# Constraint Satisfaction Problem

Chapter#6

Instructor: Engr. Saeeda Kanwal

# Constraint Satisfaction Problem

- So far, states evaluated by heuristics and goal

- Constraint satisfaction offers a powerful problem-solving paradigm
  - View a problem as a **set of variables** to which we have to assign **values** that satisfy a number of **problem-specific constraints**.

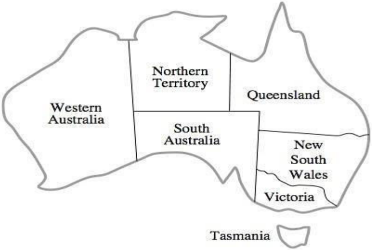
# CSPs Formulation

- $X$, a set of variables, $\{X_1, \ldots, X_n\}$
- $D$, a set of **domains** for each $X$. $\{D_1, \ldots, D_n\}$
- $C$, a set of constraints

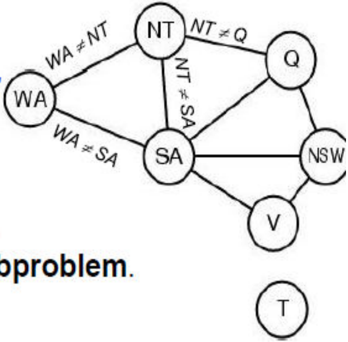
# Constraint Satisfaction Probelm
## Example-Map Coloring


- Color each region either red,green or blue
- No adjacent region can have the same color

# CSP Representation




- ***Constraint graph:***
  - *nodes* are variables
  - *arcs* are constraints

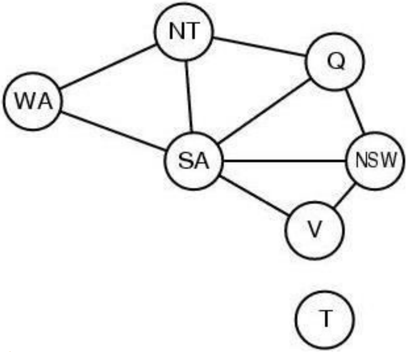



- ***Standard representation pattern:***
  - variables with values

- ***Constraint graph*** **simplifies search.**
  - **e.g. Tasmania is an independent subproblem.**

- ***Variables***: (WA,NT,Q,SA,NSW,V,T)

# Constraint Satisfaction Problems
## CSPs

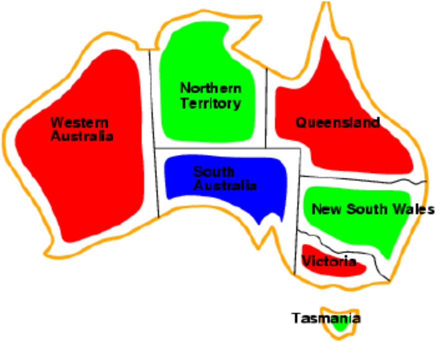




- **Domains:** $D_i$ = {red,green,blue}
- **Constraints: adjacent regions must have different colors**
  - e.g., WA ≠ NT
    —So (WA,NT) must be in {(red,green),(red,blue),(green,red), …}

▲ $C = \{ (\forall X_i, X_j \text{ such that } X_i \text{ touches } X_j), (Color(X_i) \neq Color(X_j))\}$

# Constraint Satisfaction Problems
CSPs

CSP is defined by the assignment of values to variables in State Space




**Solutions are complete and consistent assignments,**

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

# CSPs
## Scheduling Classes

Can you formulate it in terms of variables, domains and constraints?

