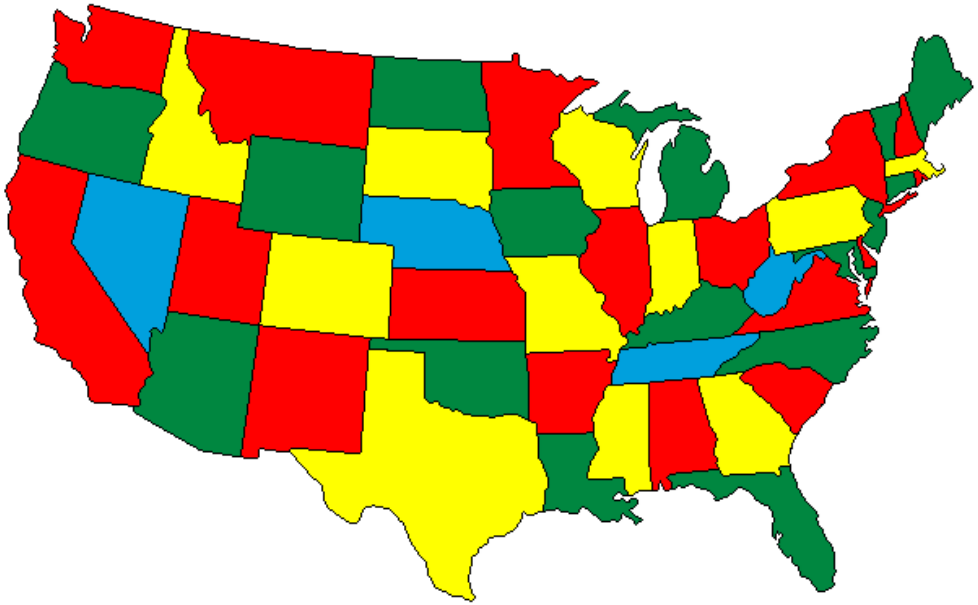
Constraint Satisfaction Problems



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

constraint satisfaction problem

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.

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

Each constraint Ci consists of a pair scope, rel ,

scope :tuple of variables that participate in the constraint.

rel :relation that defines the values that those variables can take on.

A relation can be represented as an explicit list of all tuples of values that satisfy the constraint,

Or abstract relation that supports two operations: testing if a tuple is a member of the relation and enumerating the members of the relation.

- **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
- **Solution** is a complete, consistent assignment

For example, if X1 and X2 both have the domain {A,B}, then the constraint saying the two variables must have different values can be written as (X1,X2), [(A,B), (B,A)] or as (X1,X2),X1 != X2.

Each state in a CSP is defined by an **assignment** of values to some or all of the variables, {Xi =vi,Xj = vj , . . .}.

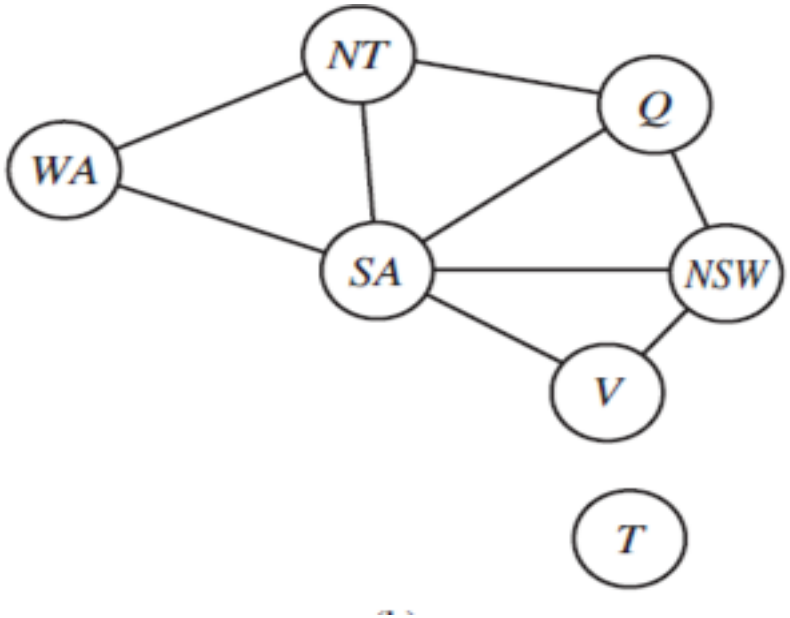
An assignment that does not violate any constraints is called a **consistent** or legal assignment.

