Constraint satisfaction problems (CSPs)

Unit-III

Outline

- Constraint Satisfaction Problems (CSP)
- Backtracking search for CSPs
- Local search for CSPs

Constraint Satisfaction Problems

- A Constraint Satisfaction Problem (CSP) involves a set of variables, each of which has a domain of possible values, and a set of constraints that define the allowable combinations of values for the variables.
- A constraint satisfaction problem (CSP) is defined by
  - X: A set of variables {X1, X2,.....,Xn} and a
  - C: A set of constraints {C1,C2,....,Cm} and
  - D: A set of domains {D1, D2,...,Dn},
- where each variable Xi has a nonempty domain Di of possible values.
- Each constraint Ci involves some subset of the variables and specifies the allowable combinations of values for that subset.

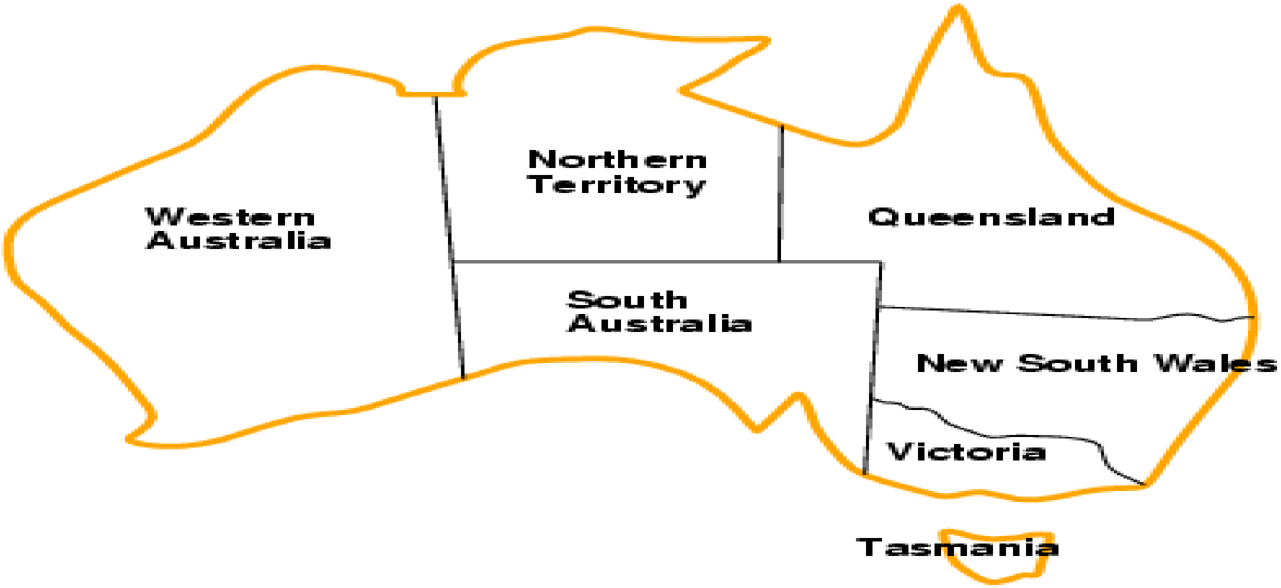
Constraint Satisfaction Problems

- The goal of a CSP is to find an assignment values to the variables that satisfies all the constraints. This assignment is called a solution to the CSP.
- **Standard search problem:** State is a "black box“ – any data structure that supports successor function, heuristic function, and goal test.
- **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
- Allows useful general-purpose algorithms with more power than standard search algorithms

Constraint Satisfaction Problems

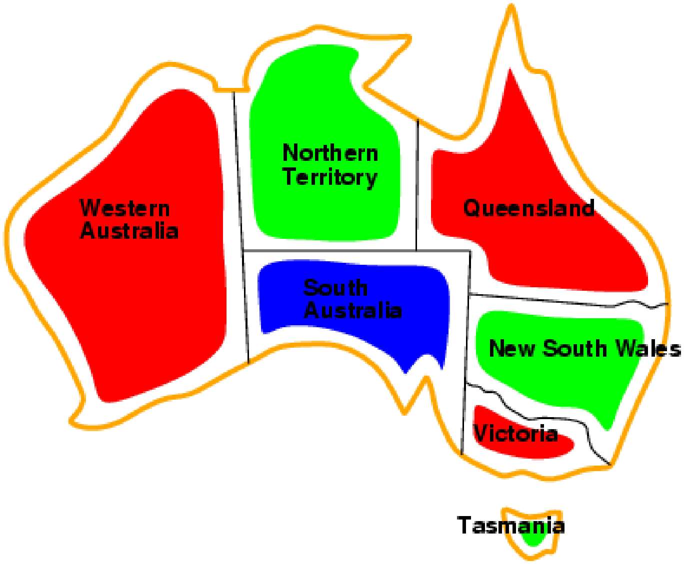
- An assignment of values to some or all of the variables that doesn’t violet any constraints is called a consistent or legal assignment.
- A complete assignment is that in which every variable is mentioned and if it satisfies all the constraints, then it is a solution.
- Partial Assignment: An assignment which assigns values to some of the variables only such type of assignments are called Partial assignments.

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, or (WA,NT) in {(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)}

Example: Map-Coloring

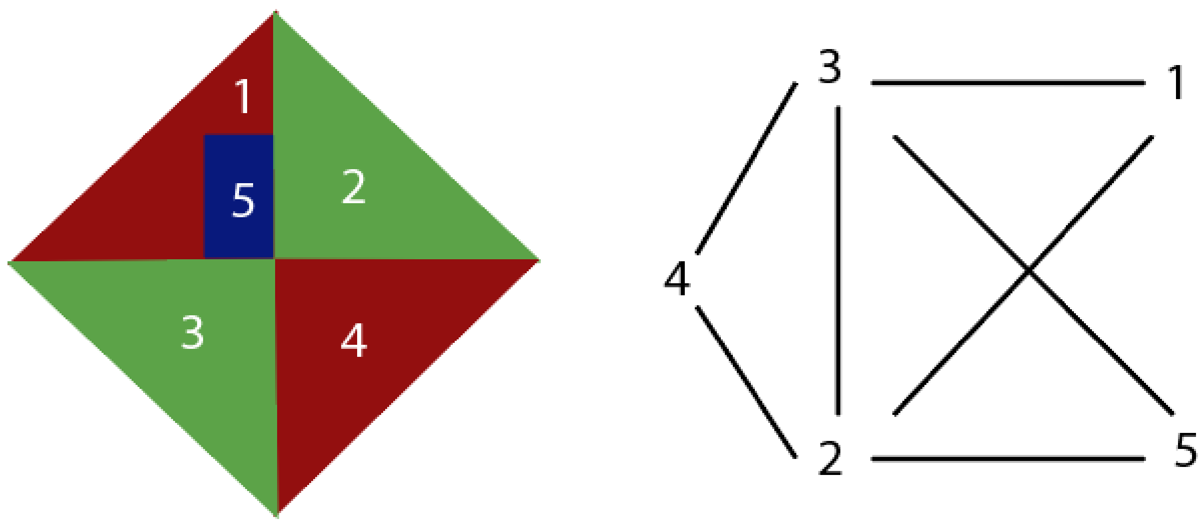


- A search procedure that does not use constraints would have to consider 3^5=243 assignments for the five neighboring variables;
- With constraints we have only 2^5=32 assignments to consider, a reduction of 87%.

