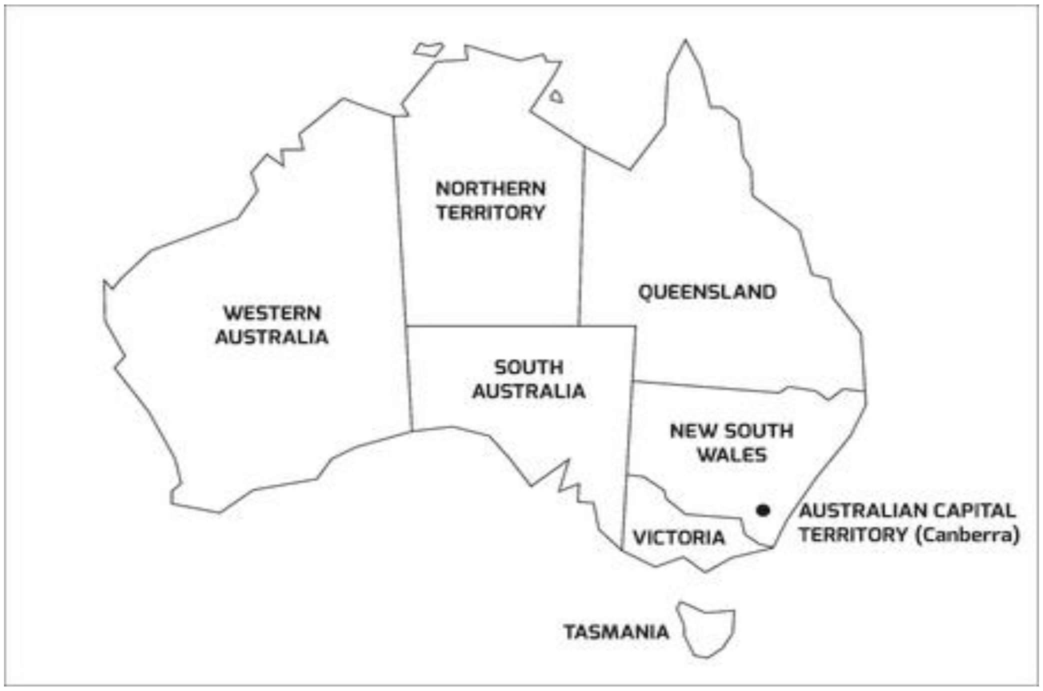
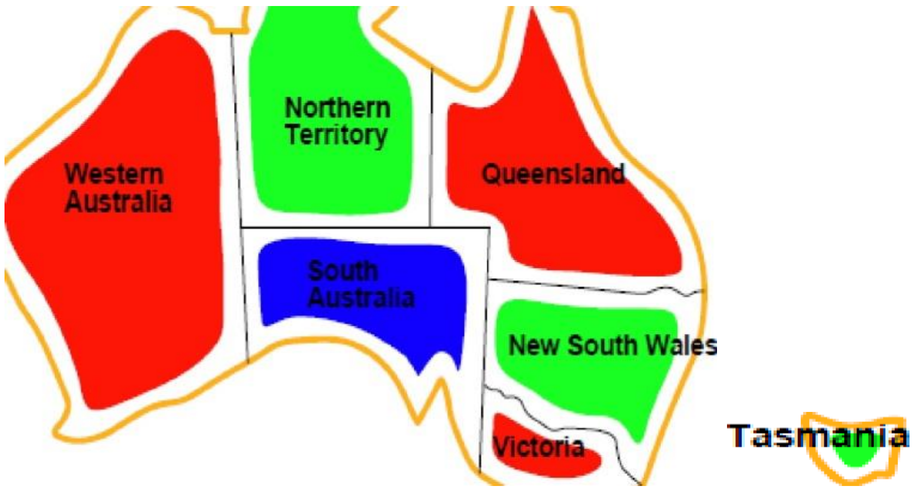
1. Graph Coloring: The problem where the constraint is that no adjacent sides can have the same color.