A **complete assignment** is one in which every variable is assigned, and

a **solution to a CSP is a consistent, complete assignment.**

A partial assignment is one that assigns values to only some of the variables.

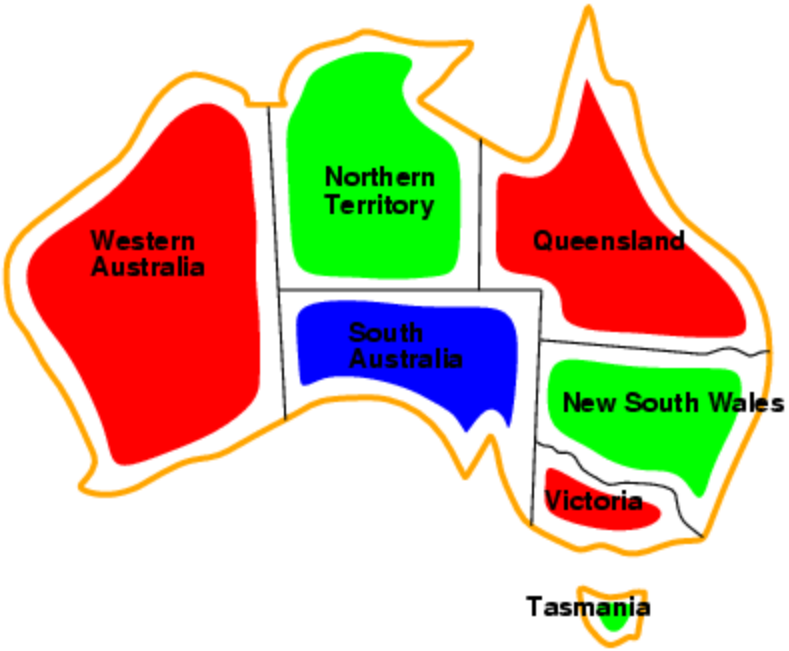
Example: Map Coloring



- **Variables:** WA, NT, Q, NSW, V, SA, T
- **Domains:** {red, green, blue}
- **Constraints:** adjacent regions must have different colors
  e.g., WA ≠ NT, or (WA, NT) in {(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)}

Complete Constraints?

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

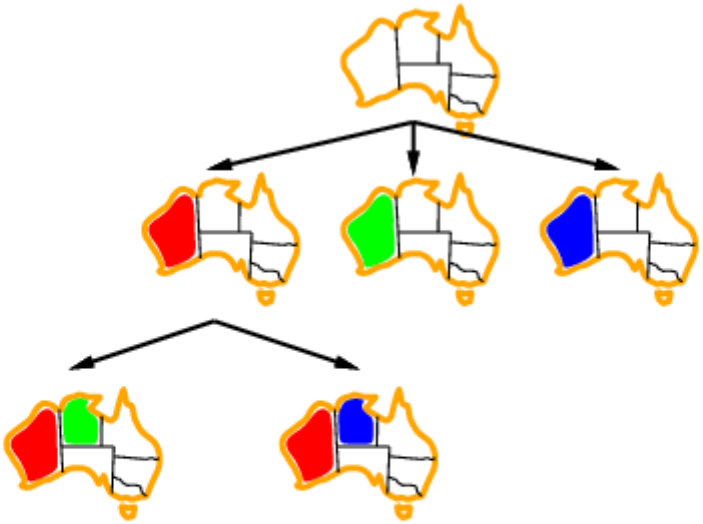
- 3^5 =243 assignments
- Easy elimination process in CSP : ex: {SA=blue}
- None of them will get blue
- only 2^5 =32 assignments to look at, a reduction of 87%.

- depth-limited search: partial assignment, and an action would be adding var = value to the assignment.
- CSP with n variables of domain size d, : the branching factor at the top level is nd because any of d values can be assigned to any of n variables. At
- the next level, the branching factor is (n − 1)d, and so on for n levels.
- We generate a tree with $n! \cdot d^n$ leaves, even though there are only d^n possible complete assignments!
- **COMMUTATIVITY : A problem is commutative if the order of application of any given** set of actions has no effect on the outcome.
- need only consider a *single variable at each node in the search tree.*
- *For example, at the root* node of a search tree for coloring the map of Australia, we might make a choice between
- SA=red, SA=green, and SA=blue, but we would never choose between SA=red and WA=blue.
- With this restriction, the number of leaves is d^n, as we would hope.