- $X = \{ CS235, CS355, CS101, etc.\}$
- $D = \{ Mon9am, Mon11am, Mon1pm \ldots Fri4pm, Fri6pm\}$
- $C =$ $\forall i, j$ if $x_i, x_j$ are co-requisites, then $Ti \neq Tj$

# Variations on thee CSP Formulization

- ***Discrete variables***
  - finite domains:
    - —$n$ variables, domain size $d \rightarrow O(d^n)$ complete assignments
    - —
    - — Map-Coloring, 8 Queens Problem
  - 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

# CSPs Formally
## Kinds of Constraints

- **Unary Constraint**: Involve a single variable (*SA ≠ green*)
- **Binary Constraint** : Involve a pair of variables (*SA ≠ WA*)
- **Higher Order (Global Constraint)**: Involve 3 or more (Sudoku, Crypt-arithmetic)

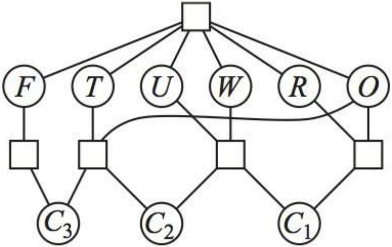
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| A | | | 3 | | 2 | | 6 | | |
| B | 9 | | | 3 | | 5 | | | 1 |
| C | | | 1 | 8 | | 6 | 4 | | |
| D | | | 8 | 1 | | 2 | 9 | | |
| E | 7 | | | | | | | | 8 |
| F | | | 6 | 7 | | 8 | 2 | | |
| G | | | 2 | 6 | | 9 | 5 | | |
|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

Constraint kind: *AllDiff*

27 *AllDiff*s

# CSPs
## Cryptarithmetic Puzzles

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




Constraints:

- $AllDiff(F, T, U, W, R, O)$
- $O + O = R + 10 \times C_{10}$
- $C_{10} + W + W = U + 10 \times C_{100}$
- $C_{100} + T + T = O + 10 \times C_{1000}$
- $C_{1000} = F$

# Solving CSPs
## Backtracking

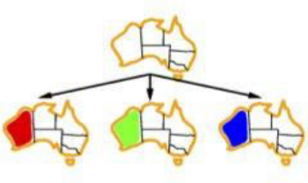
- Variable assignments are commutative: $(WA = red \Rightarrow NT = green) \iff (NT = green \Rightarrow WA = red)$
- Only need to consider assignments to a single variable at each node
- Backtracks when variable has no legal value
- Can solve n-queens for $n \approx 25$

# Solving CSPs
## Backtracking

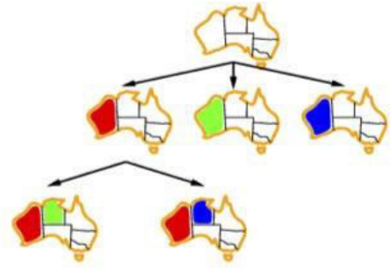
```
function BacktrackingSearch(csp)
    return Backtrack({},csp) //returns solution or failure

function Backtrack(assignment,csp)
    if assignment is complete return assignment
    u_var=SelectUnassignedVariable(csp)
    for each value in OrderDomainValues(u_var,assignment,csp) do
        if isConsistent(value,assignment)
            add {u_var=value} to assignment
            inferences = Inference(csp,u_var,value)
            if inferences!=failure
                add inferences to assignment
                return result
                 if result = failure then
                result = Backtrack(assignment,csp)
        remove {u_var=value} and inferences from assignment
    return failure
```

# Backtracking
## Example (from  Marc Erich Latoshik)

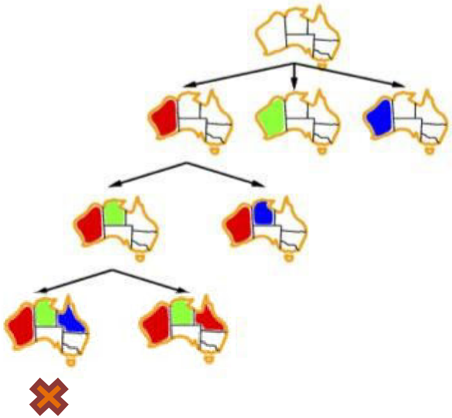
# Backtracking
## Example



# Backtracking
## Example



# Backtracking
## Example



- N-Queens. Place N Queens on an N X N chess board so that no Queen can attack any other Queen.
  - N Variables, one per row.