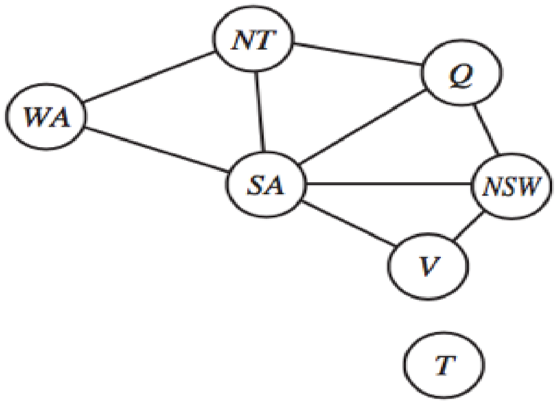
- Variables : *WA, NT , SA, Q, NSW, V, T*
- Domains: *(Red, green, blue)*
- Constraints: adjacent regions must have different colors
- Solutions :- satisfying all constraints,

e.g., *WA— **Red**, NT — **GREEN**, Q — **Red**, NSW' — Green, V-**Red,** SA — **Blue,** T — **Green***



Constraint Graph

Nodes = variables, **arcs =** constraints



Search Tree Generation for Graph Coloring:

Types of Constraints

1. Unary constraints involve a single variable,

e.g., SA ≠ green

2. **Binary constraints** involve pairs of variables, e.g.,

SA ≠ WA

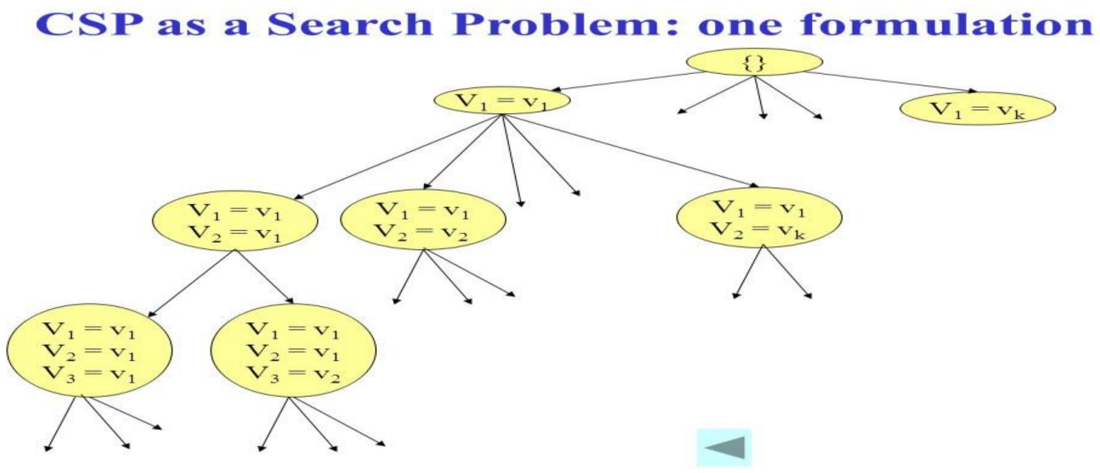
3. Higher-order constraints involve 3 or more variables

e.g., crypt arithmetic puzzles

CSP AS A SEARCH PROBLEM:-

Problem Formulation of CSP

1. **Initial state:** {} – It shows the empty assignment that means all variables are unassigned
2. **Successor function:** a value is assigned to one of the unassigned variables with no conflict
3. **Goal test:** It check whether the current assignment is completed or not.
4. **Path cost:** It is Cost for path to reach the goal state.