- Solutions are complete and consistent assignments, e.g., WA = red, NT = green,Q = red,NSW = green,V = red,SA = blue,T = green

Graph Coloring

- **Graph Coloring:** The problem where the constraint is that no adjacent sides can have the same color.



Graph Coloring

Sudoku

- **Sudoku Playing:** The gameplay where the constraint is that no number from 0-9 can be repeated in the same row or column.

SUDOKU

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

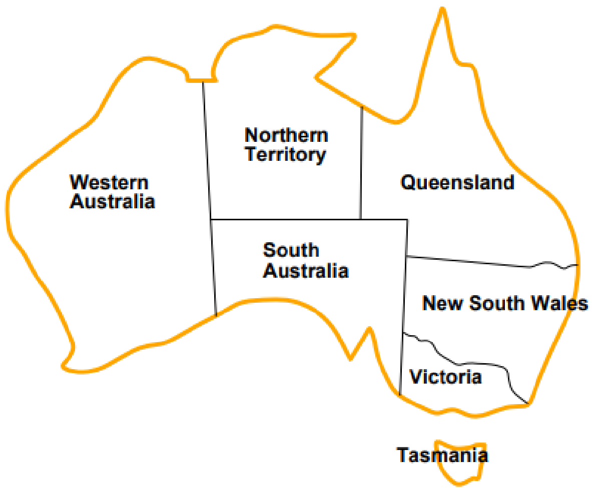
Puzzle

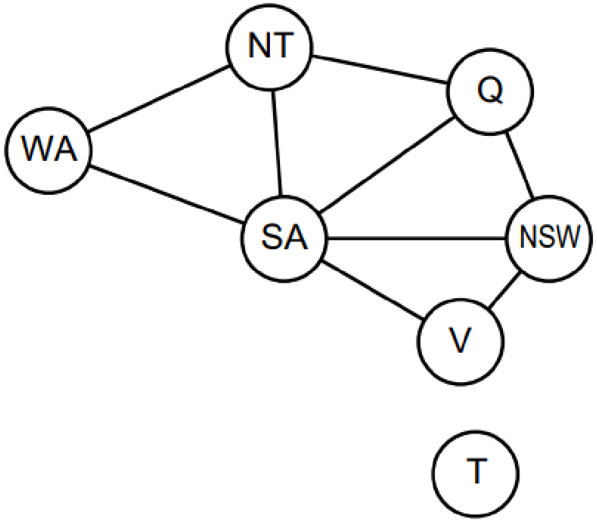
4	1	7	6	8	3	2	5	9
2	6	9	5	7	1	8	3	4
3	8	5	4	9	2	6	7	1
8	4	2	7	6	5	9	1	3
6	5	3	8	1	9	7	4	2
9	7	1	2	3	4	5	6	8
7	2	6	3	4	8	1	9	5
1	3	8	9	5	6	4	2	7
5	9	4	1	2	7	3	8	6

Solution

Constraint graph

- Binary CSP: each constraint relates two variables
- Constraint graph: nodes are variables, arcs are constraints





- A constraint graph can be used to visualize binary constraints.
- General-purpose CSP methods use the graph structure to speed up search. e.g., Tasmania is an independent subproblem!

Varieties of CSPs

- Discrete variable:
  - finite domains:
    - n variables, domain size $d \rightarrow O(d^n)$ complete assignments
    - e.g., Boolean CSPs, incl.~Boolean satisfiability (NP-complete)
  - infinite domains:
    - integers, strings, etc.
    - e.g., job scheduling, variables are start/end days for each job
    - need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$
    - linear constraints solvable, nonlinear undecidable
- Continuous variables
- e.g., start/end times for Hubble Space Telescope observations
  - linear constraints solvable in polynomial time by linear programming

Varieties of constraints

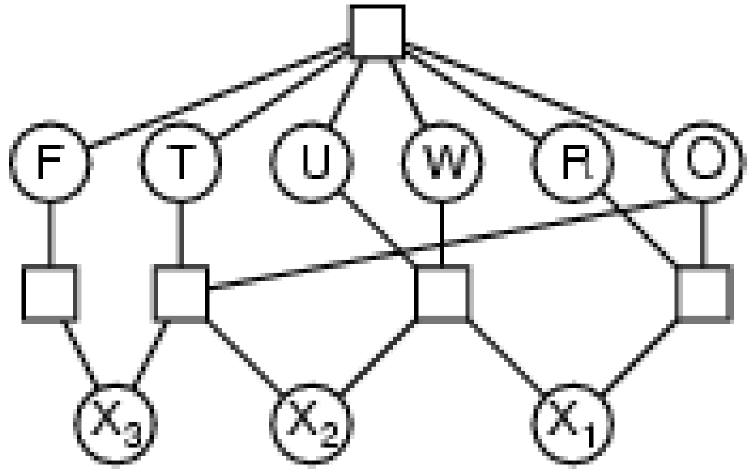
- Unary constraints involve a single variable.
  - e.g., SA ≠ green
- Binary constraints involve pairs of variables.
  - e.g., SA ≠ WA
- The ternary constraint Between(X,Y,Z), for example, can be defined as {(X, Y, Z), X <Y < Z or X >Y > Z}.
- Global Constraints/Higher-order constraints involve 3 or more variables or an arbitrary number of variables.
  - e.g., cryptarithmetic column constraints
- The name is traditional but confusing because a global constraint need not involve all the variables in a problem.