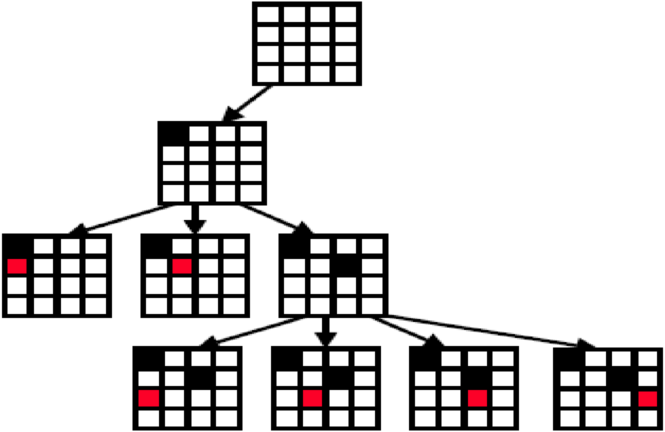
Value of Qi is the column the Queen in row i is placed.

  - Constrains:

$V_i \neq V_j$ for all $i \neq j$ (cannot put two Queens in same column)
$|V_i-V_j| \neq |i-j|$ (Diagonal constraint)
(i.e., the difference in the values assigned to Vi and Vj can't be equal to the difference between i and j.

