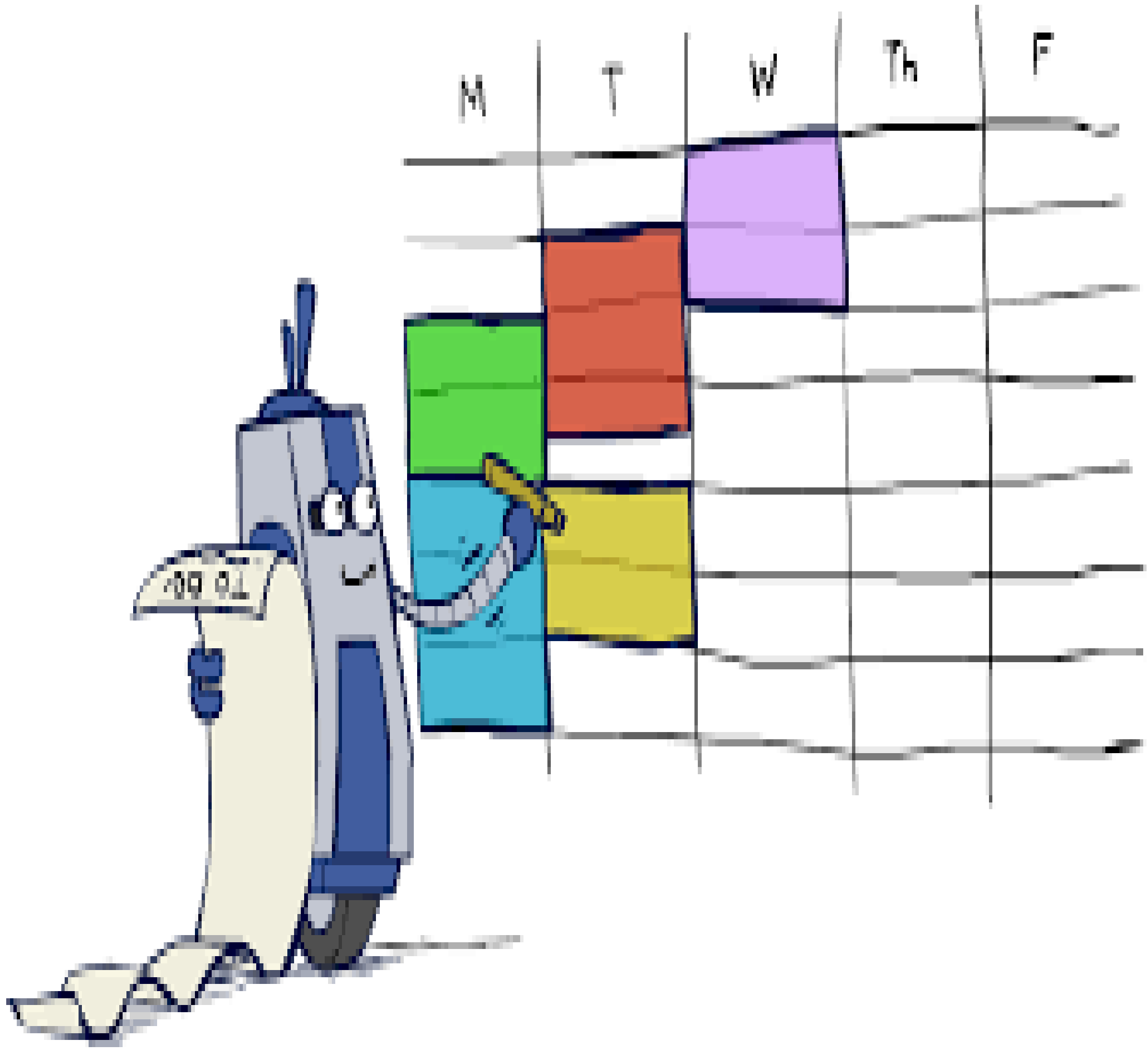
CONSTRAINT SATISFACTION PROBLEM

CSP is a problem consisting of finite set of variables, which are associated with finite domain, and constraints, which restrict the values that the variables can simultaneously take. The challenge is to assign a value to each variable satisfying all the constraints.



CONSTRAINT SATISFACTION PROBLEM COMPONENTS

- CSPs represented using 3components **X**, **D** and **C:**

  1. **X** is a set of **variables** {X1,X2, ……Xn} and a fixed values for each Xi represent a **State.**
  2. Each variable Xi has a nonempty **domain** **{D1,D2,………Dn}** of **possible values** where Di = **{v1 , …, vk }** **set of allowable values.** Each ***constraint C_i*** limits the values that variables can take.
  3. **C** is a set of **constraints** that specify set of allowable values for variable Vi **{C1,C2,…….Cm}. Goal Test** defined by set of constraints**.** Each constraint **Ci** is mentioned as a pair ***<scope,rel> where*** **scope is set of variables that participate in constraint definition and relation defines values that variables can take.**

- **Each *state* in a CSP is an *assignment* of values to some or all variables, {X_i=V_i, X_j=V_j, …}.** CSP search algorithms take advantage of this structure of states to solve complex problems.

CSP SOLUTION

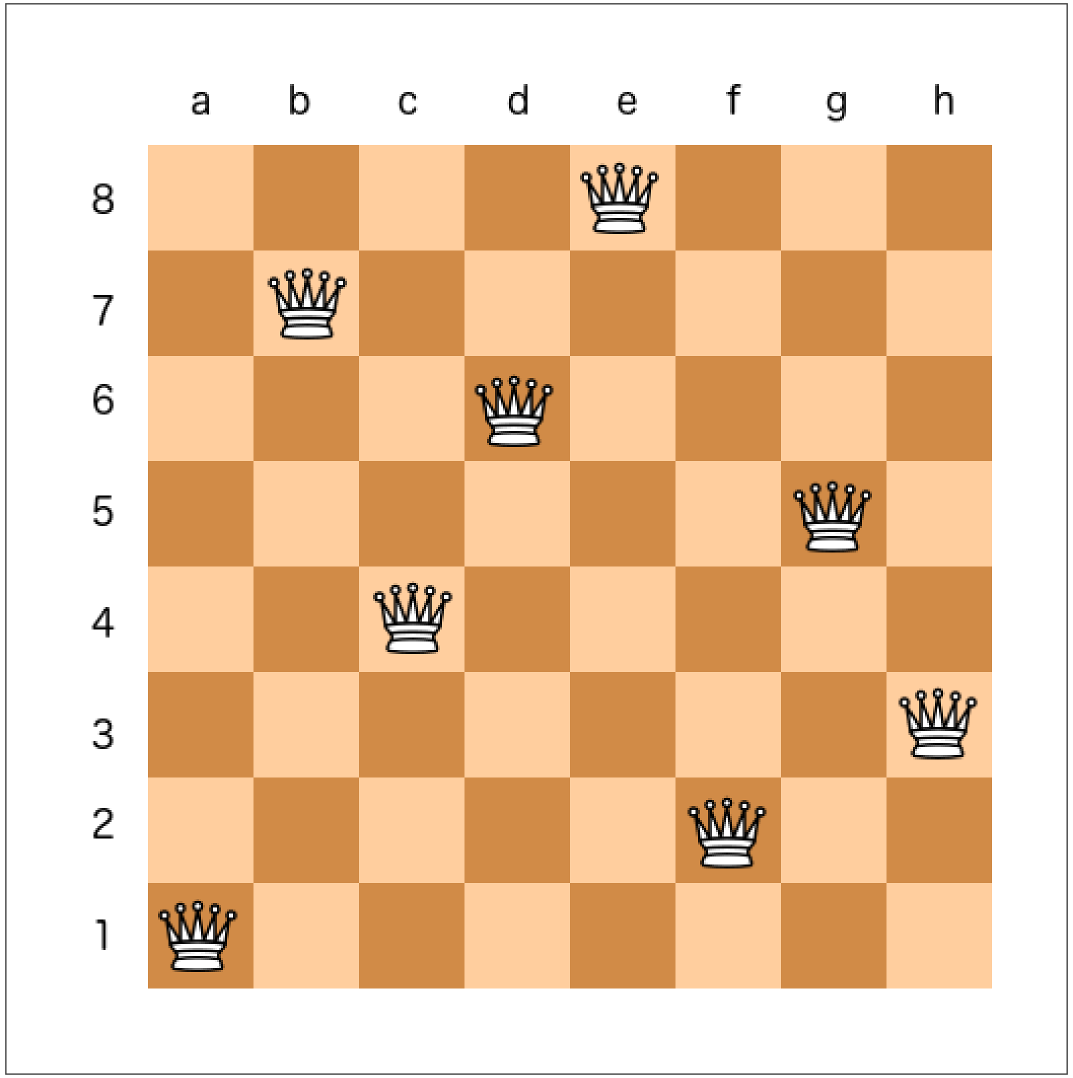
- ***Consistent assignment or Legal Assignment:*** An assignment does not violate any constraints.
- ***Complete Assignment:*** An assignment in which **every variable is assigned a value.**
- ***Partial Assignment:*** An assignment in which some variables have no values.
- **A *solution* to a CSP is a complete assignment that satisfies all constraints ie consistent assignment.**
- Some CSPs require an **objective function** that **maximizes or minimizes** solution.