CSP as a Search Problem: one formulation
{}
$V_1 = v_1$
$V_1 = v_k$
$V_1 = v_1$ $V_2 = v_1$
$V_1 = v_1$ $V_2 = v_2$
$V_1 = v_1$ $V_2 = v_k$
$V_1 = v_1$ $V_2 = v_1$ $V_3 = v_1$
$V_1 = v_1$ $V_2 = v_1$ $V_3 = v_2$

No of children at Top Level: -	N*D
No of children at 2Level :-	(N-1)*D
No of children at 3 Level :-	(N-2)*D
Total Leaves:-	n!*D^N
Total no of Assignments:	D^N

Here N- number of variable, D- number of domain

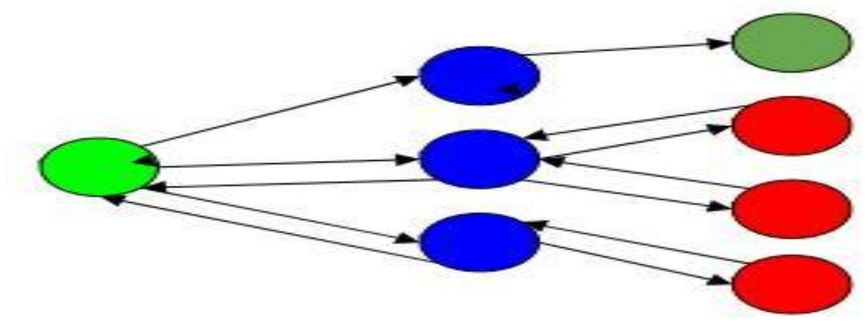
BACKTRACKING SEARCH FOR CSP

What is Backtracking?

The backtracking is used to find all possible combination to solve an optimization problem.

In backtracking problem, the algorithm tries to find a sequence path to the solution which has some small checkpoints from where the problem can backtrack if no feasible solution is found for the problem.

Example



Here,

Green is the start point, blue is the intermediate point, red are points with no feasible solution, **Dark green** is end solution.

Algorithm

```
Step 1 - if current position is goal, return success
Step 2 - else,
Step 3 - if current position is an end point, return failed.
Step 4 - else, if current position is not end point, explore and
repeat above steps.
```

Backtracking Search for CSP:-

- Depth-first search for CSPs with single-variable assignments is called backtracking search.
- These are special property of CSPs
- They are commutative that mean the order in which assign variable does not matter. A*B=B*A
- [WA = red then NT = green] same as [NT = green then WA = red]
- Only need to consider assignments to a single variable at each node