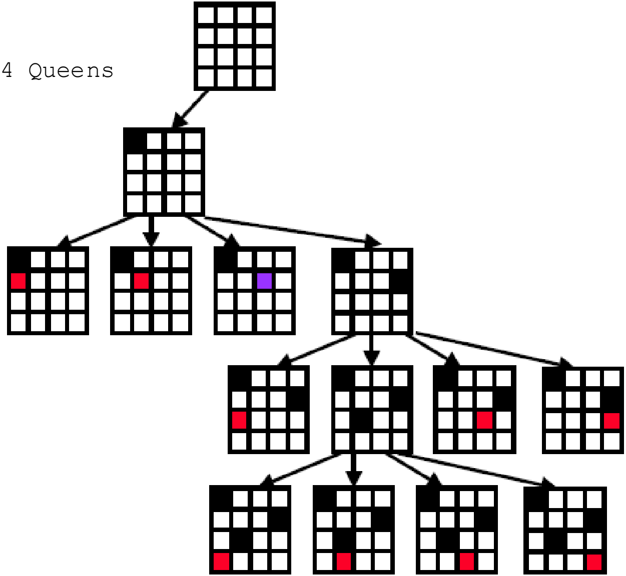
# Backtracking
## Example 2

- 4X4 Queens



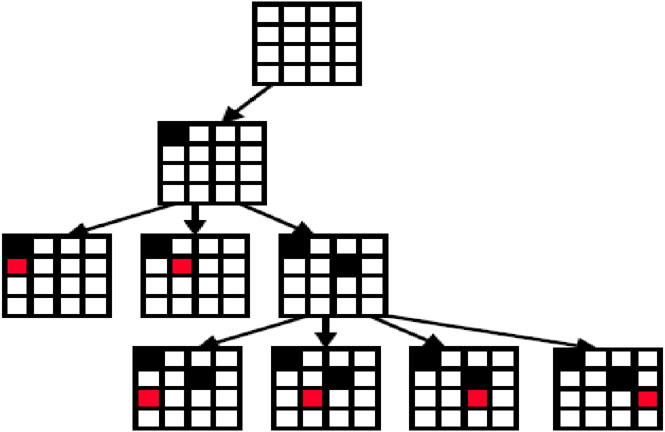
# Backtracking
## Example 2

- 4X4 Queens



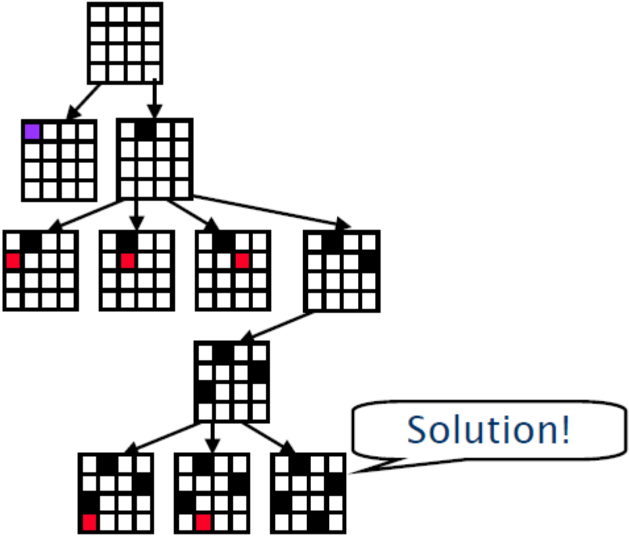
# Backtracking
## Example 2

- 4X4 Queens

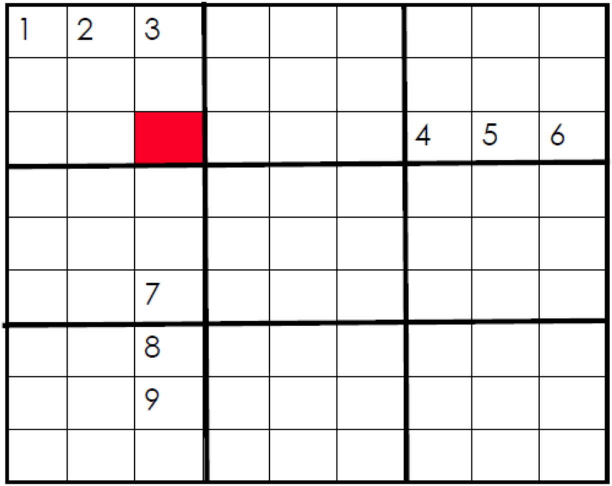


# Backtracking
## Example 2

- 4X4 Queen

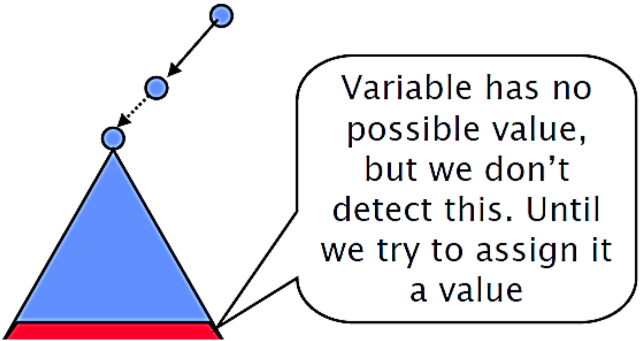


# Problem with plain Backtracking



# Problem with plain Backtracking

- Sudoku:
  - The 3,3 cell has no possible value.  But in the backtracking search we don't detect this until all variables of a row/column or sub-square constraint are assigned. So  we have the following situation



# Inference in CSPs
## Forward Checking

- Constraint *propagation* refers to the technique of "looking ahead" in the search at the as yet unassigned variables.
- Try to detect if any obvious failures have occurred.
- "Obvious" means things we can test/detect efficiently.
- Even if we don't detect an obvious failure we might be able to eliminate some possible part of the future search.

## Inference in CSPs
### Forward Checking

- Forward checking is an extension of backtracking search that employs a "modest" amount of propagation (lookahead).

- When a variable is instantiated we check all constraints that have only one uninstantiated variable remaining.

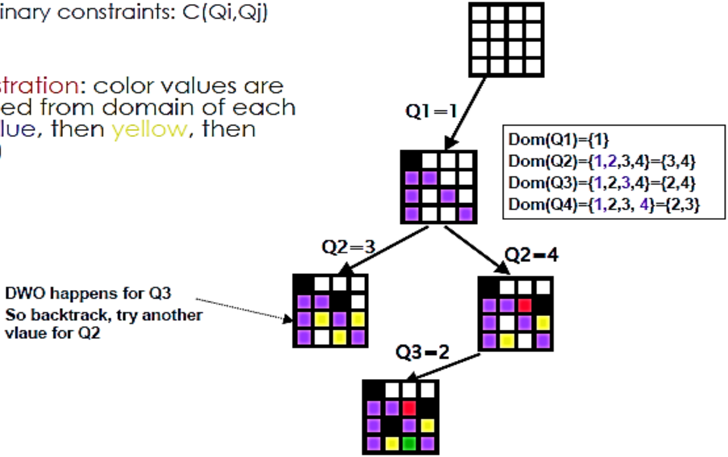
- For that uninstantiated variable, we check all of its values, pruning those values that violate the constraint.