CSP TO BE FORMULATED AS A STANDARD SEARCH PROBLEM

- **A CSP can easily be expressed as a standard search problem.**
- GOAL is to find a CONSISTENT ASSIGNMENT (if one exists)
  - *Initial State:* the empty assignment {}.
  - *Operators:* Assign value to unassigned variable provided that there is no conflict.
  - *Goal test:* assignment consistent and complete.
  - *Path cost:* constant cost for every step.
  - Solution is found at depth n, for n variables
  - Hence depth first search can be used

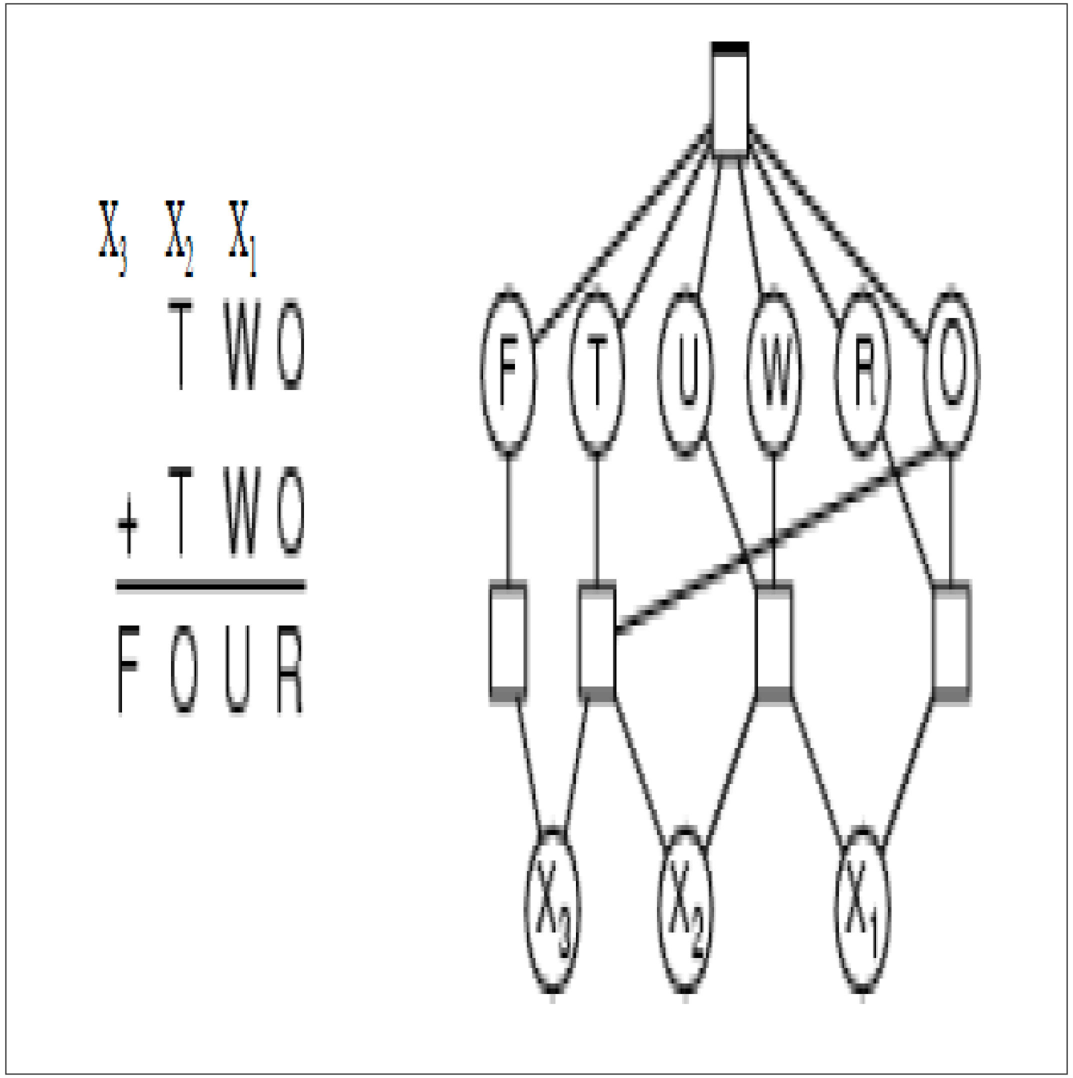
Popular Problems with CSP

- **CryptArithmetic** (Coding alphabets to numbers.)
- **n-Queen** (In an n-queen problem, n queens should be placed in an nXn matrix such that no queen shares the same row, column or diagonal.)
- **Map Coloring** (coloring different regions of map, ensuring no adjacent regions have the same color)
- **Crossword** (everyday puzzles appearing in newspapers)
- **Sudoku** (a number grid)
- **Scheduling problems**
  - **Job shop scheduling**
  - **Scheduling the Hubble Space Telescope**



EXAMPLE: 8-Queens

- **Variables:** Queens, one per column
  - Q1, Q2, …, Q8
- **Domains:** row placement, {1,2,…,8}
- **Constraints:**
  - Qi != Qj (j != i)
  - |Qi – Qj| != |i – j|



EXAMPLE: CRYPT ARITHMATIC

- **Variables:** F T U W R O, X1 X2 X3
- **Domain:** {0,1,2,3,4,5,6,7,8,9}
- **Constraints:**
  - Alldiff (F,T,U,W,R,O)
  - O + O = R + 10 · X1
  - X1 + W + W = U + 10 · X2
  - X2 + T + T = O + 10 · X3
  - X3 = F, T ≠ 0, F ≠ 0

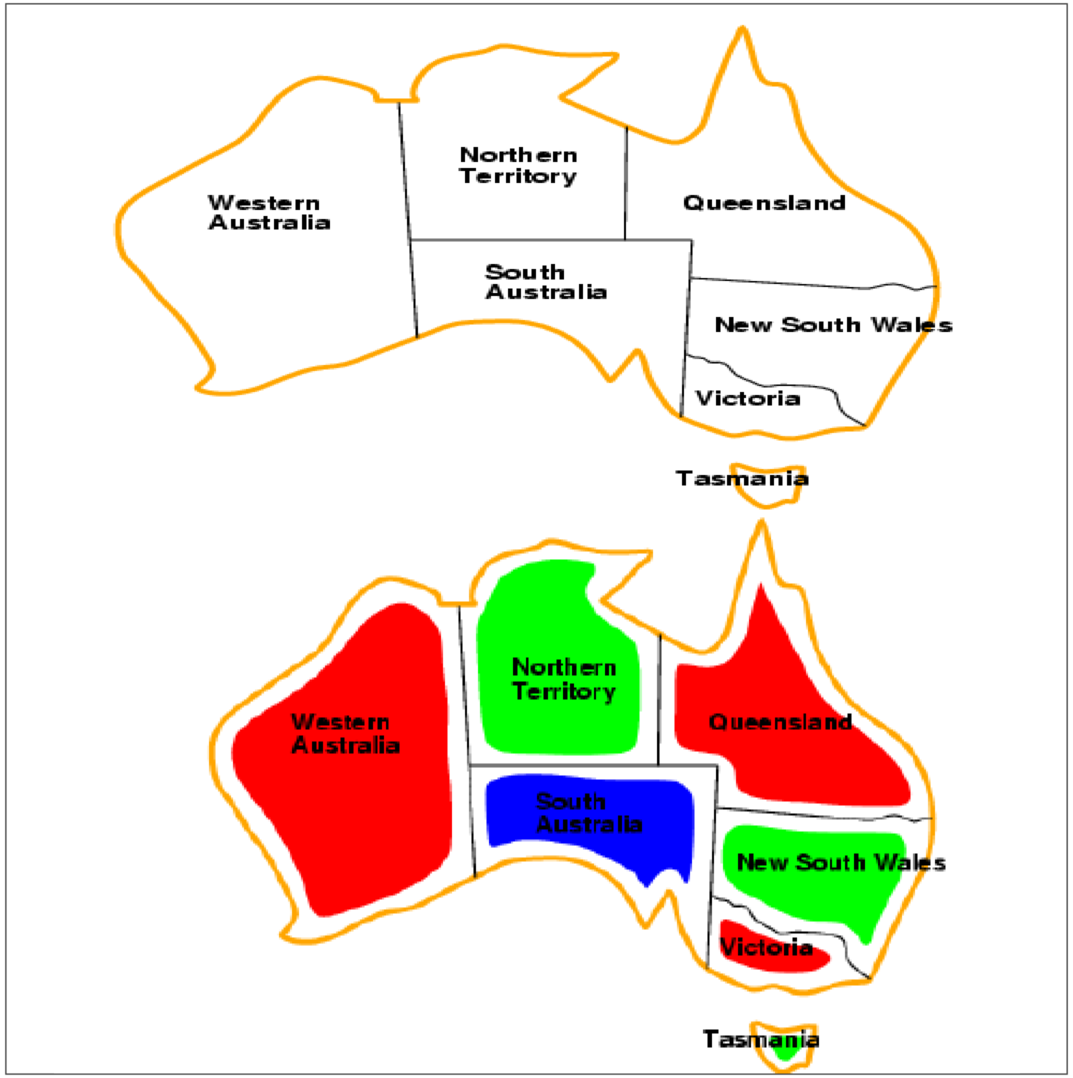
	1	2	3	4	5	6	7	8	9
A		6		1		4		5	
B			8	3		5	6		
C	2								1
D	8			4		7			6
E			6				3		
F	7			9		1			4
G	5								2
H			7	2		6	9		
I		4		5		8		7	