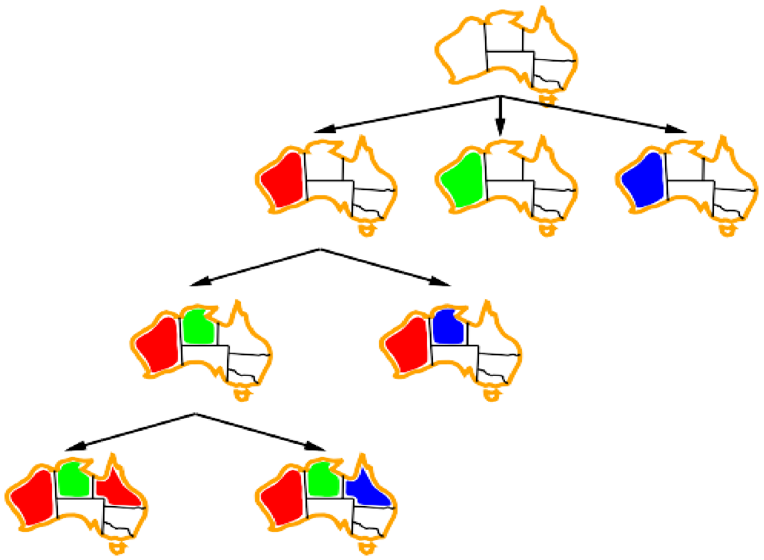
b = d

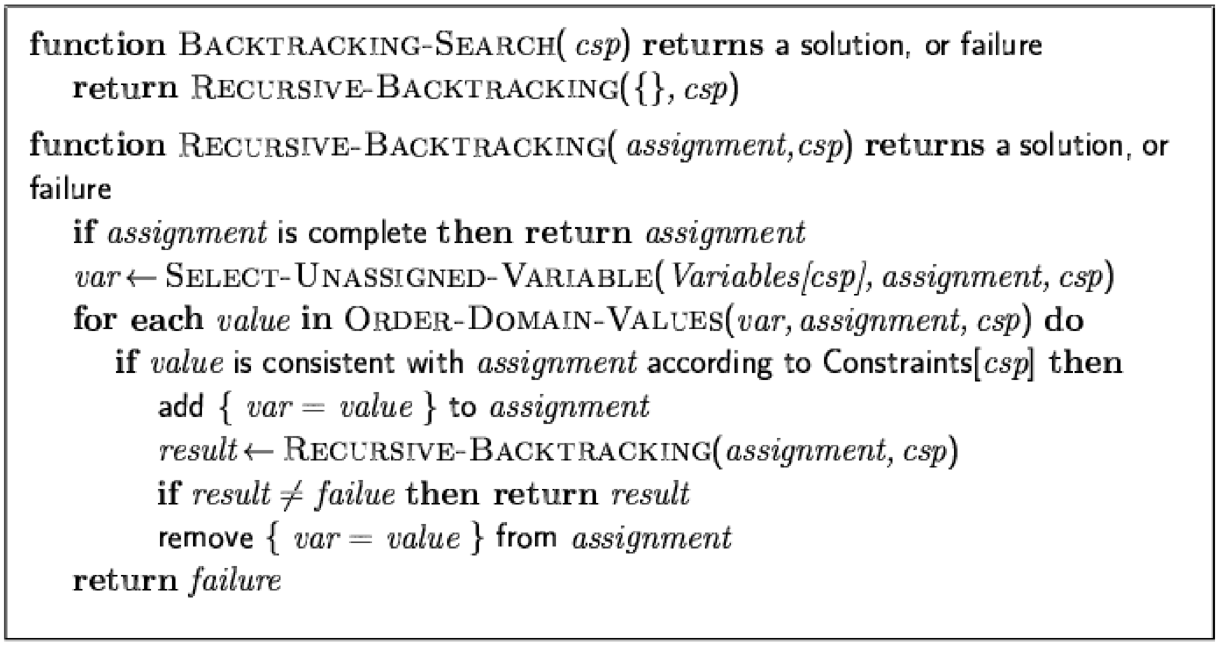
No of children at Top Level: - D

No of children at 2Level:- D^2

Total Leaves: - D^N

Can solve n-queens for $n \approx 25$





```
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 $\neq$ failue then return result
         remove { var = value } from assignment
   return failure
```

Procedure BackTracking(Var,Con)

pick initial state

R=set of all possible states

Select state with var assigenment

Add to search space

Check for con

If satisfied

Continue

Else

Go to last decision point(DP)

Prune(Reduce) the search sub-space form DP

Continue with next decision option

If state=Goal state

return solution

Else

Continue.

Back tracking Solve the Following Types of Problems

- M-Coloring Problem
- N Queen Problem
- Crypt arithmetic Puzzle
- Sudoku Solving Algorithm

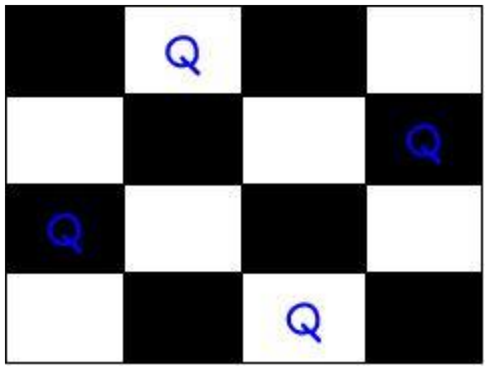
N-Queen Problem.

Example- 4-Queen problem.

In N-Queen problem, we are given an NxN chessboard and we have to place n queens on the board in such a way that no two queens attack each other.

A queen will attack another queen if it is placed in horizontal, vertical or diagonal points in its way.