# Inference in CSPs
## Forward Checking

```
function ForwardChecking(csp) //returns a new domain for
     each var
    for each variable X in csp do
        for each unassigned variable Y connected to X do
            for each value d in Domain(Y)
                if d is inconsistent with Value(X)
                    Domain(Y)={Domain(Y)-d}
    return csp //with modified domains
```

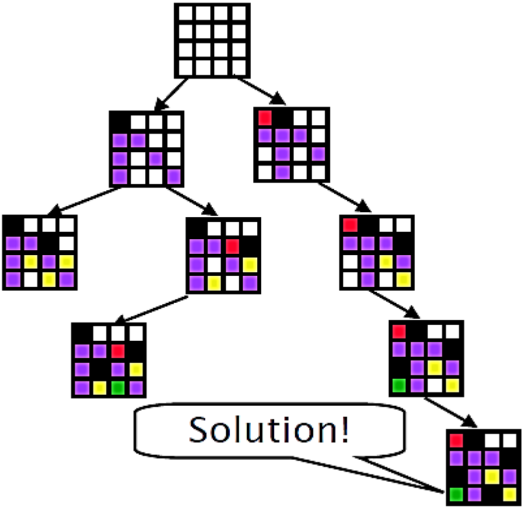
# Inference in CSP
## Example

- 4X4 Queens
  - Q1,Q2,Q3,Q4 with domain {1..4}
  - All binary constraints: C(Qi,Qj)
- FC illustration: color values are removed from domain of each row (blue, then yellow, then green)



# Inference in CSP
## Example

- 4X4 Queens continue…



# Constraint Propagation (Variable and Value Ordering) Example: Color-Mapping

General-Purpose methods can give huge gains in speed:

1: Which variable should be assigned next?

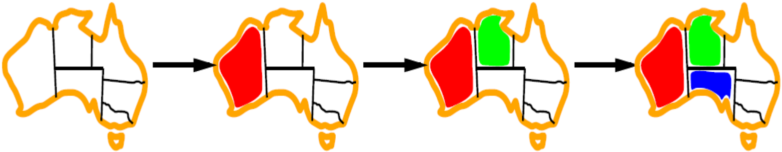
2: In what order should its values be tried?

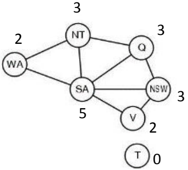
3: Can we detect in inevitable failure early?

# Most-Constrained Variable (Minimum Remaining Value MRV)

Most constrained variable:

choose the variable with the fewest legal values





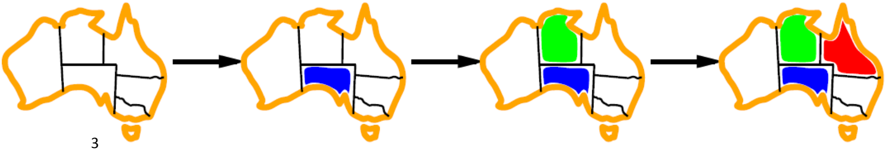

WA=(2)Red
NT(Remaining values)=2(Blue , Green)
while Q and NSW have 3 remaining values so, go for NT
SA( Remaining values)= 1(Blue)

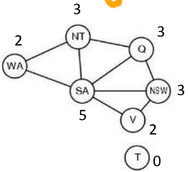
# Most-Constraining Variable (Degree Heuristics)

Tie-breaker among most constrained variables

Most constraining variable:

choose the variable with the most constraints on remaining variables

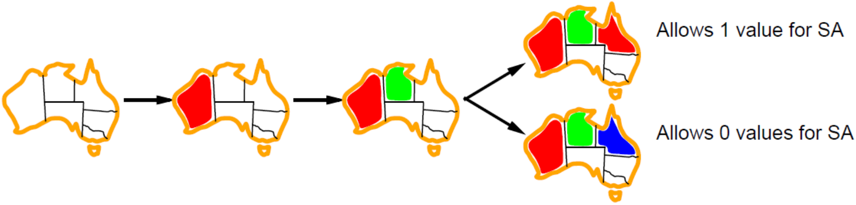





SA=5 (Most Constraints) BLUE
NT, Q, NSW = 3 TIE b/w them
break it by choosing the first
choice either NT or NSW.
So, choose NT
Then Q, NSW OR
NSW, Then Q and NT