EXAMPLE: SUDOKU

- **Variables:**
  - A1, A2, A3, …, A7, A8, A9
  - Letters index rows, top to bottom
  - Digits index columns, left to right
- **Domains: The nine positive digits**
  - A1 --> {1, 2, 3, 4, 5, 6, 7, 8, 9}
- **Constraints:**
  - *Alldiff*(A1, A2, A3, A4, A5, A6, A7, A8, A9)

GRAPH COLORING PROBLEM TYPES

- There are 2 kinds of graph coloring problems
  - **M-colorability Decision problem:** Graph and colors given you should tell in how many possible ways the graph can be colored with no 2 adjacent vertices having same color with the given ‘m’ set of colors.
  - **M-colorability Optimization problem:** Graph is given colors are not given you should be able to find what is the minimum number ‘m’ of colors required to color the graph in such a way that no 2 adjacent vertices having same color.
- The smallest integer m with which a graph G can be colored in such a way that no two adjacent vertices have same color. M is referred to as **Chromatic number of the graph**. Its known to be chromatic number problem.
- For a **planar graph**(if a graph can be drawn in a plane in such a way no 2 edges cross each other) chromatic number can't be more than **4** (**Four color theorem**).
- In a connected graph in which every vertex has at most Δ neighbors, the vertices can be colored with only Δ colors, except for two cases, complete graphs and cycle graphs of odd length, which require $\Delta + 1$ colors.(Brook’s Theorem)

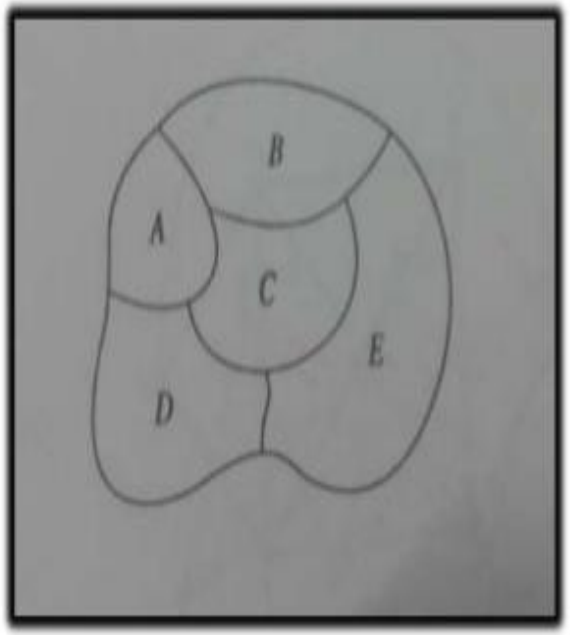


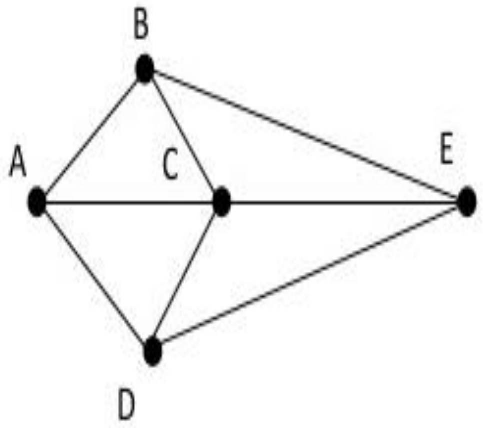
EXAMPLE: MAP COLORING PROBLEM

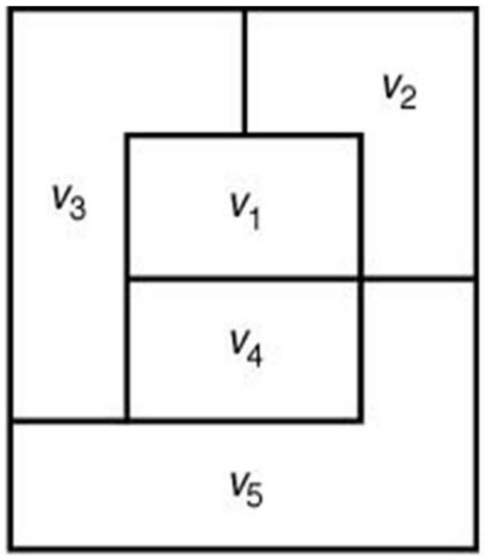
- **Variables:** ***WA, NT, Q, NSW, V, SA, T***
- **Domains:** D_i **= {red,green,blue}**
- **Constraints: adjacent regions must have different colors**
  - e.g., WA ≠ NT
    - So (WA,NT) must be in {(red,green),(red,blue),(green,red), …}

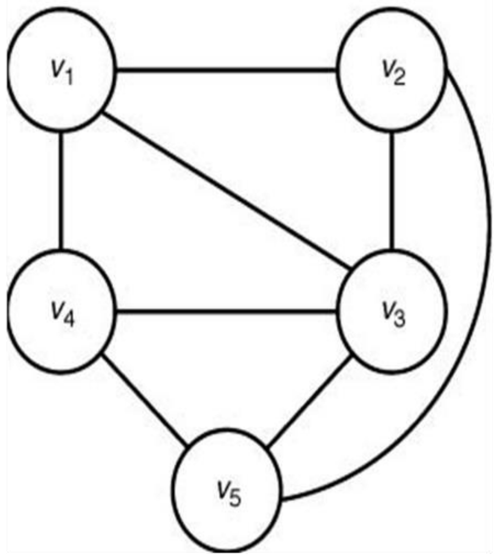
MAP COLORING PROBLEM

- A map can be transformed into a constraint graph by representing each region of map into a node and if two regions are adjacent, then the corresponding nodes are joined by an edge.
- Given graph can be colored using 3 colors



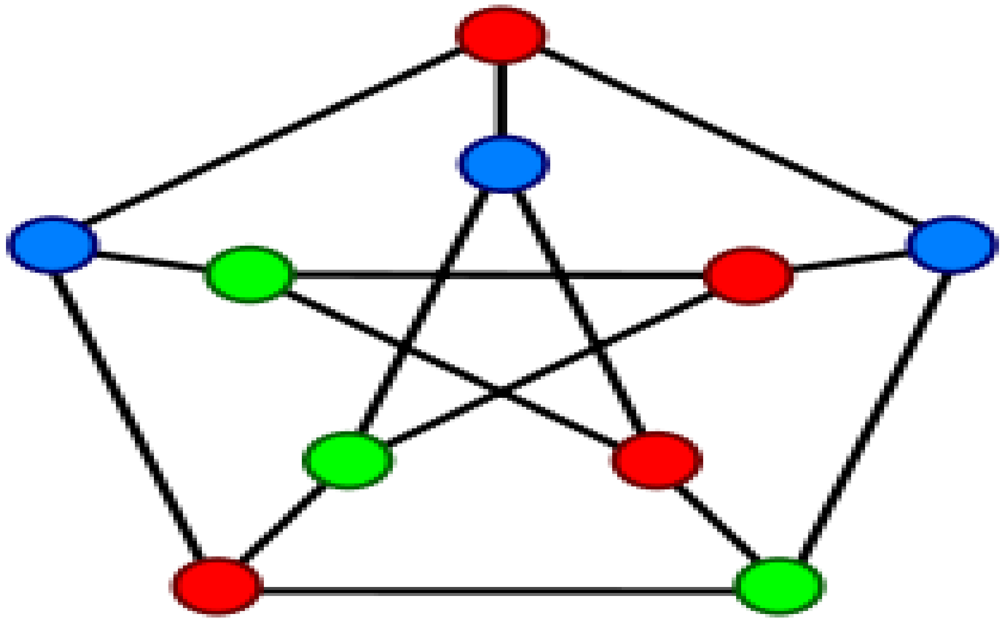


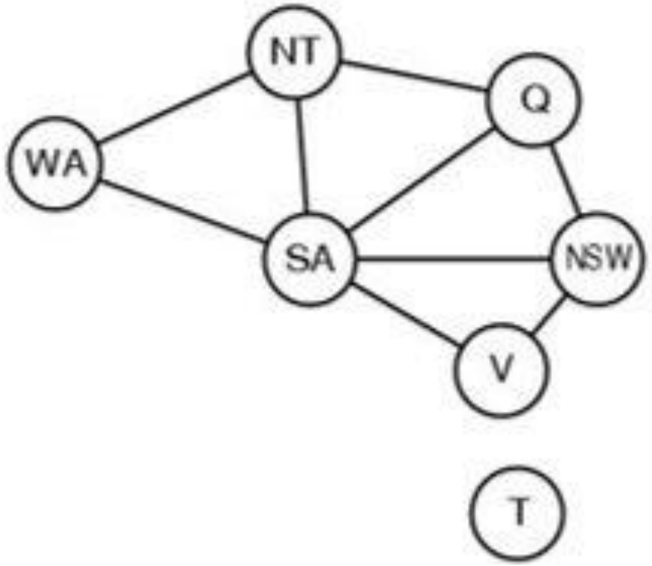




CONSTRAINT GRAPH

- *Constraint Graph* helps to visualize *CSP.*
  - *nodes* are variables
  - *arcs* are constraints or link that connects 2 variables that participate in a constraint.





