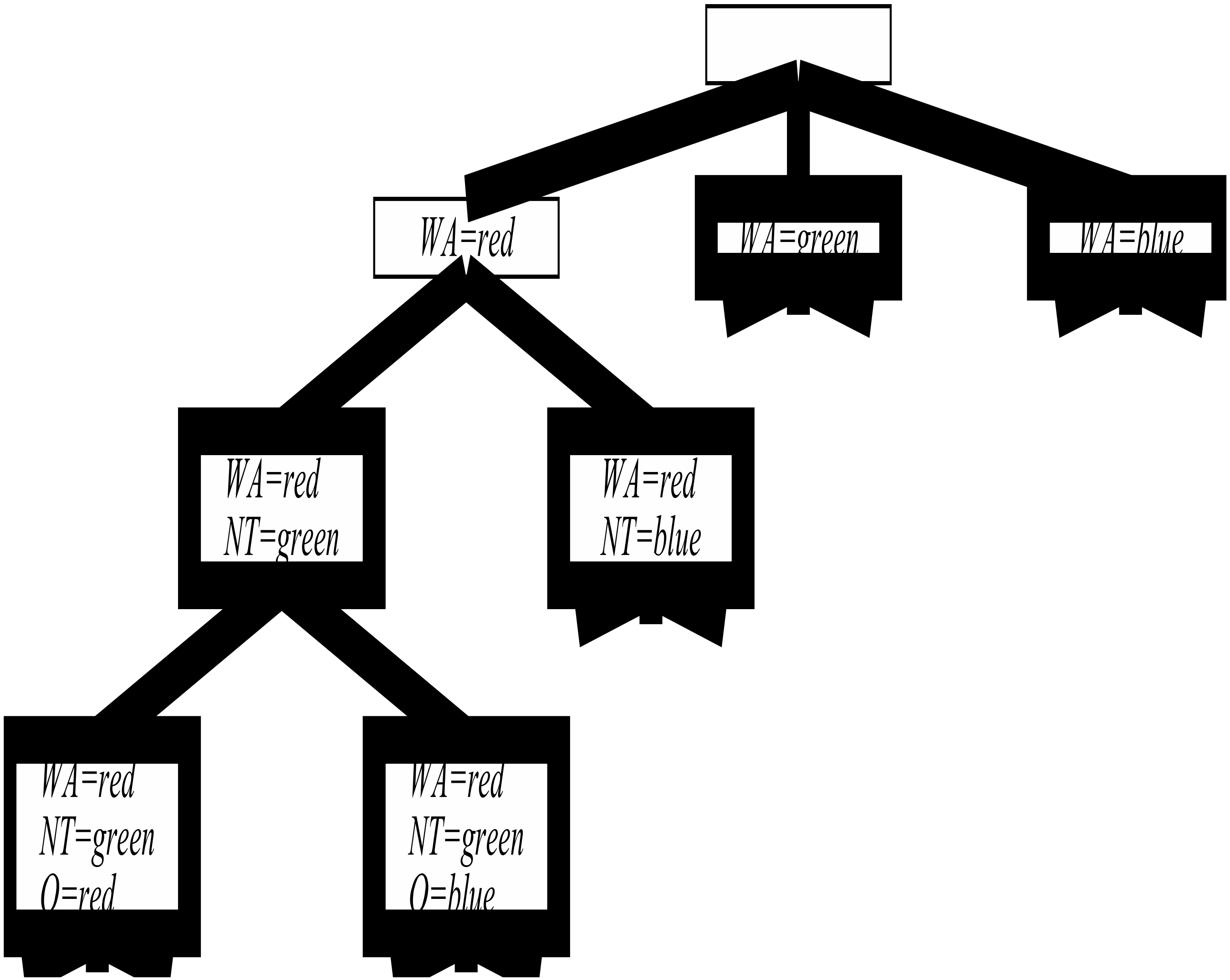
- e.g., WA = red, NT = green,Q = red,NSW = green,
  V = red,SA = blue,T = green


Western Australia
Northern Territory
Queensland
South Australia
New South Wales
Victoria
Tasmania


NT
Q
WA
SA
NSW
V
T

Graph Coloring

Adjacency Matrix


WA=red
WA=green
WA=blue
WA=red
NT=green
WA=red
NT=blue
WA=red
NT=green
Q=red
WA=red
NT=green
Q=blue

Benefits of CSP

- **Clean specification of many problems, generic goal, successor function & heuristics**
  - Just represent problem as a CSP & solve with general package
- **CSP "knows" which variables violate a constraint**
  - And hence where to focus the search
- ***CSPs:* Automatically prune off all branches that violate constraints**
  - (State space search could do this only by *hand-building constraints into the successor function)*

Algorithm: Constraint Satisfaction

1. Propagate available constraints. To do this, first set *OPEN* to the set of all objects that must have values assigned to them in a complete solution. Then do until an inconsistency is detected or until *OPEN* is empty:
   (a) Select an object *OB* from *OPEN*. Strengthen as much as possible the set of constraints that apply to *OB*.
   (b) If this set is different from the set that was assigned the last time *OB* was examined or if this is the first time *OB* has been examined, then add to *OPEN* all objects that share any constraints with *OB*.
   (c) Remove *OB* from *OPEN*.
2. If the union of the constraints discovered above defines a solution, then quit and report the solution.
3. If the union of the constraints discovered above defines a contradiction, then return failure.
4. If neither of the above occurs, then it is necessary to make a guess at something in order to proceed. To do this, loop until a solution is found or all possible solutions have been eliminated:
   (a) Select an object whose value is not yet determined and select a way of strengthening the constraints on that object.
   (b) Recursively invoke constraint satisfaction with the current set of constraints augmented by the strengthening constraint just selected.

Sudoku Example

	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		

	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