# Least-Constraining Variable

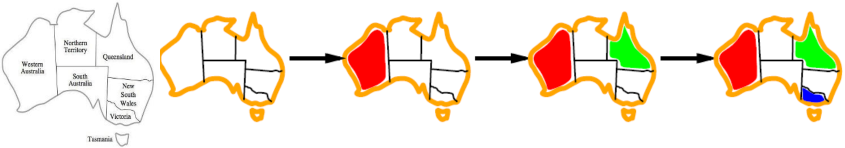
Given a variable, choose the least constraining value:
the one that rules out the fewest values in the remaining variables

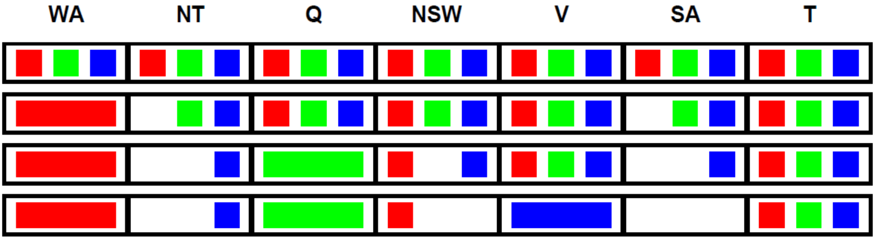



Combining these heuristics makes 1000 queens feasible

# Forward Checking Example-2

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

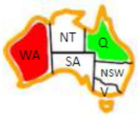


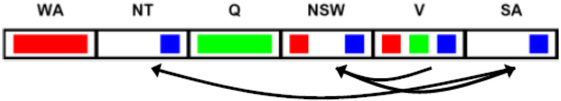




No possible assignments for SA, we try other assignments

# Arc Consistency

- A simple form of propagation makes sure all arcs are simultaneously consistent:

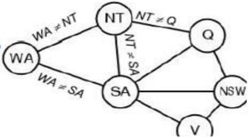


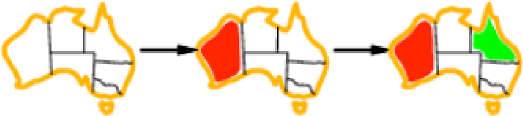

- Arc consistency detects failure earlier than forward checking
- Important: If X loses a value, neighbors of X need to be rechecked!
- Must rerun after each assignment!

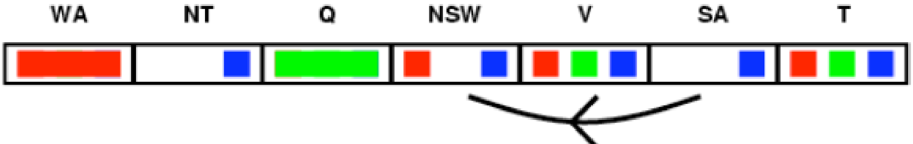
*Remember: Delete from the tail!*

# Arc Consistency



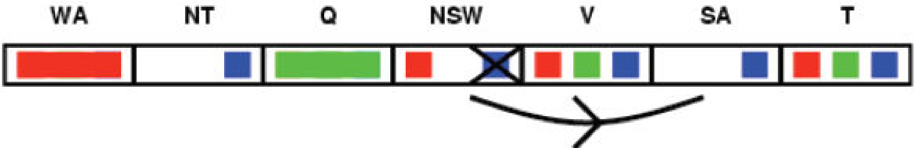

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff for every value $x$ of $X$ there is some allowed $y$