CONSTRAINT TYPES

- *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
- *Preference* **(soft constraints) e.g.** *red* is better than *green* **can be represented by a cost for each variable assignment => ** Constrained optimization **problems.**

VARIABLE TYPES

- *Discrete variables*
  - **Finite domains:**
    - *n* variables, domain size *d* --> $O(d^n)$ complete assignments
    - e.g., Boolean CSPs, includes 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., $StartJob_1 + 5 \leq StartJob_3$
- *Continuous variables*
  - e.g., start/end times for Hubble Space Telescope observations
  - linear constraints solvable in polynomial time by linear programming

REAL-WORLD CSPS

- Assignment problems: e.g., who teaches what class
- Timetabling problems: e.g., which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Circuit layout
- Fault diagnosis

HOW TO SOLVE CONSTRAINT SATISFACTION PROBLEM

- A CSP can easily be expressed as a **standard search problem.**
- The main idea is to exploit the constraints to **eliminate large portions of search space.**

SOLUTION:

- **Incremental formulation**
    - Assign a value to an unassigned variable provided that it does not violate a constraint
    - End up with $n!*d^n$ leafs even though there are only d^n complete assignments.
- **Backtracking Search** over Assignments
    - **Depth- first search** that choses **values for one variable at a time** and **backtracks when a variable has no legal values left to assign.**
    - **Advantage: Solve CSP efficiently even without domain specific knowledge.**

BACKTRACKING SEARCH

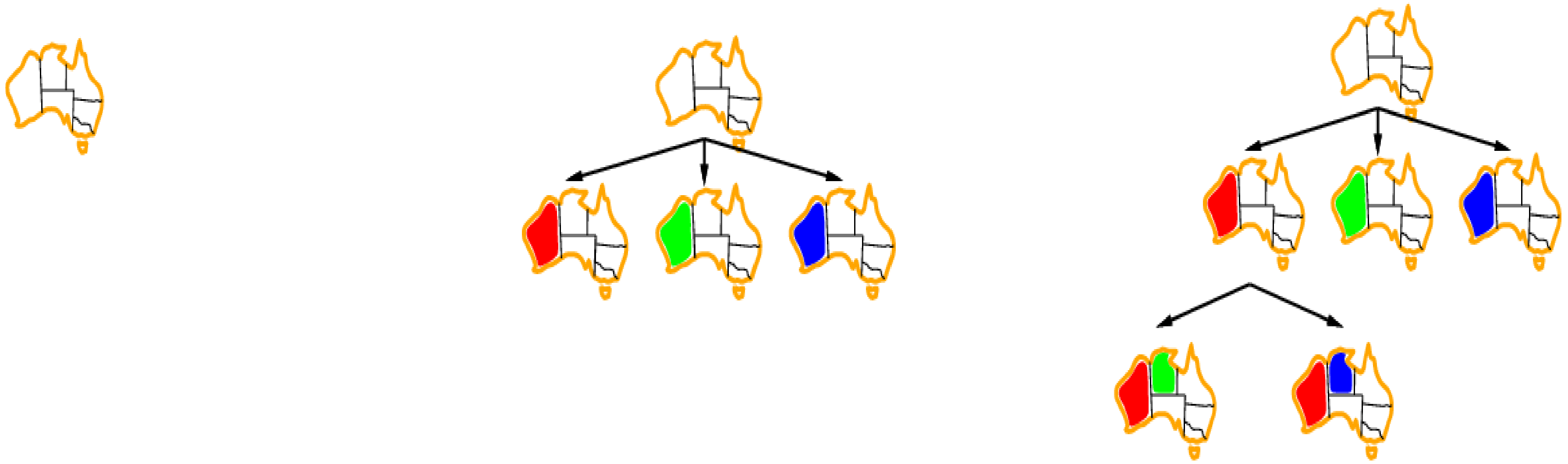
- **Intelligent Backtracking = DFS + variable-ordering + fail-on-violation**
- A **depth-first search** that chooses values for **one variable at a time** and **backtracks** when **a variable has no legal values left to assign**.
- When a **node** is expanded, check that **each successor state is consistent before adding it to the queue.** If its **not consistent or legal** **go back to the previous legal state** and **start solving by selecting remaining domaining values**.

BACKTRACKING SEARCH

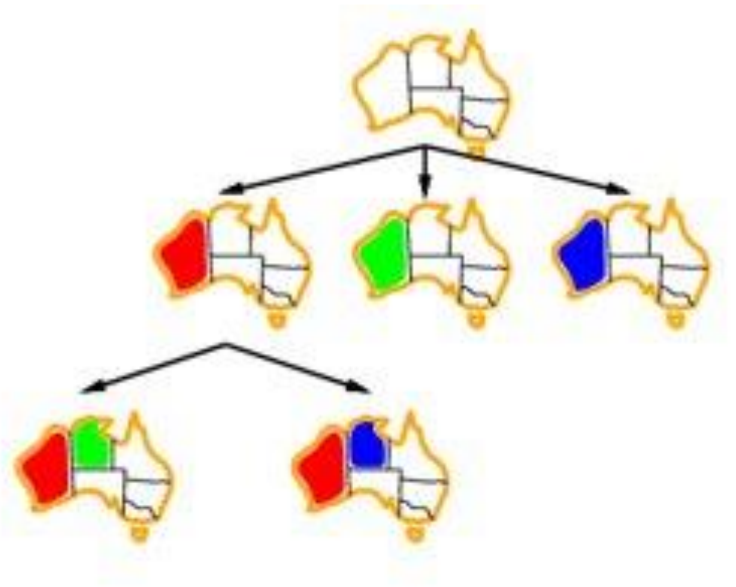
function BACKTRACKING-SEARCH(*csp*) **returns** solution/failure
    return RECURSIVE-BACKTRACKING({ }, *csp*)

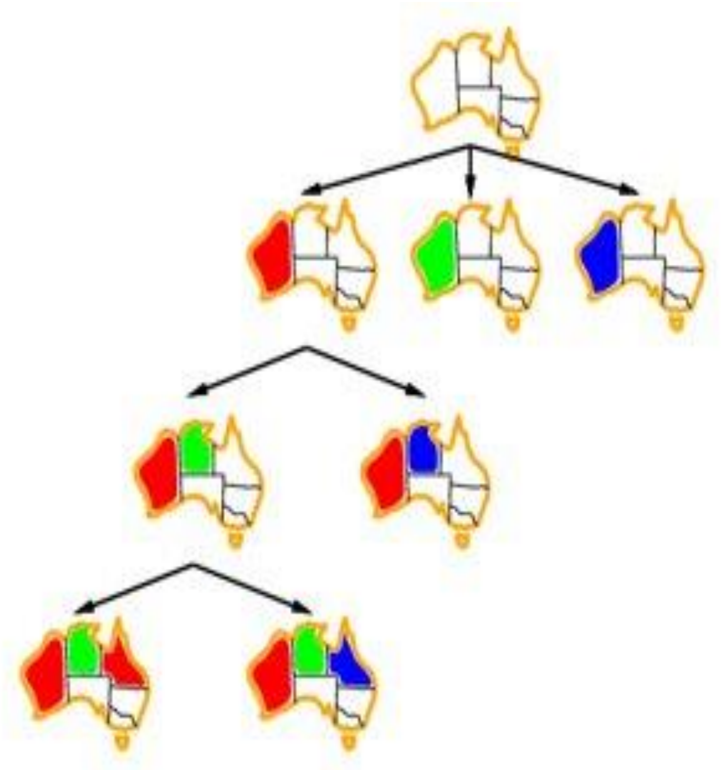
function RECURSIVE-BACKTRACKING(*assignment*, *csp*) **returns** soln/failure
    if *assignment* is complete **then return** *assignment*
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*], *assignment*, *csp*)
    for each *value* **in** ORDER-DOMAIN-VALUES(*var*, *assignment*, *csp*) **do**
        if *value* is consistent with *assignment* given CONSTRAINTS[*csp*] **then**
            add $\{var = value\}$ to *assignment*
            result ← RECURSIVE-BACKTRACKING(*assignment*, *csp*)
            if $result \neq failure$ **then return** *result*
            remove $\{var = value\}$ from *assignment*
    return *failure*