Varieties of constraints

- 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, all variables in a row, column, or 3X3 box must satisfy an Alldiff constraint.
- Preferences/soft constraints e.g., red is better than green often representable by a cost for each variable assignment → constrained optimization problems

Example: Cryptarithmetic

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



- Variables: *F T U W R O* X_1 X_2 X_3
- Domains: {*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: crypt arithmetic problem

```
     C4 C3  C2  C1
        S    E   N   D
  +     M    O   R   E
     ________________
     M  O    N   E   Y
```

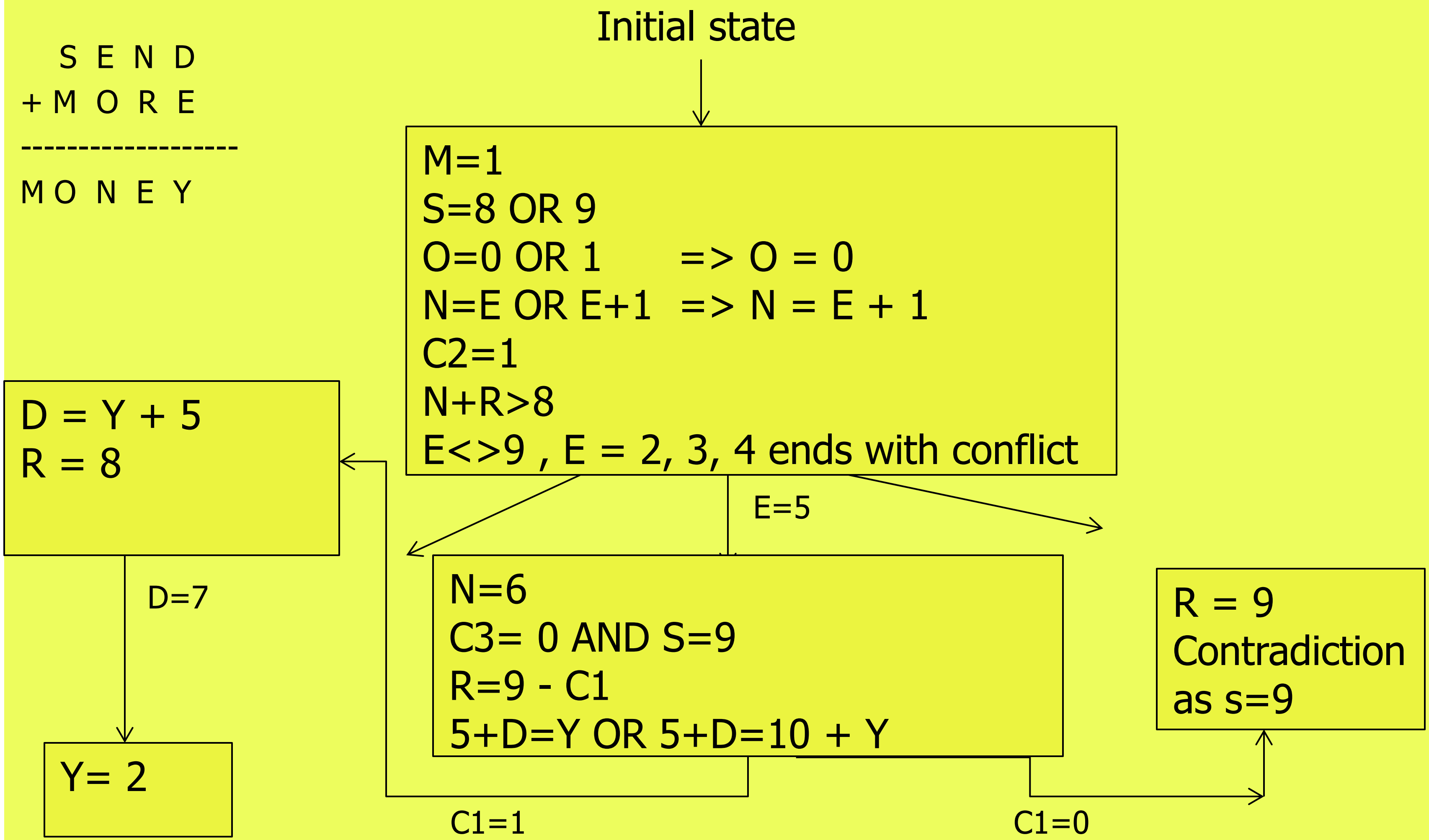
1. D + E = 10 * C1 + Y
2. C1 + N + R = 10 * C2 + E
3. C2 + E + O = 10 * C3 + N
4. C3 + S + M = 10 * C4+ O
5. C4 = M = 1 As M has to be non-zero .

Where C1,C2,C3 and C4 can be either 0 or 1.

Other alphabets can take unique values from the set { 0,1,2,3,4,5,6,7,9}

Example: crypt arithmetic problem

```
  S E N D
+ M O R E
-------------------
M O N E Y
```



SOLUTION: {S=9,E=5,N=6,D=7,M=1,O=0, R=8,Y=2} and { C1 =1, C2 = 1, C3 =0 , C4 = 4 }

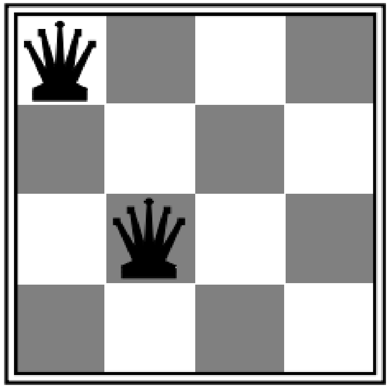
Sudoku as a CSP

- Variables: 81 variables
  - A1, A2, A3, …, I7, I8, I9
  - Letters index rows, top to bottom
  - Digits index columns, left to right
- Domains: The nine positive digits
  - A1 $\in$ {1, 2, 3, 4, 5, 6, 7, 8, 9}
  - Etc.
- Constraints: *Alldiff* constraints
  - *Alldiff*(A1, A2, A3, A4, A5, A6, A7, A8, A9)
  - Etc.
- [Why constraint satisfaction?]

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

4-Queens as a CSP

- Assume one queen in each column. Which row does each one go in?



$Q_1 = 1 \quad Q_2 = 3$

- Variables Q 1, Q 2, Q 3, and Q 4
- Domains Di = { 1, 2, 3, 4 }
- Constraints Qi ≠ Qj (cannot be in same row)
- |Qi – Qj | ≠ |i – j| (cannot be on same diagonal)
- Translate each constraint into set of allowable values for its variables
- E.g., values for (Q1, Q2) are (1, 3) (1, 4) (2, 4) (3, 1) (4, 1) (4, 2)

Real-world CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Transportation scheduling
- Factory scheduling: Job-shop scheduling
- Notice that many real-world problems involve real-valued variables

Standard search formulation (incremental)

- A CSP can easily be expressed as a standard search problem.
- Let's start with the straightforward approach, then fix it
- States are defined by the values assigned so far
  - Initial state: the empty assignment { }
  - Successor function: assign a value to an unassigned variable that does not conflict with current assignment
  - → fail if no legal assignments
  - Goal test: the current assignment is complete

Standard search formulation (incremental)

- Note:

1. This is the same for all CSPs:
2. Every solution appears at depth *n* with *n* variables→ use depth-first search.
3. Path is irrelevant, so can also use complete-state formulation.
4. However, with domain of size d, branching factor b = (n − ℓ) d at depth ℓ, hence n!d^n leaves!