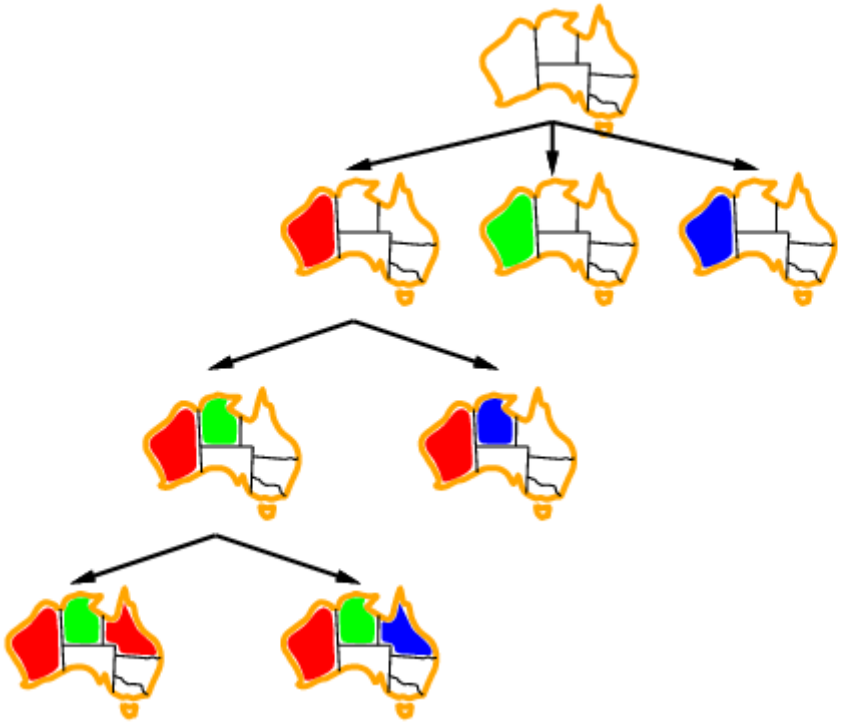
Example

Example

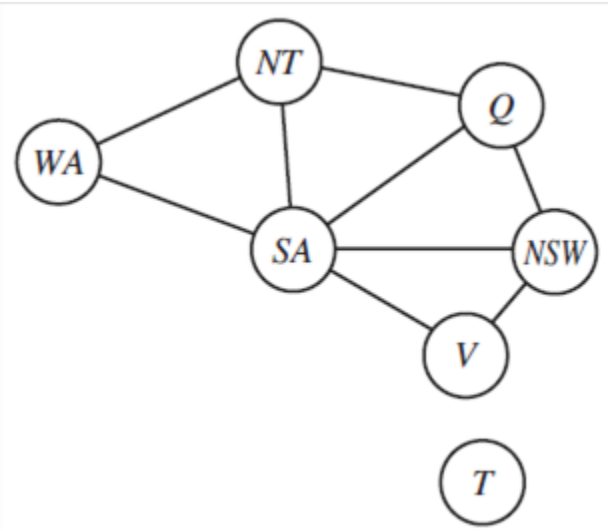
Example



Example



- an algorithm can search (choose a new variable assignment from several possibilities) or do a specific type of **inference called 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.
- The key idea is **local consistency.** If we treat each variable as a node in a graph and each binary constraint as an arc, then the process of enforcing local consistency in each part of the graph causes inconsistent values to be eliminated throughout the graph



node-consistent

- A single variable (corresponding to a node in the CSP network) is **node-consistent if all** the values in the variable's domain satisfy the variable's unary constraints.
- For example: SA dislike green, the variable SA starts with domain {red , green, blue},
- make it node consistent by eliminating green, leaving SA with the reduced domain {red , blue}.

Arc consistency

- **arc-consistency of var:** every value in its domain satisfies the variable's binary constraints.
- 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.
- For example, consider the constraint $Y = X^2$ where the domain of both X and Y is the set of digits.
- We can write this constraint explicitly as (X, Y), {(0, 0), (1, 1), (2, 4), (3, 9))} .
- To make X arc-consistent with respect to Y , we reduce X's domain to {0, 1, 2, 3}.
- make Y arc-consistent with respect to X, then Y 's domain becomes {0, 1, 4, 9} and the whole CSP is arc-consistent.