BACKTRACKING SEARCH FOR MAP COLORING PROBLEM



BACKTRACKING SEARCH FOR MAP COLORING PROBLEM



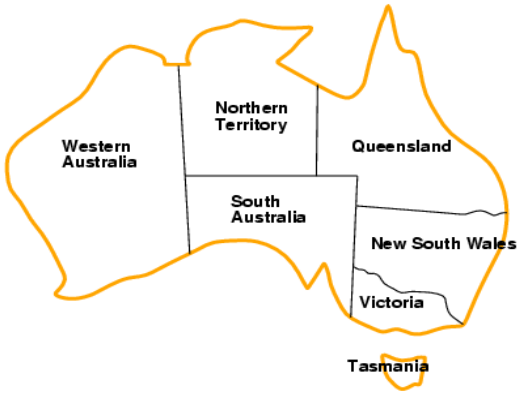


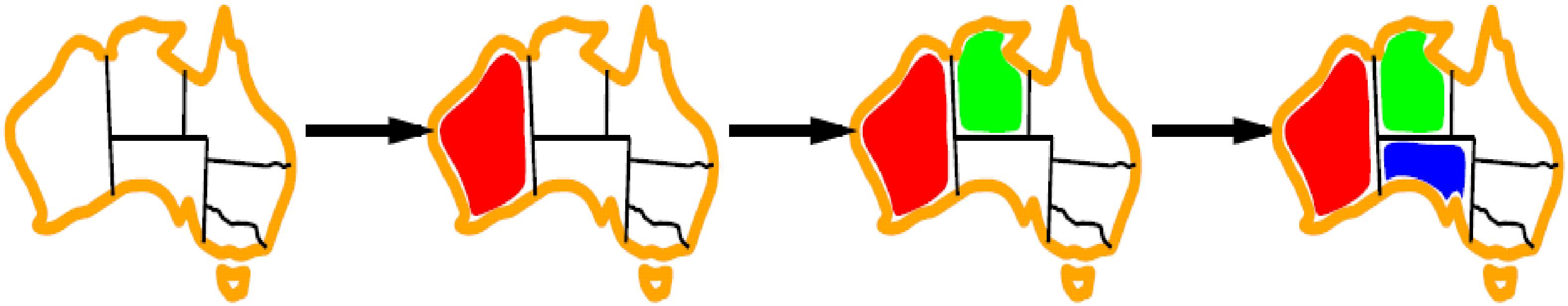
IMPROVING BACKTRACKING EFFICIENCY: CSP HEURISTICS & PRUNING TECHNIQUES

- **In what order should variable should be assigned next ? In which order variable's value should be tried next?**
- **Variable ordering** and **value selection heuristics** help significantly.
- **Variable ordering:** Which variable should be assigned next?
    - **Most constrained variable**
        - **Minimum Remaining value Heuristics**
    - **Most constraining variable**
        - **Degree heuristics**
- **Value ordering:**
    - **Least constraining value**
- Can we **detect failures early**?
    - **Forward checking** prevents **assignments that guarantee later failure.**
- Can we exploit problem structure
    - **Arc Consistency**

VARIABLE ORDERING: MINIMUM REMAINING VALUES (MRV)

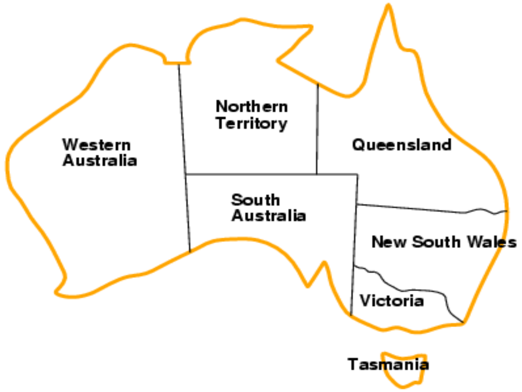
- Expand variables with **minimum size domain first.**
- The idea of choosing the variable with the **fewest "legal" value** or **"most constrained variable"** or **"fail-first" heuristic**, is to pick a variable that is most likely to cause a failure soon thereby pruning the search tree. If some variable X has no legal values left, the MRV heuristic will select X and failure will be detected immediately—avoiding pointless searches through other variables.
- Before the assignment to the rightmost state: one region has one remaining; one region has two; three regions have three. Choose the region with only one remaining

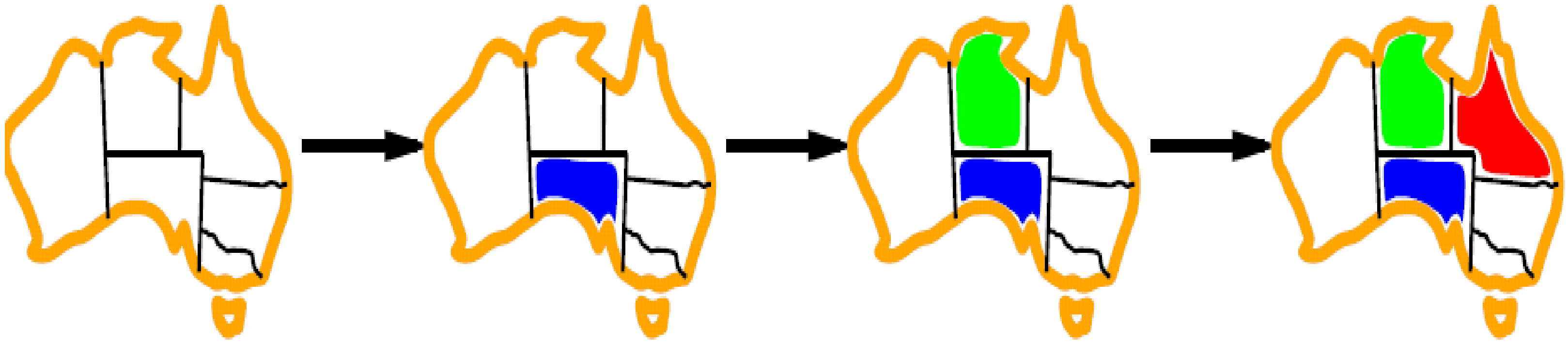




VARIABLE ORDERING: DEGREE HEURISTIC FOR RESOLVING TIES AMONG VARIABLES

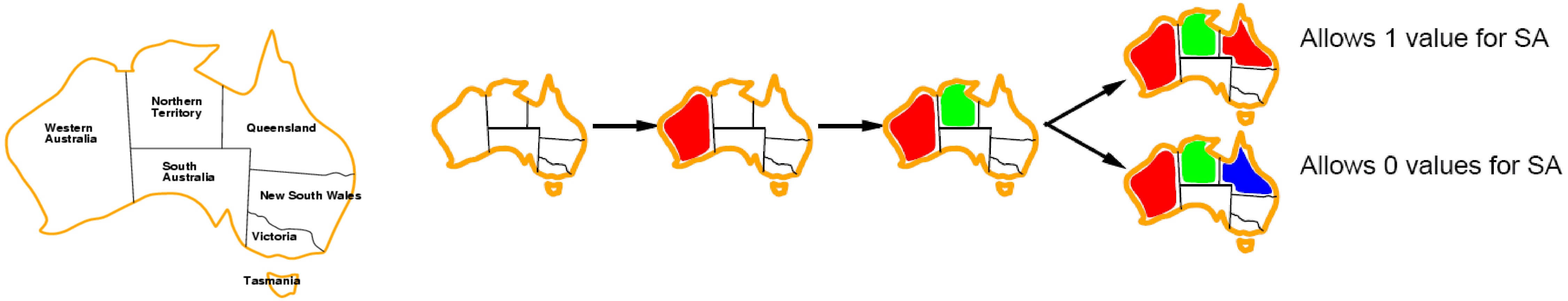
- **Degree heuristic can be useful as a tie breaker and reduce branching factor.**
- The degree heuristic attempts to reduce the branching factor on future choices by selecting the variable that is involved in the largest number of constraints on other unassigned variables
- **Before the assignment to the rightmost state, WA and Q have the same number of remaining values ({R}). So, choose the one adjacent to the most states. This will cut down on the number of legal successor states to it.**
- **Most constraining variable:** choose the variable with the most constraints on remaining variables





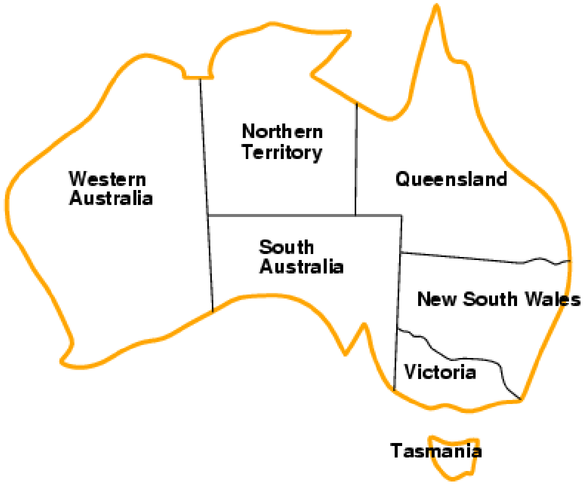
VALUE-ORDERING: LEAST CONSTRAINING VALUE