Backtracking search

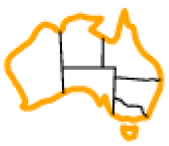
- Variable assignments are commutative},
- i.e., [WA = red then NT = green] same as [NT = green then WA = red]
- Only need to consider assignments to a single variable at each node.
- Chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.
- *Depth-first search* for CSPs with single-variable assignments is called backtracking search
- Backtracking search is the basic uninformed search algorithm for CSPs
- Can solve *n*-queens for $n \approx 25$

Backtracking search

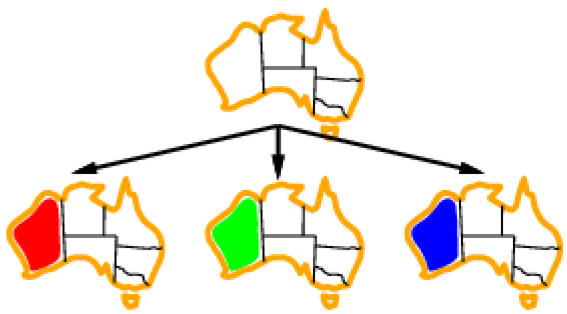
```
function BACKTRACKING-SEARCH( csp) returns a solution, or failure
  return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING( assignment, csp) returns a solution, or
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 according to Constraints[csp] then
      add { var = value } to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failue then return result
      remove { var = value } from assignment
  return failure
```

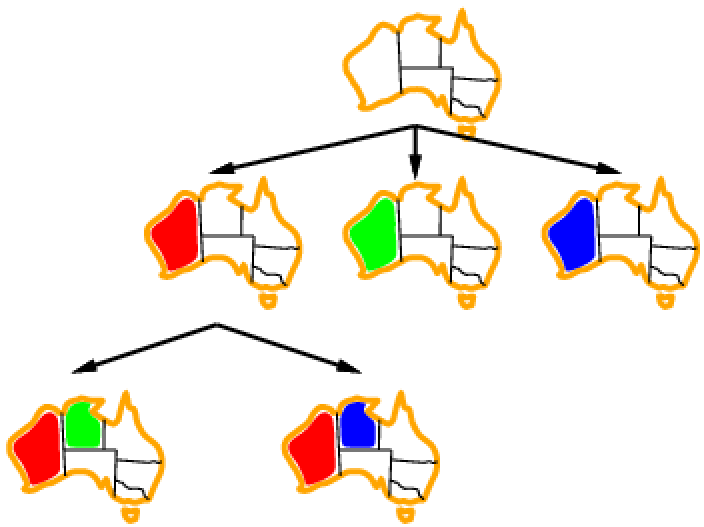
Backtracking example



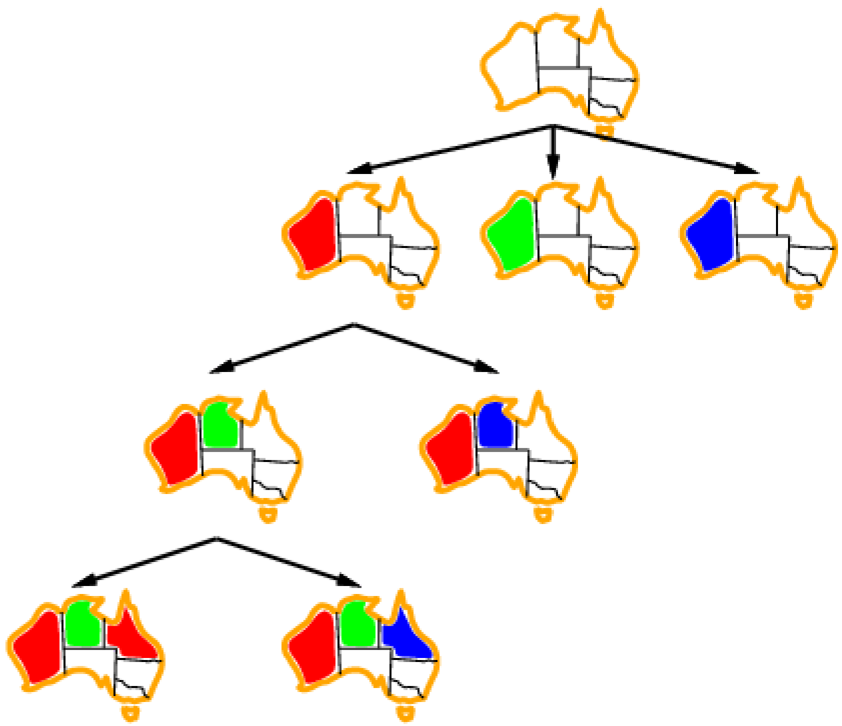
Backtracking example



Backtracking example



Backtracking example

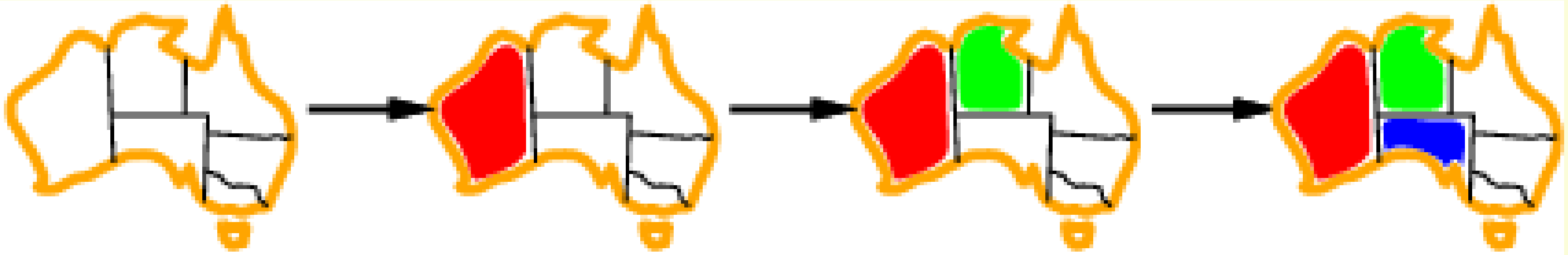


Improving backtracking efficiency

- 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?