Here, the solution is −



Here, the binary output for n queen problem with 1’s as queens to the positions are placed.

```
{0 , 1 , 0 , 0}
{0 , 0 , 0 , 1}
{1 , 0 , 0 , 0}
{0 , 0 , 1 , 0}
```

Backtracking Algorithm

- The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens.
- In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

Backtracking Algorithm

```
1) Start in the leftmost column
2) If all queens are placed
    return true
3) Try all rows in the current column.
   Do following for every tried row.
    a) If the queen can be placed safely in this row
       then mark this [row, column] as part of the
       solution and recursively check if placing
       queen here leads to a solution.
    b) If placing the queen in [row, column] leads to
       a solution then return true.
    c) If placing queen doesn't lead to a solution then
       unmark this [row, column] (Backtrack) and go to
       step (a) to try other rows.
3) If all rows have been tried and nothing worked,
   return false to trigger backtracking.
```

Improving backtracking efficiency

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect unavoidable failure early?

Heuristics can be used to improve the performance of a backtracking solver.

Role of Heuristic

- The objective of a heuristic is to produce a solution in a reasonable time frame that is good enough for solving the problem at hand.
- A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.

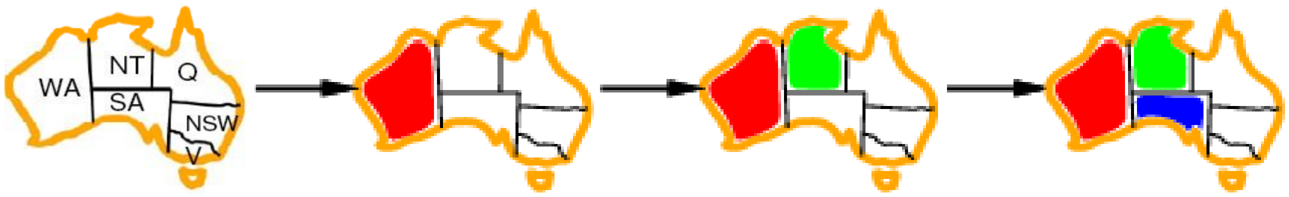
Three commonly given heuristics for simple backtracking solvers are:

- **Minimum-remaining-values** (how many values are still valid for this variable)
- **Degree heuristic** (how many other variables are affected by this variable)
- **Least-constraining-value** (which variable will leave the most other values for other variables)

1. Which variable should be assigned next?

Minimum-remaining-values (MRV) heuristic:

- Heuristic used along with Backtracking algorithm to solve Constraint Satisfaction Problems such as the Graph Coloring Problem.
- It is also known as **most constrained variable”** or “fail-first” heuristic.
- The idea of choosing the variable with the **fewest** “legal” value.
- Choose the variable that has the **least values left in its domain,**

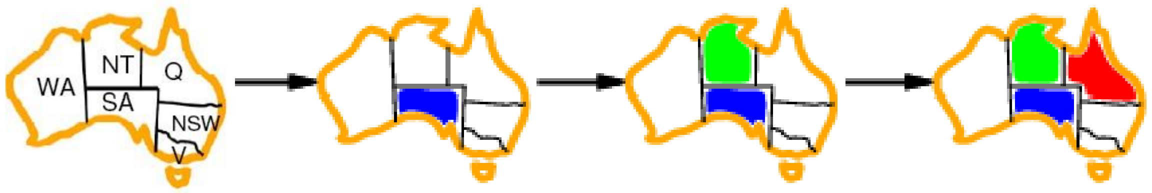


E.g. After the assignment for WA=red and NT=green, there is only one possible value for SA, so it makes sense to assign SA=blue next rather than assigning Q.