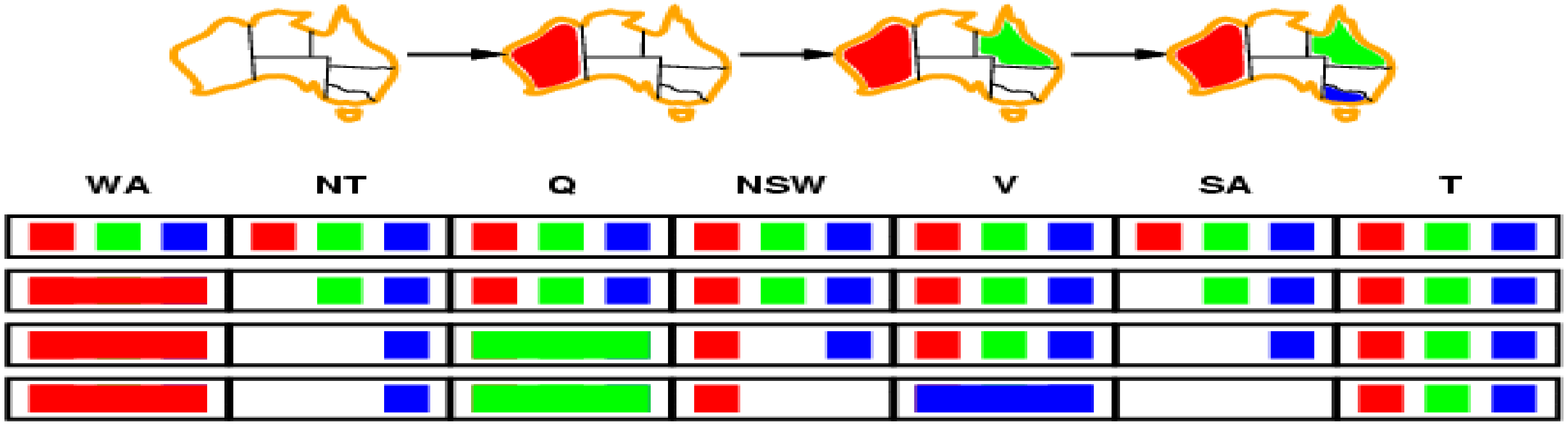
- Value ordering prefers the value that rules out the fewest choice for the neighboring variables in the remaining variables of constraint graph. **This leave the maximum flexibility for subsequent variable assignments.**



FORWARD CHECKING

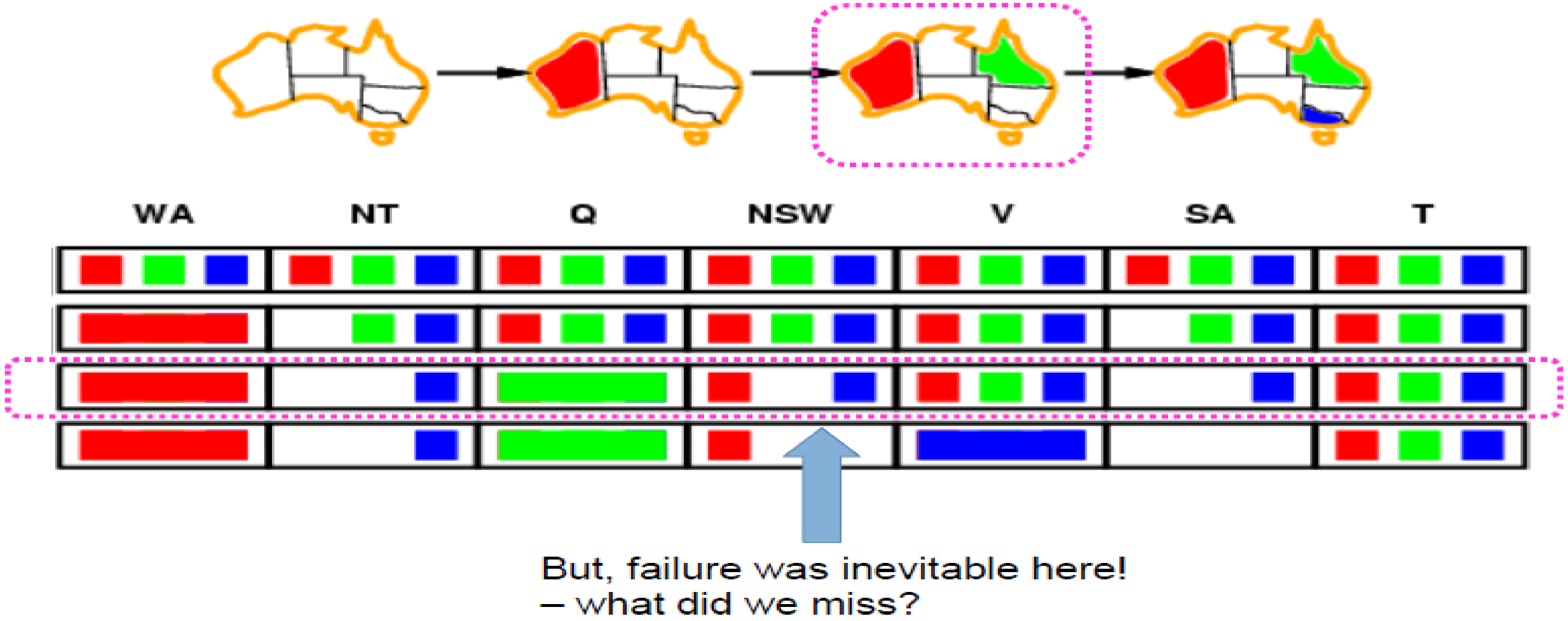
- Whenever a variable X is assigned, the forward-checking process establishes **arc consistency** for it: for each unassigned variable Y that is connected to X by a constraint, delete from Y's domain any value that is inconsistent with the value chosen for X.
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values





FORWARD CHECKING

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



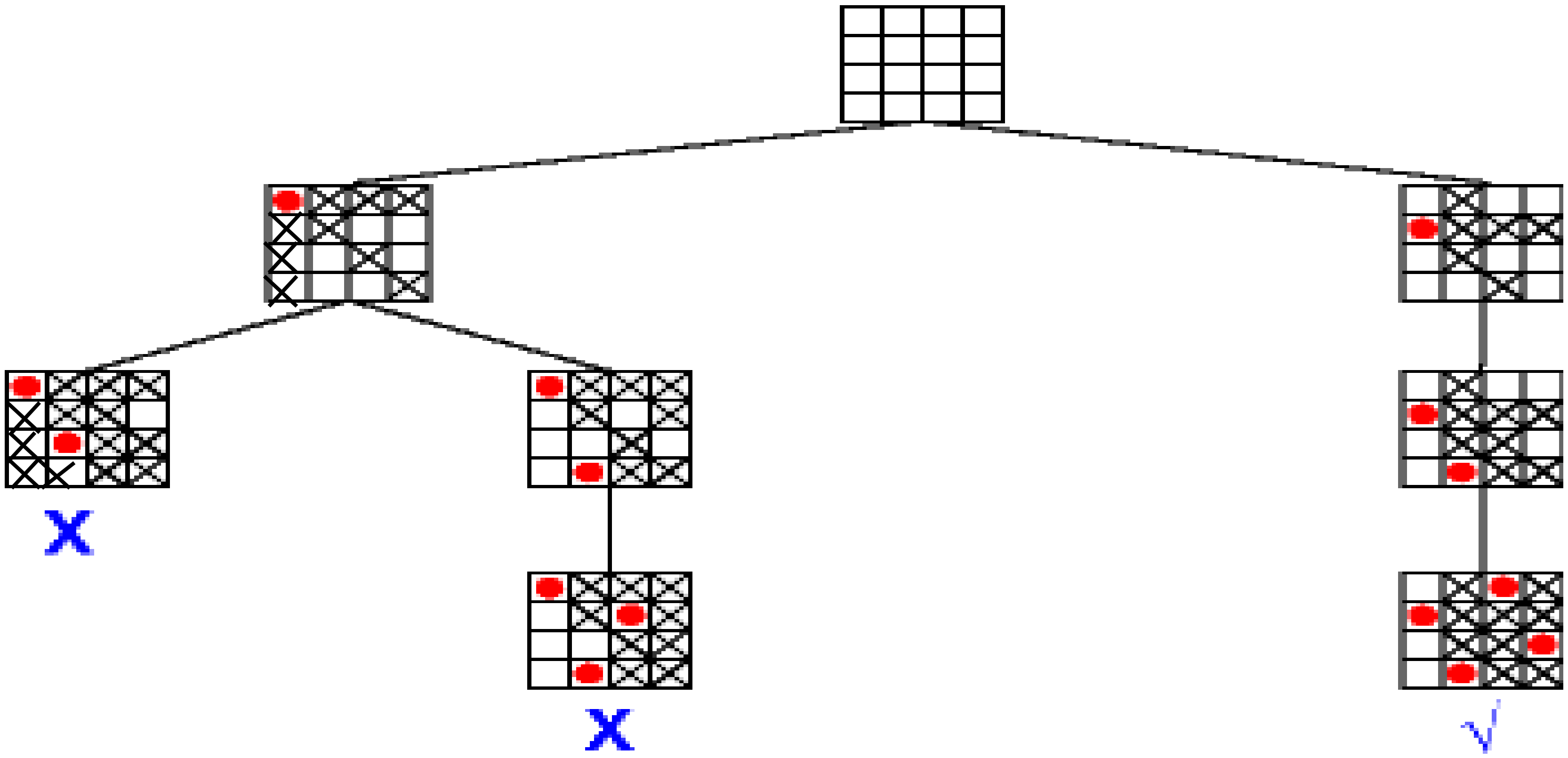
Constraint propagation repeatedly enforces constraints locally