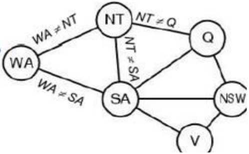


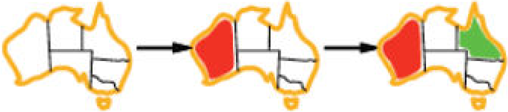
# Arc Consistency

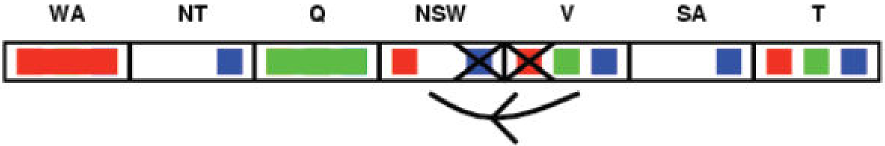
- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff for every value $x$ of $X$ there is some allowed $y$



# Arc Consistency

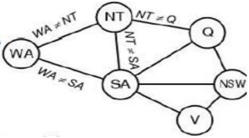


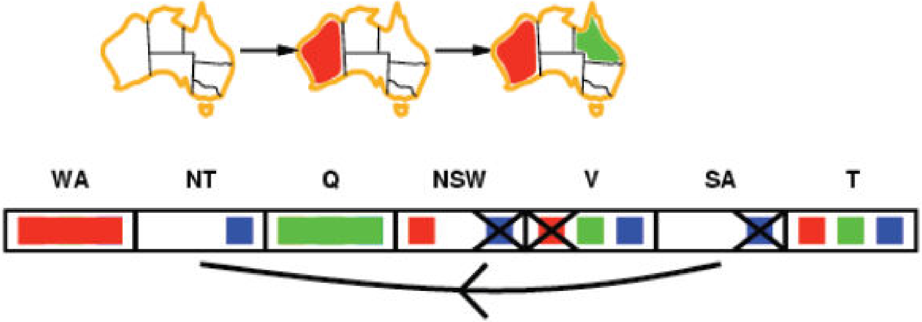



If $X$ loses a value, neighbors of $X$ need to be rechecked

# Arc Consistency




- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

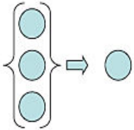


# Inference in CSP'S

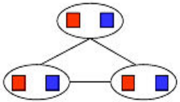
- Increasing degrees of consistency
  - 1-Consistency (Node Consistency): Each single node's domain has a value which meets that node's unary constraints

  - 2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other

  - 3-Consistency (Path Consistency)
  - K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the $k^{th}$ node.



- Higher k more expensive to compute
- (You need to know the k=2 case: arc consistency)



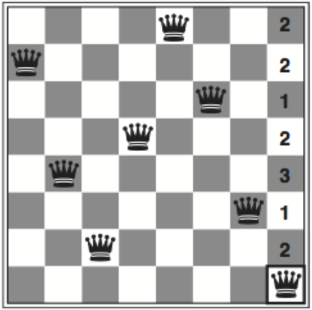
# Intelligent backtracking: Looking backward


- Consider a fixed variable ordering Q, NSW, V , T, SA, WA, NT. Suppose we have generated the partial assignment {Q=red, NSW =green, V =blue, T =red}.
  When we try the next variable,
- SA, we see that every value violates a constraint(chronological backtracking)
- Recoloring Tasmania cannot possibly resolve the problem with South Australia.
- A more intelligent approach to backtracking is to backtrack to a variable that might fix the problem
- The set (in this case {Q=red ,NSW =green, V =blue, }), is called the **conflict set** for SA.
- In this case, back jumping would jump over Tasmania and try a new value for V .

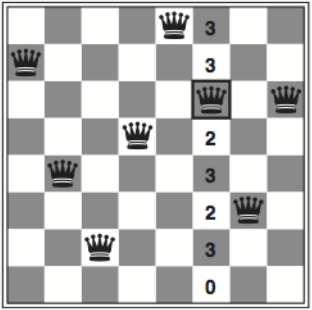
# Local Search in CSPs
## Min-Conflicts

```
function MinConflicts(csp,max_steps)
// csp, max_steps is num of steps before giving up
    current = an initial assignment for csp
    for i=1 to max_steps do
        if current is a solution for csp
            return current
        var = a randomly chosen conflicted variable in
    csp
        value = the value v for var that minimizes
    Conflicts
        set var = value in current
    return failure
```

# Local Search in CSPs
## Example



# Local Search in CSPs
## Example



# Local Search in CSPs
## Example