Degree heuristic:

- Selecting the variable that has the largest number of constraints on other unassigned variables is called **degree heuristics.**
- Choose the one that affects the most other variables
- It is Tie-breaker among most constrained variables

Note:- The **degree** of a **vertex** is the number of edges connecting it

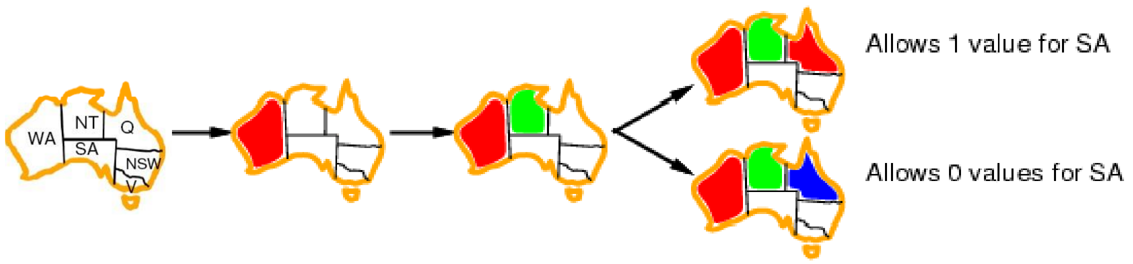


Example: NT and SA have same **MRV** so it select variable which have highest **Degree heuristic.**

e.g. SA is the variable with highest degree 5;
The other variables have degree 2 or 3;
T has degree 0.

2. For a given variable In What Order Should Its Values Be Tried?

LEAST CONSTRAINING VALUE: for a given a variable, choose the least constraining value: the one that rules out the fewest values in the remaining variables.



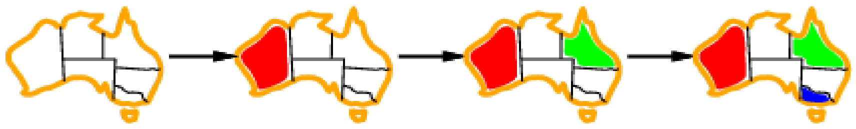
Try to leave the maximum flexibility for subsequent variable assignments

e.g. We have generated the partial assignment with WA=red and NT=green and that our next choice is for Q**. Blue would be a bad choice** because it eliminates the last legal value left for Q's neighbor, **SA**, therefore prefers red to blue.

3. Can we detect unavoidable failure early?

Forward Checking:-

- Forward checking keeps track of next move that are available for the unassigned variables.
- It Keep track of remaining legal values for unassigned variables
- Whenever a value is assigned to a variable, values that are now illegal for other variables are removed.
- Terminate search when any variable has no legal values.



WA	NT	Q	NSW	V	SA	T
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦
🟥	🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟩 🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥 🟦	🟥 🟩 🟦	🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥	🟦		🟥 🟩 🟦

In the above diagram we can say that 'SA' has not any legal value there is not any color for it. And we can terminate the search

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

WA	NT	Q	NSW	V	SA	T
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦
🟥	🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟩 🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥 🟦	🟥 🟩 🟦	🟦	🟥 🟩 🟦

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

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.

There are different types of local consistency:

Node consistency

A single variable (a node in the CSP network) is **node-consistent** if all the values in the variable's domain satisfy the variable's **unary constraint.**

We say that a network is node-consistent if every variable in the network is node-consistent.