FORWARD CHECKING

	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

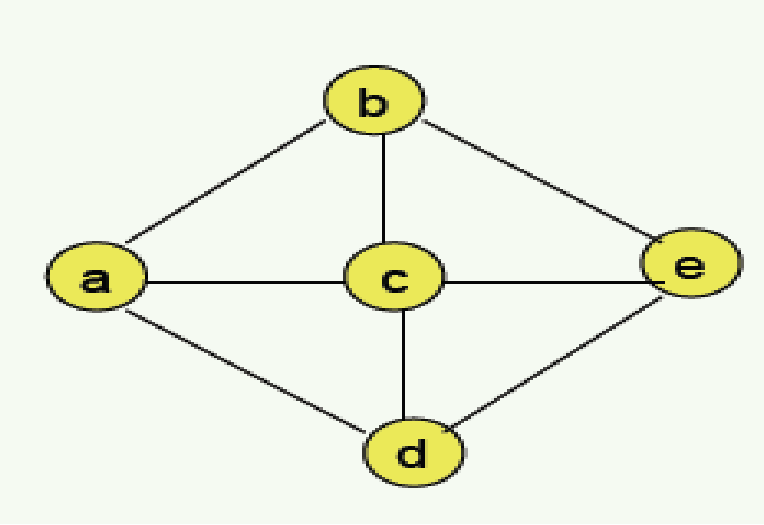
Figure 6.7 The progress of a map-coloring search with forward checking. $WA = red$ is assigned first; then forward checking deletes *red* from the domains of the neighboring variables NT and SA. After $Q = green$ is assigned, *green* is deleted from the domains of NT, SA, and NSW. After $V = blue$ is assigned, *blue* is deleted from the domains of NSW and SA, leaving SA with no legal values.

4-QUEENS PROBLEM AND FC



PROBLEM

- Find a solution for this CSP by using the following heuristics: minimum value heuristic, degree heuristic, forward checking., least constraining value Explain each step of your answer. The domain for every variable is [1,2,3,4]. There are 2 unary constraints:
  - variable "a" cannot take values 3 and 4.
  - variable "b" cannot take value 4.



SOLUTION

- There are 8 binary constraints stating that variables connected by an edge cannot have the same value.

    MVH--> a=1 (for example)

    FC+MVH -->b=2

    FC+MVH+MD -->c=3

    FC+LCV -->d=2

    FC -->e=1

CONSTRAINT PROPAGATION

- **Forward checking (FC) is in effect eliminating parts of the search space**
- **Constraint propagation goes further than FC by repeatedly enforcing constraints locally**
  - Needs to be faster than actually searching to be effective
- Constraint propagation is the process of communicating the domain reduction of a decision variable to all of the constraints that are stated over this variable.
- Constraint propagation is the process of communicating the domain reduction of a decision variable to all of the constraints that are stated over this variable.
- **Arc-consistency (AC) is a systematic procedure for constraint propagation**
- **Simplest form of propagation makes each** arch consistent

TYPES OF CONSISTENCY

- Maintaining NODE and ARC consistency further reduces the potential DOMAINS of variables, thereby reducing the amount of searching.
- **NODE CONSISTENCY (1-consistency)**
- a node is consistent if and only if all values in its domain satisfy all unary constraints on the corresponding variable. (note change here)
- Unary constraint contains only one variable, e.g., $x1 \neq R$
- **ARC CONSISTENCY (2-consistency)**
- an arc, or edge, ($xi \rightarrow xj$) in the constraint graph is consistent if and only if for every value "a" in the domain of xi , there is some value "b" in the domain of xj such that the assignment $\{xi , xj \} = \{a, b\}$ is permitted by the constraint between xi and xj .

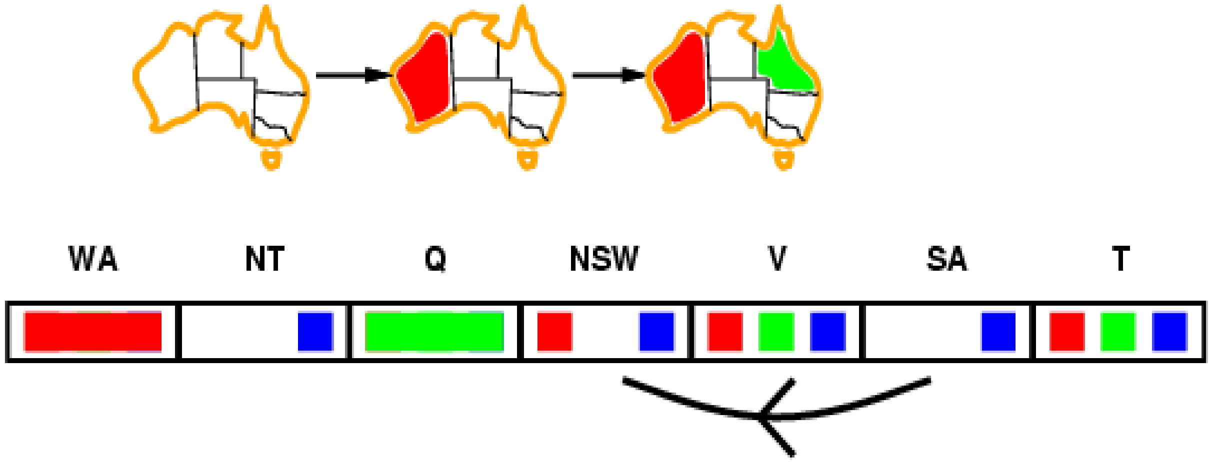
ARC CONSISTENCY CHECKING

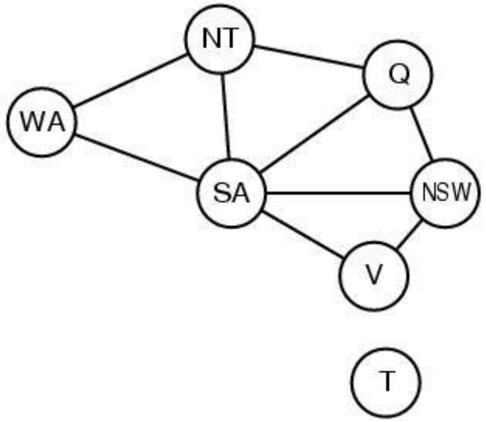
- **ARC must be run until no inconsistency remains**
- **Trade-off**
  - Requires some overhead to do, but generally more effective than direct search
  - In effect it can eliminate large (inconsistent) parts of the state space more effectively than search can

- **Need a systematic method for arc-checking**
  - If *X* loses a value, neighbors of *X* need to be rechecked:
  - **i.e. incoming arcs can become inconsistent again outgoing arcs will stay consistent).**

ARC CONSISTENCY

- ***Arc X -->Y* (link in constraint graph) is consistent iff** for every value *x* of *X* there is some allowed *y*. Delete values from tail in order to make each arc consistent
- **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**
- **Consider state of search after WA and Q are assigned:**
  - ***SA --> NSW* is consistent: if *SA=blue NSW could be =red***





ARC CONSISTENCY

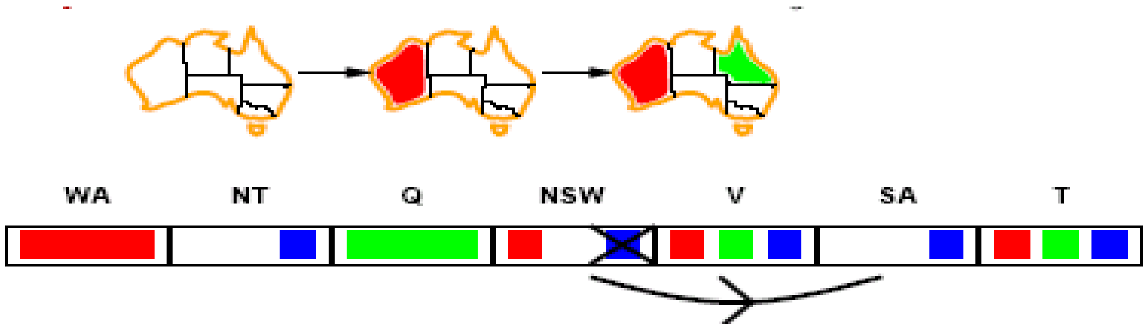
- **We will try to make the arc consistent by deleting x’s for which there is no y (and then check to see if anything else has been affected – algorithm is in a few slides)**
- ***NSW --> SA: if NSW=red SA could be =blue***

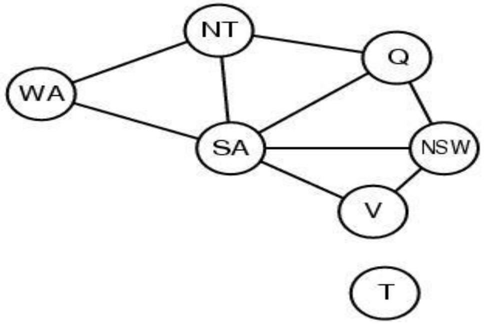
But, if NSW=blue, there is no color for SA.