Path consistency

- A two-variable set $\{X_i,X_j\}$ is path-consistent with respect to a third variable X_m if, for every assignment $\{X_i = a,X_j = b\}$ consistent with the constraints on $\{X_i,X_j\}$, there is an assignment to Xm that satisfies the constraints on $\{X_i,X_m\}$ and $\{X_m,X_j\}$.
- This is called path consistency because one can think of it as looking at a path from X_i to X_j with X_m in the middle.
- Ex: coloring the Australia map with two colors.
- make the set {WA,SA} path consistent with respect to NT.
- only two enumerations : {WA = red ,SA = blue} and {WA = blue,SA = red}.
- with both of these assignments NT can be neither red nor blue (because it would conflict with either WA or SA).
- Because there is no valid choice for NT, eliminate both assignments, and end up with no valid assignments for {WA,SA}.

- **Global constraints**
- **resource constraint-** atmost constraint
- For example, in a scheduling problem, let P1, . . . , P4 denote the numbers of personnel assigned to each of four tasks.
- Constraint:no more than 10 personnel are assigned in total is written as Atmost(10, P1, P2, P3, P4).
- for example, if each variable has the domain {3, 4, 5, 6}?

Real-world CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetable problems
  - e.g., which class is offered when and where?
- Transportation scheduling
- Factory scheduling

- More examples of CSPs: http://www.csplib.org/

We consider a small part of the car assembly, consisting of 15 tasks: install axles (front and back), affix all four wheels (right and left, front and back), tighten nuts for each wheel, affix hubcaps, and inspect the final assembly. We can represent the tasks with 15 variables:

$$X = \{Axle_F, Axle_B, Wheel_{RF}, Wheel_{LF}, Wheel_{RB}, Wheel_{LB}, Nuts_{RF}, Nuts_{LF}, Nuts_{RB}, Nuts_{LB}, Cap_{RF}, Cap_{LF}, Cap_{RB}, Cap_{LB}, Inspect\} .$$

The value of each variable is the time that the task starts. Next we represent **precedence constraints** between individual tasks. Whenever a task T_1 must occur before task T_2, and task T_1 takes duration d_1 to complete, we add an arithmetic constraint of the form

$$T_1 + d_1 \leq T_2 .$$

In our example, the axles have to be in place before the wheels are put on, and it takes 10 minutes to install an axle, so we write

$$Axle_F + 10 \leq Wheel_{RF}; \quad Axle_F + 10 \leq Wheel_{LF};$$
$$Axle_B + 10 \leq Wheel_{RB}; \quad Axle_B + 10 \leq Wheel_{LB} .$$

Next we say that, for each wheel, we must affix the wheel (which takes 1 minute), then tighten the nuts (2 minutes), and finally attach the hubcap (1 minute, but not represented yet):

$$Wheel_{RF} + 1 \leq Nuts_{RF}; \quad Nuts_{RF} + 2 \leq Cap_{RF};$$
$$Wheel_{LF} + 1 \leq Nuts_{LF}; \quad Nuts_{LF} + 2 \leq Cap_{LF};$$
$$Wheel_{RB} + 1 \leq Nuts_{RB}; \quad Nuts_{RB} + 2 \leq Cap_{RB};$$
$$Wheel_{LB} + 1 \leq Nuts_{LB}; \quad Nuts_{LB} + 2 \leq Cap_{LB} .$$

Suppose we have four workers to install wheels, but they have to share one tool that helps put the axle in place. We need a **disjunctive constraint** to say that $Axle_F$ and $Axle_B$ must not overlap in time; either one comes first or the other does:

$$(Axle_F + 10 \leq Axle_B) \quad \textbf{or} \quad (Axle_B + 10 \leq Axle_F) .$$