Most constrained variable

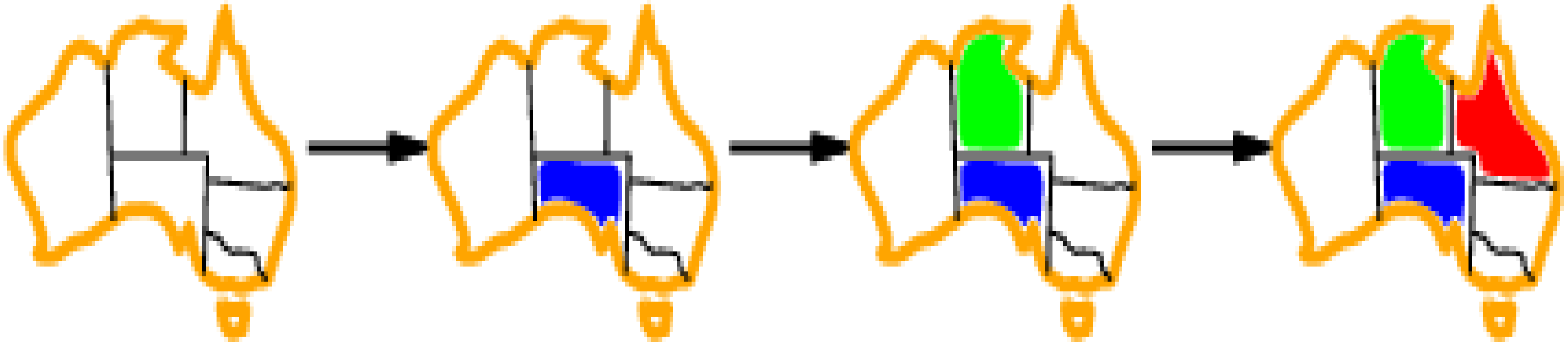
- Most constrained variable:
  Choose the variable with the fewest legal values



- Minimum remaining values (MRV) heuristic

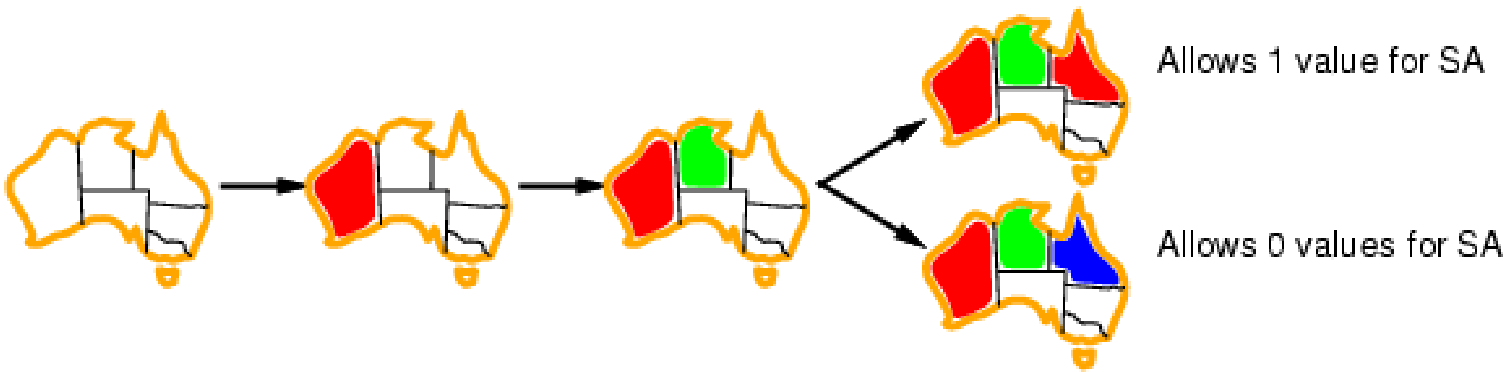
Most constraining variable

- Tie-breaker among most constrained variables
- Most constraining variable:
- Choose the variable with the most constraints on remaining variables.



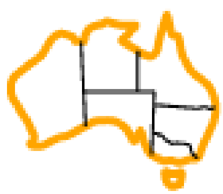
Least constraining value

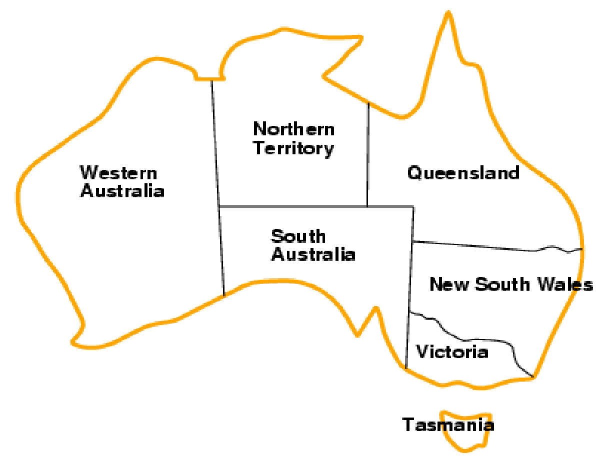
- Given a variable, choose the least constraining value:
- The one that rules out the fewest values in the remaining variables