So, remove blue from the domain of NSW

Propagate the constraint: need to check Q --> NSW SA --> NSW V --> NSW

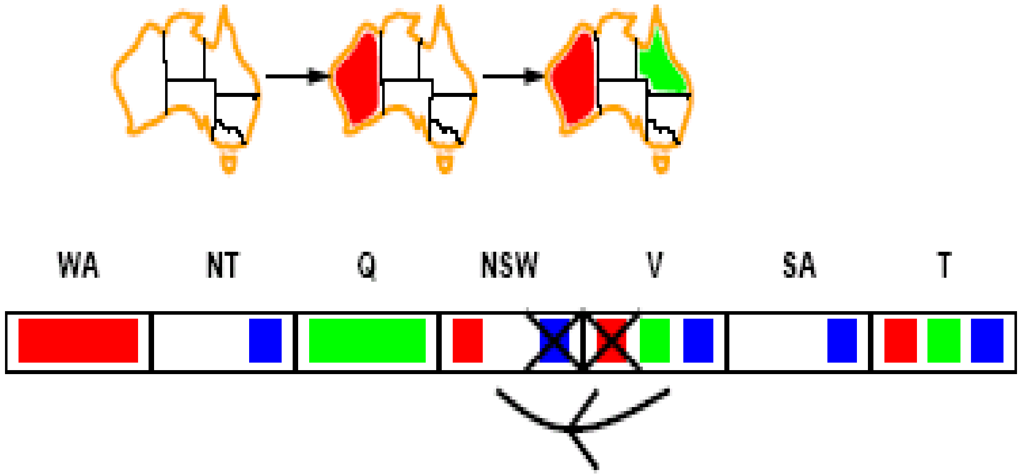
If we remove values from any of Q, SA, or V’s domains, we will need to check THEIR neighbors

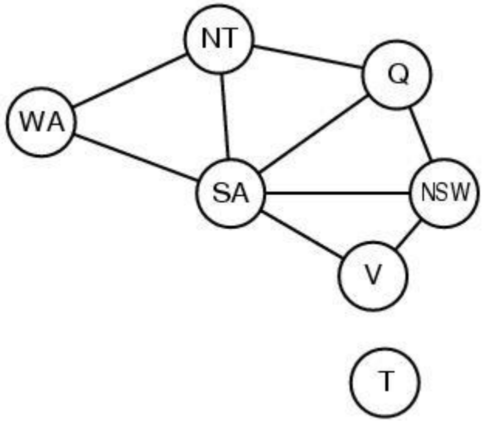




ARC CONSISTENCY

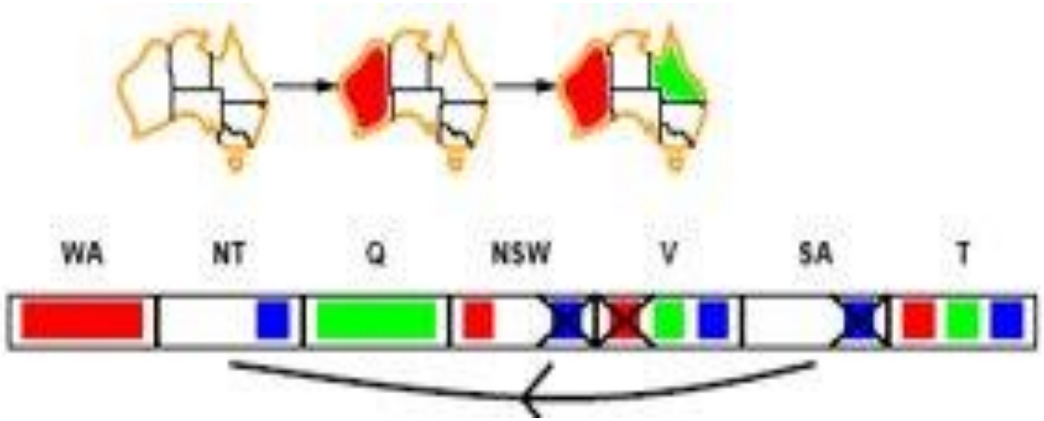
- After removing red from domain of V to make V --> NSW arc consistent
- *SA --> V, NSW --> V check out; no changes*
- *Check the remaining arcs: most check out, until we check SA --> NT, NT --> SA. Whichever is checked first will result in failure.*

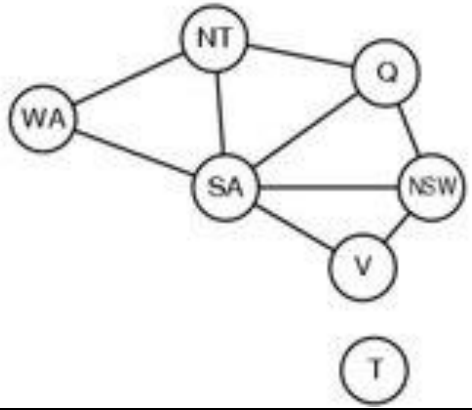




ARC CONSISTENCY

- ***SA --> NT* is not consistent** and cannot be made consistent
- **Arc consistency detects failure earlier than FC**
- **This process was all in one call to the INFERENCE function right after we assigned Q=green. Forward checking proceeded in the search, assigning a value to V.**





ARC CONSISTENCY ALGORITHM(AC-2)

function AC-2(*csp*) returns false if inconsistency found, else true, may reduce *csp* domains

  local variables: *queue,* a queue of arcs, initially all the arcs in *csp*

  while queue is not empty do /* *initial queue must contain both* (X_i, X_j) *and* (X_j, X_i) */

    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)

    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) then

      if size of $D_i = 0$ then return false

      for each X_k in NEIGHBORS[X_i] − {X_j} do

        add (X_k, X_i) to queue if not already there

  return true

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) returns *true* iff we delete a

    value from the domain of X_i

  removed ← *false*

  for each x in DOMAIN[X_i] do

    if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraints

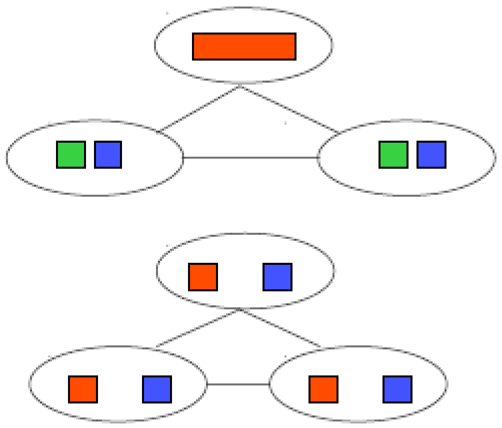
      between X_i and X_j

    then delete x from DOMAIN[X_i]; *removed* ← *true*

  return *removed*

ARC CONSISTENCY DOES NOT DETECT ALL INCONSISTENCIES

- After enforcing arc consistency:
- Can have one solution left
- Can have multiple solutions left
- Can have no solutions left (and not know it)
- Arc consistency still runs inside a backtracking search!



What went wrong here?

INTELLIGENT BACKTRACKING

- Chronological Backtracking At dead end, backup to the most recent variable.

Eg:{Q-red, NSW=green, V=blue, T=red}. For next variable SA no values are left out and backtracking to T cannot resolve problem with SA.

- Backjumping At dead end, backup to the most recent variable that eliminated some value in the domain of the dead end variable. Ie bactrack variable that fix problem
- Keep track of conflict set(set of assignments that ae inconflict with values

of SA. Eg:{Q-red, NSW=green, V=blue}

- Conflict-directed Backjumping A backjumping algorithm that uses

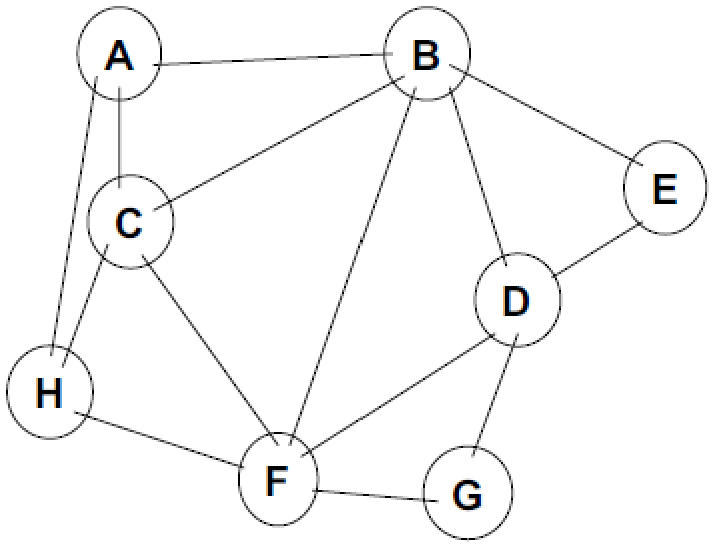
conflict sets to backtrack.ie backjumping occurs when current value in domain is

in conflict with current assignment.

- **Forward checking** supplies **conflict set**. **Backjumping** occurs when every value in a domain is in conflict with current assignment ie for one variable domain becomes empty.
- Constraint Learning is idea of finding minimum set of variables from conflict set that causes problem.

CHALLENGE

Find a solution for this CSP by using the following heuristics: minimum value heuristic, degree heuristic, forward checking., least constraining value Explain each step of your answer. The domain for every variable is [1,2,3,4].



Thank You

anooja@somaiya.edu