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

Crypt arithmetic problem

(The name is traditional but confusing because it need not involve *all* the variables in a problem). One of the most common global constraints is *Alldiff*, which says that all of the variables involved in the constraint must have different values. In Sudoku problems (see Section 6.2.6), all variables in a row or column must satisfy an *Alldiff* constraint. Another example is provided by **cryptarithmetic** puzzles. (See Figure 6.2(a).) Each letter in a cryptarithmetic puzzle represents a different digit. For the case in Figure 6.2(a), this would be represented as the global constraint $Alldiff(F, T, U, W, R, O)$. The addition constraints on the four columns of the puzzle can be written as the following n-ary constraints:

$$
\begin{aligned}
O + O &= R + 10 \cdot C_{10} \\
C_{10} + W + W &= U + 10 \cdot C_{100} \\
C_{100} + T + T &= O + 10 \cdot C_{1000} \\
C_{1000} &= F \,,
\end{aligned}
$$

where C_{10}, C_{100}, and C_{1000} are auxiliary variables representing the digit carried over into the tens, hundreds, or thousands column. These constraints can be represented in a **constraint hypergraph**, such as the one shown in Figure 6.2(b). A hypergraph consists of ordinary nodes (the circles in the figure) and hypernodes (the squares), which represent n-ary constraints.

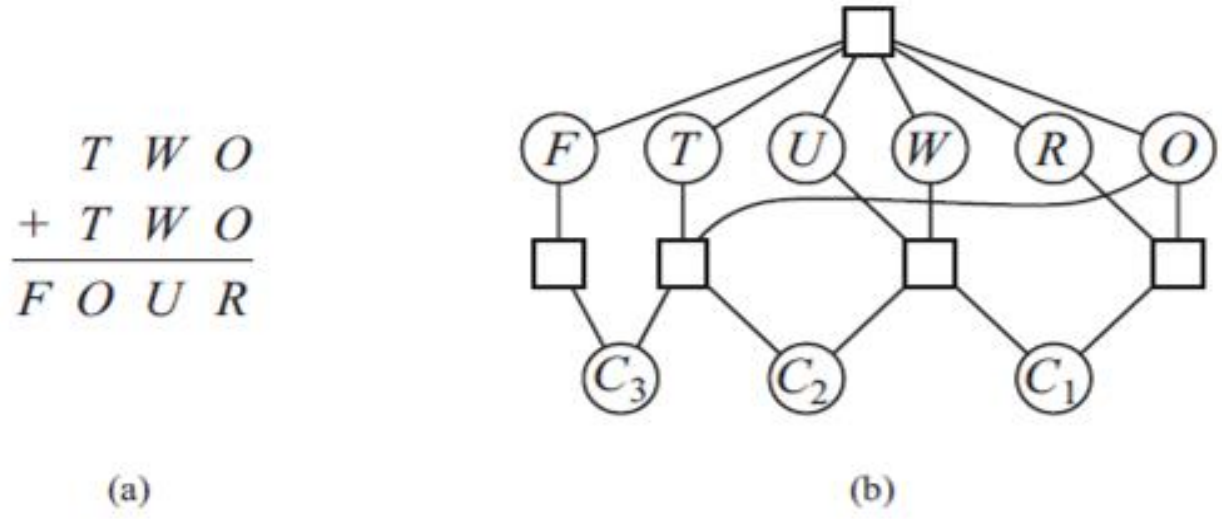


Figure 6.2 (a) A cryptarithmetic problem. Each letter stands for a distinct digit; the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, with the added restriction that no leading zeroes are allowed. (b) The constraint hypergraph for the cryptarithmetic 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.

- **bounds propagation**
- **For example, in an airline-scheduling problem**
- let's suppose there are two flights, F1 and F2, for which the planes have capacities 165 and 385, respectively.
- The initial domains for the numbers of passengers on each flight are then

  D1 = [0, 165] and D2 = [0, 385] .
- Now suppose we have the additional constraint that the two flights together must carry 420 people: F1 + F2 = 420.
- Propagating bounds constraints, we reduce the domains to ?
- D1 = [35, 165] and D2 = [255, 385] .

Variables: X_{ij}

Domains: {1, 2, …, 9}

Constraints:

Alldiff(X_{ij} in the same *unit*)

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		

(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 6.4 (a) A Sudoku puzzle and (b) its solution.

Alldiff constraints: one for each row, column, and box of 9 squares.

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

Example: Cryptarithmetic

- **Variables:** T, W, O, F, U, R

  X_1, X_2
- **Domains**: {0, 1, 2, …, 9}
- **Constraints:**

  Alldiff(T, W, O, F, U, R)

  $O + O = R + 10 * X_1$

  $W + W + X_1 = U + 10 * X_2$

  $T + T + X_2 = O + 10 * F$

  $T \neq 0, F \neq 0$