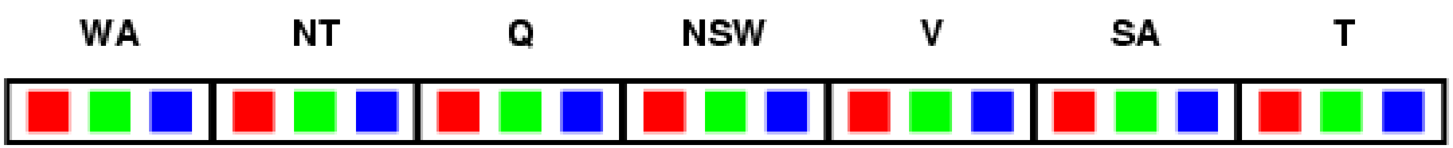


Forward checking

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

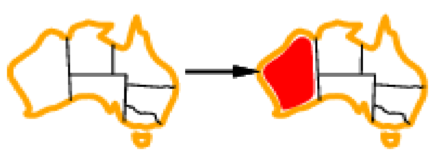


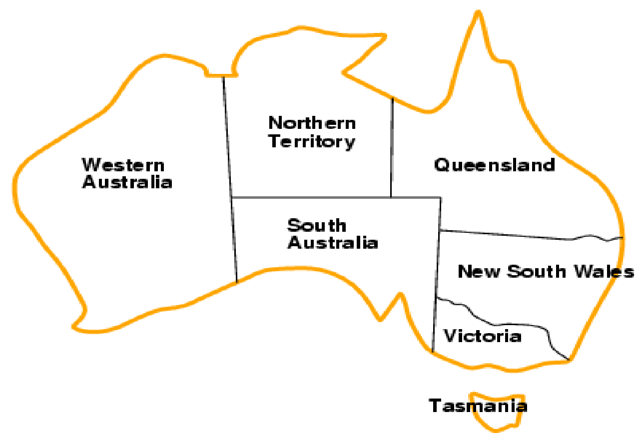


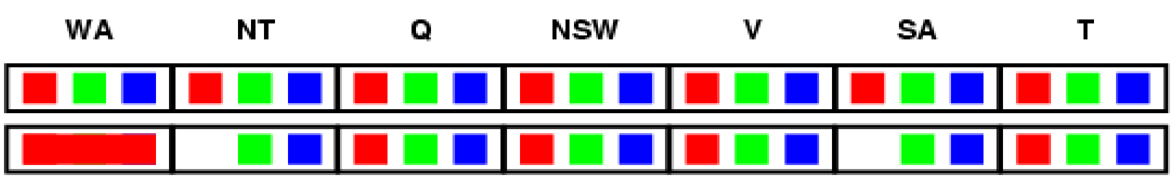


Forward checking

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

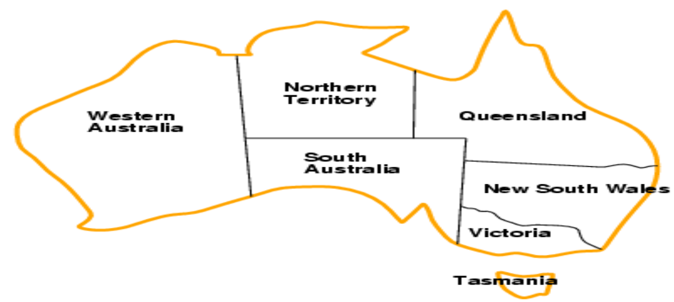


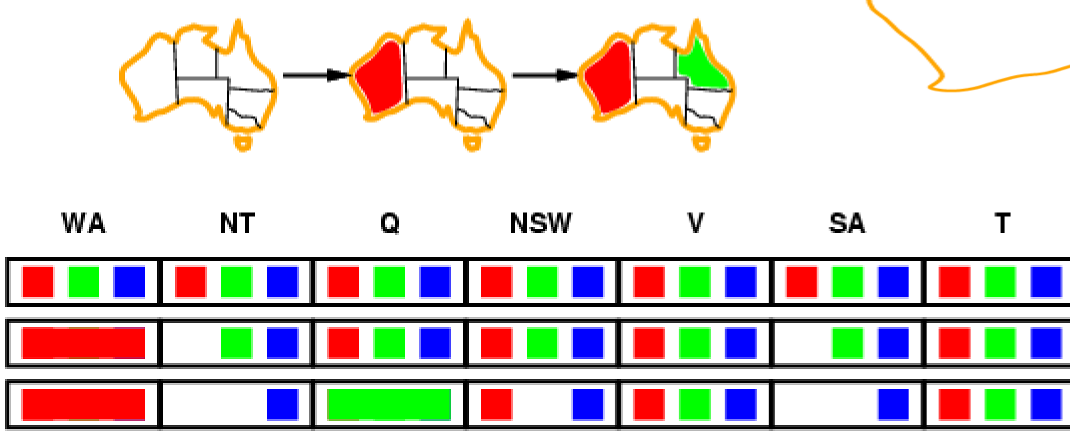




Forward checking

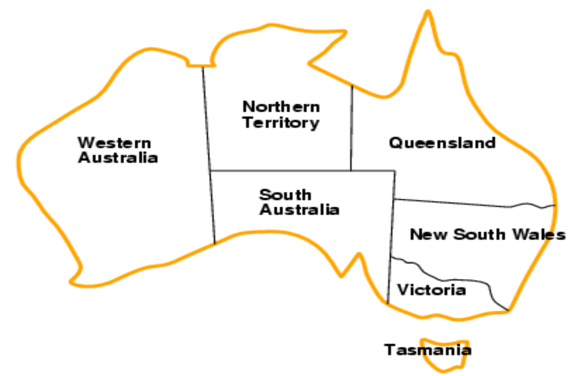
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values.

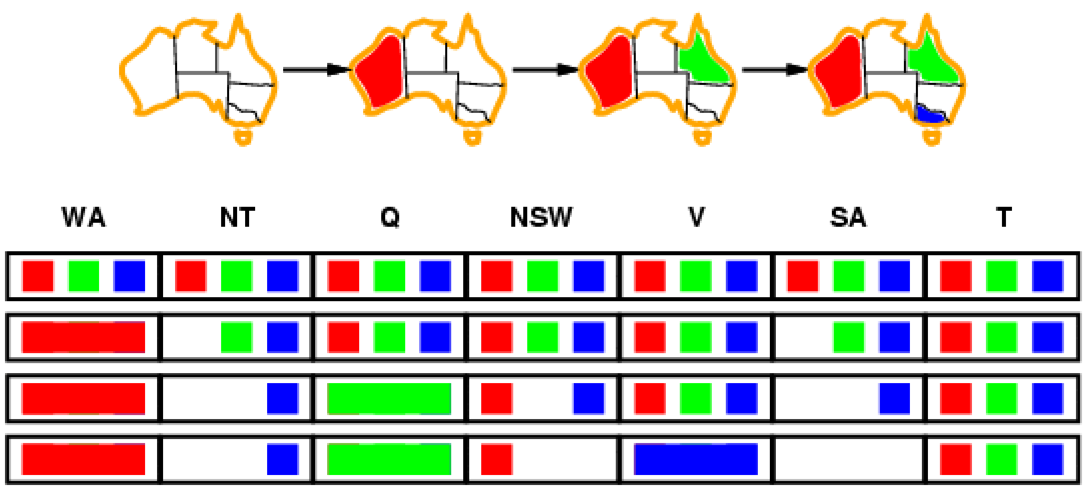




Forward checking

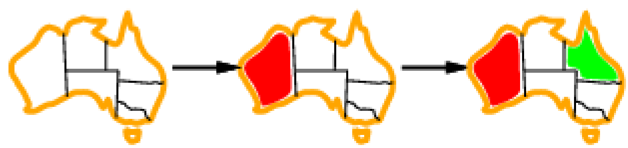
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