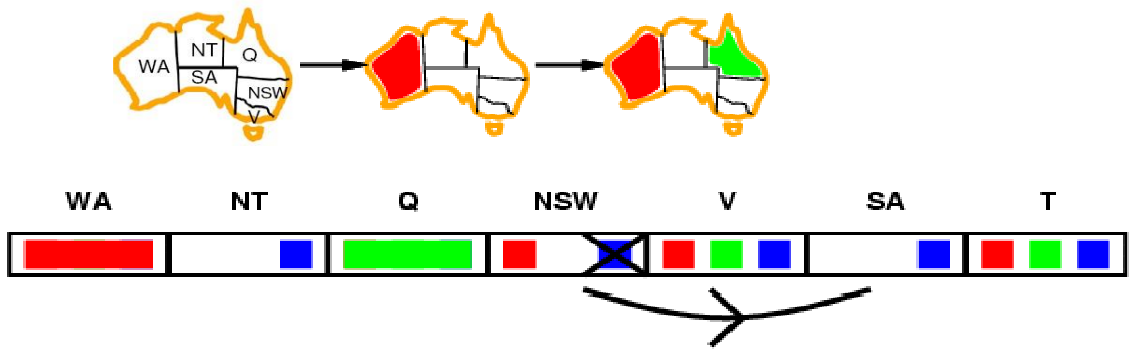
Arc consistency

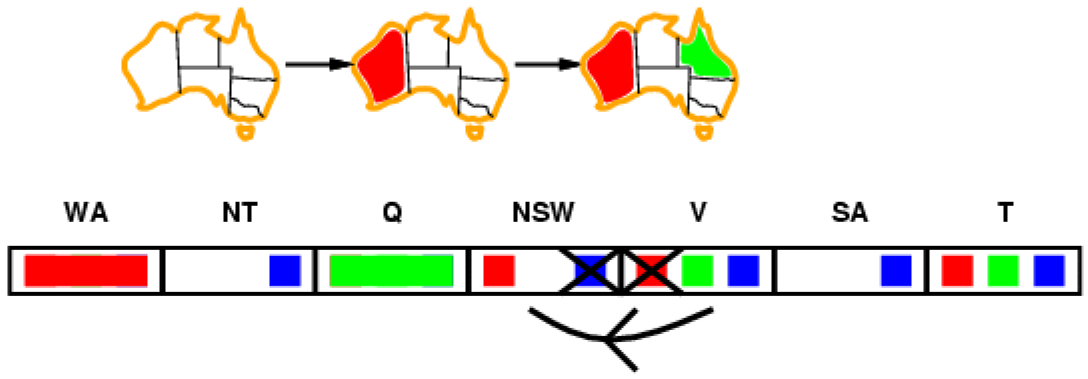
A variable in a CSP is **arc-consistent** if every value in its domain satisfies the variable's **binary constraints.**

A directed arc *X* →*Y* is consistent iff
for every value x of X, there exists a value y of Y , such that (x; y) satisfies the constraint between X and Y

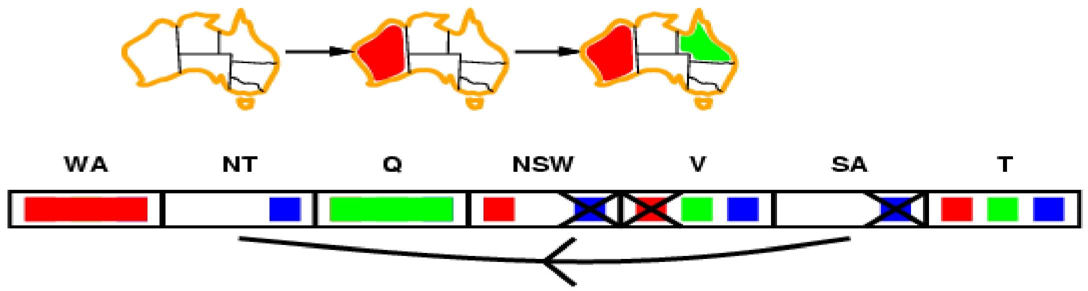
A network is arc-consistent if every variable is arc-consistent with every other variable.

- Remove values from the domain of X to enforce arc-consistency
- Arc consistency detects failures earlier





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



Intelligent Backtracking:-

In the intelligent backtracking we backtrack to the actual node or the variable that has caused the termination of current path or where conflict occurs.

	1	2	3	4
Initial Domain	**RGB**	**RGB**	**RGB**	**RGB**
1=R	**R**	**GB**	**GB**	**GB**
2=G	**R**	**G**	**GB**	**GB**
3=B	**R**	**G**	**B**	**conflict for Blue**
After backtracking				
3=G	**R**	**G**	**B**	**GB**
4=B	**R**	**G**	**G**	**B**