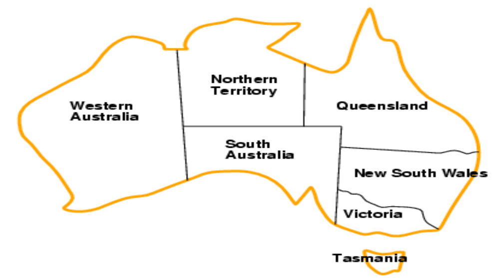




Constraint propagation

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

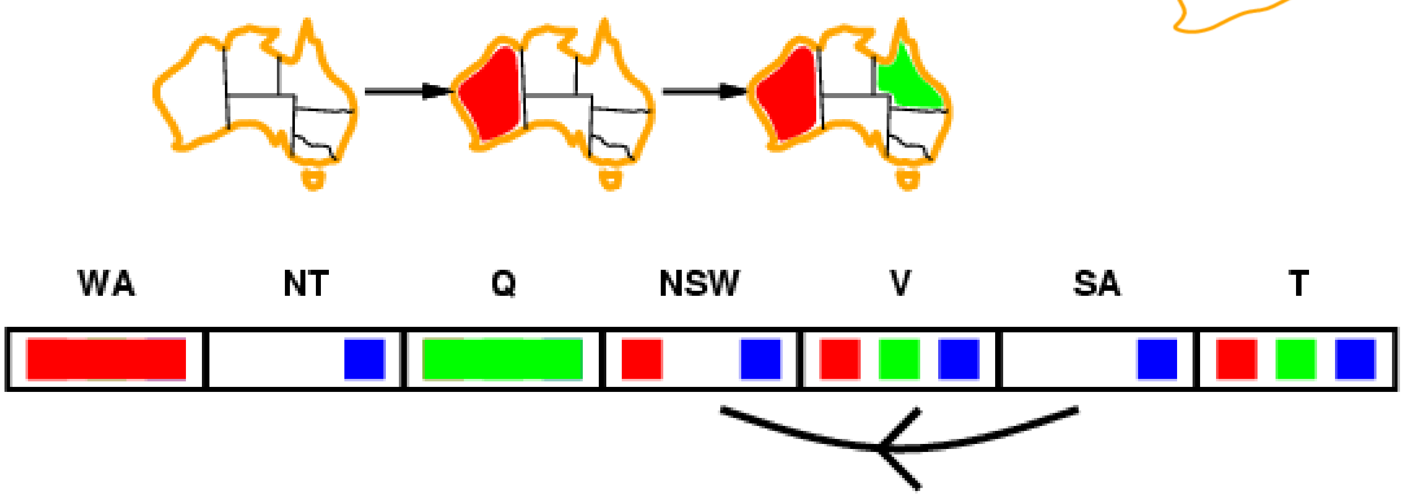




- NT and SA cannot have both blue!
- Constraint propagation repeatedly enforces constraints locally.

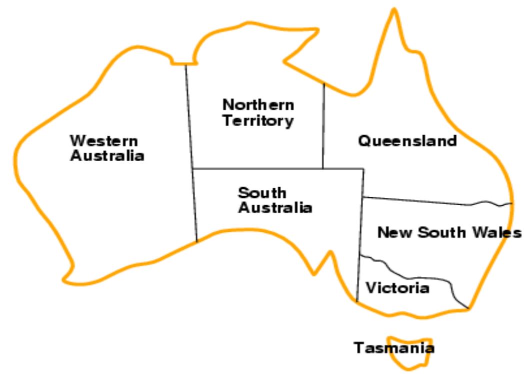
Arc consistency

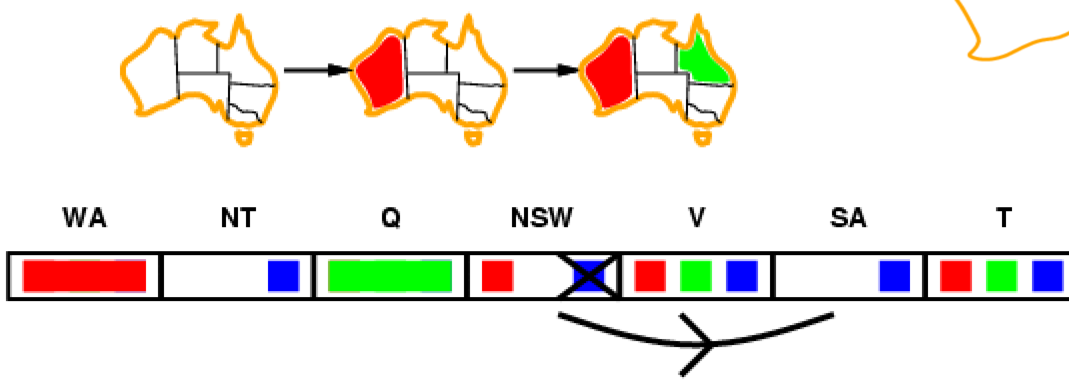
- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff for every value x of X, there is some allowed y
-



Arc consistency

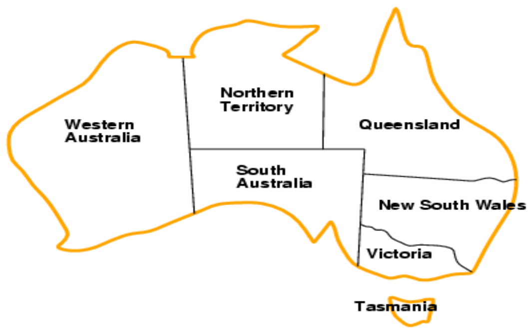
- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff,
- for every value x of X, there is some allowed y
-

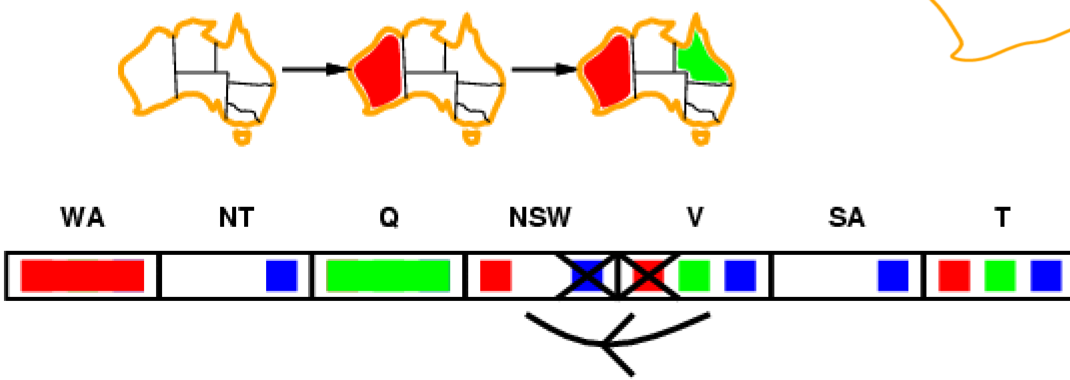




Arc consistency

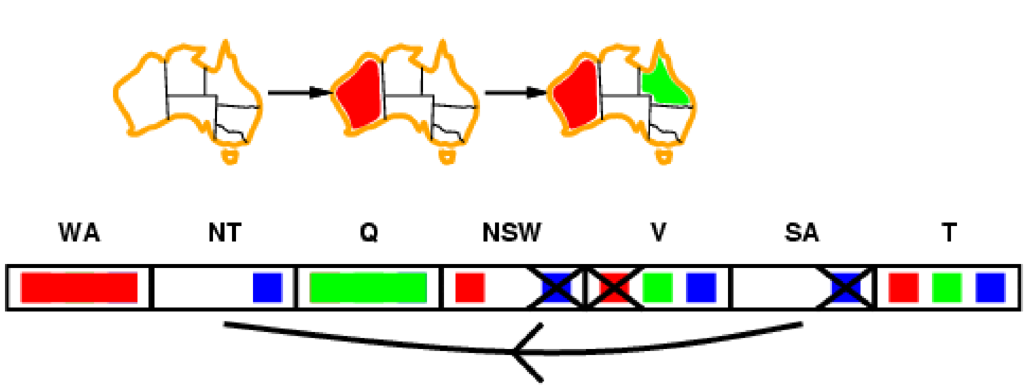
- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff
- for every value x of X there is some allowed y
-

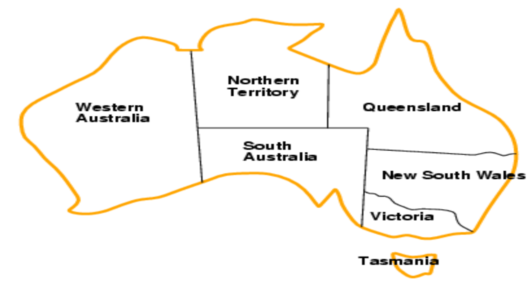




- If X loses a value, neighbors of X need to be rechecked

Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ 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 AC-3

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains
  inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if RM-INCONSISTENT-VALUES(X_i, X_j) **then**
      for each X_k **in** NEIGHBORS$[X_i]$ **do**
        add (X_k, X_i) to *queue*

---

function RM-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff remove a value
  removed $\leftarrow$ *false*
  for each x **in** DOMAIN$[X_i]$ **do**
    if no value y in DOMAIN$[X_j]$ allows (x,y) to satisfy constraint(X_i, X_j)
      then delete x from DOMAIN$[X_i]$; *removed* $\leftarrow$ *true*
  return *removed*

Solving CSPs by Searching

- A finite CSP could be solved by exhaustively searching the total assignments where, a total assignment assigns a value to every variable..
- The **generate-and-test** algorithm to find one solution is as follows: check each total assignment in turn; if an assignment is found that satisfies all of the constraints, return that assignment.
- A generate-and-test algorithm to find all solutions is the same except, instead of returning the first solution found, it enumerates the solutions.
- If there are n variables, each with domain size d, there are d^n total assignments. If there are e constraints, the total number of constraints tested is O(ed^n).
- As **n** becomes large, this becomes intractable very quickly.

Solving CSPs by Searching

- Suppose the delivery robot must carry out a number of delivery activities, **a, b, c, d**, and **e**.
- Suppose that each activity happens at any of times 1, 2, 3, or 4. Let A be the variable representing the time that activity **a** will occur, and similarly for the other activities.
- The variable domains, which represent possible times for each of the deliveries, are

$$dom(A) = \{1,2,3,4\},\quad dom(B) = \{1,2,3,4\},\quad dom(C) = \{1,2,3,4\},$$

$$dom(D) = \{1,2,3,4\},\quad dom(E) = \{1,2,3,4\}.$$

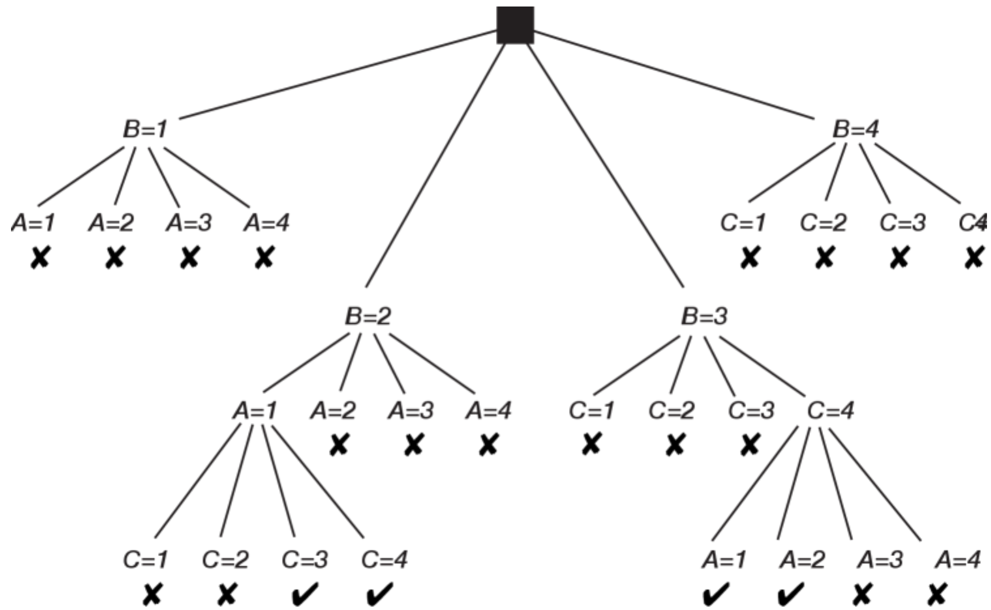
Suppose the following constraints must be satisfied:

$$\{\quad (B \neq 3),\ (C \neq 2),\ (A \neq B),\ (B \neq C),\ (C < D),\ (A = D),$$

$$(E < A),\ (E < B),\ (E < C),\ (E < D),\ (B \neq D).\}$$

Solving CSPs by Searching

- **Example.** Consider a CSP with variables A, B, and C, each with domain {1,2,3,4}, and constraints A<B and B<C. A possible search tree is shown in Figure.



- In this figure, a node corresponds to all of the assignments from the root to that node. The potential nodes that are pruned because they violate constraints are labeled ✗.
- The leftmost ✗ corresponds to the assignment {A= 1, B= 1}. This violates the A<B constraint, and so it is pruned.

Solving CSPs by Searching

- This CSP has four solutions. The leftmost one is {A= 1, B= 2, C= 3}. The size of the search tree, and thus the efficiency of the algorithm, depends on which variable is selected at each time.
- A static ordering, such as always splitting on A then B then C, is usually less efficient than the dynamic ordering used here,
- But it might be more difficult to find the best dynamic ordering than to find the best static ordering.
- The set of answers is the same regardless of the variable ordering.
- There would be 4^3=64 total assignments tested in a generate-and-test algorithm.
- For the search method, there are 8 total assignments generated, and 16 other partial assignments that were tested as to whether they satisfy some of the constraints.

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