# Constraint Satisfaction Problems

# Modeling, Inference and Learning

- Modeling: In the context of state-based models, we seek to find minimum cost paths (for search problems) or maximum value policies (for MDPs and games)
- Inference: To compute these solutions, we can either work on the search/game tree or on the state graph
  - The question is where this model actually comes from? Answer: Learning from data (Machine Learning/Data Science)
- Hence modeling, inference, and learning are the three key terms in present day AI

# Modeling, Inference and Learning

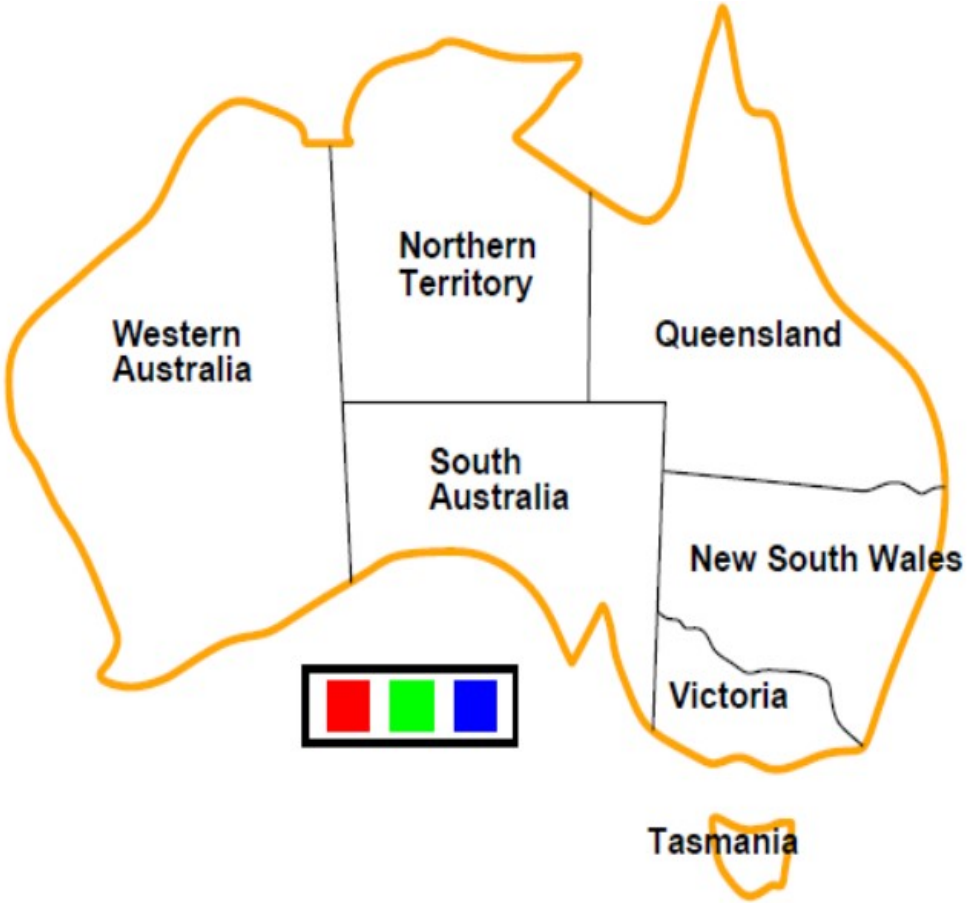
- Modeling: specifies local interactions
- Inference: find globally optimal solutions
- Example: Building a search problem
  - specify how the states are connected through actions and what the local action costs are
  - Need not to specify the long-term consequences of taking an action
  - Inference take all of this local information into account and produce globally optimal solutions (minimum cost paths)
- Why follow Modeling-(followed by) Inference based approach:
  - worry only about local interactions makes modeling easier, but we still get the benefits of a globally optimal solution via inference which are constructed independent of the domain-specific details

# State based Models

- A state is a summary of all the past actions sufficient to choose future actions optimally
- With states, till now we were taking an ordered sequence of actions to reach a goal
- However, in some tasks, order is irrelevant. In these cases, maybe search isn't the best way to model the task
  - Example?

# Example: Map Coloring

- how can we color each of the 7 provinces {red, green,blue} so that no neighboring provinces have the same color?

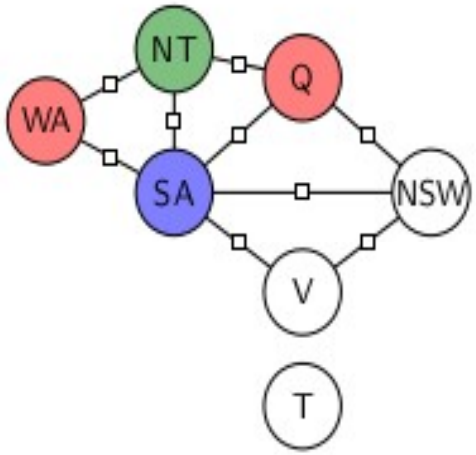


# Example: Map Coloring


One Possible Solution

# Example: Map Coloring as a Search Problem

- State: partial assignment of colors to provinces
- Action: assign next uncolored province a compatible color
- What's missing?
  - ➢ There's more problem structure!
    - ❑ Variable ordering doesn't affect correctness
    - ❑ Variables are interdependent in a local way



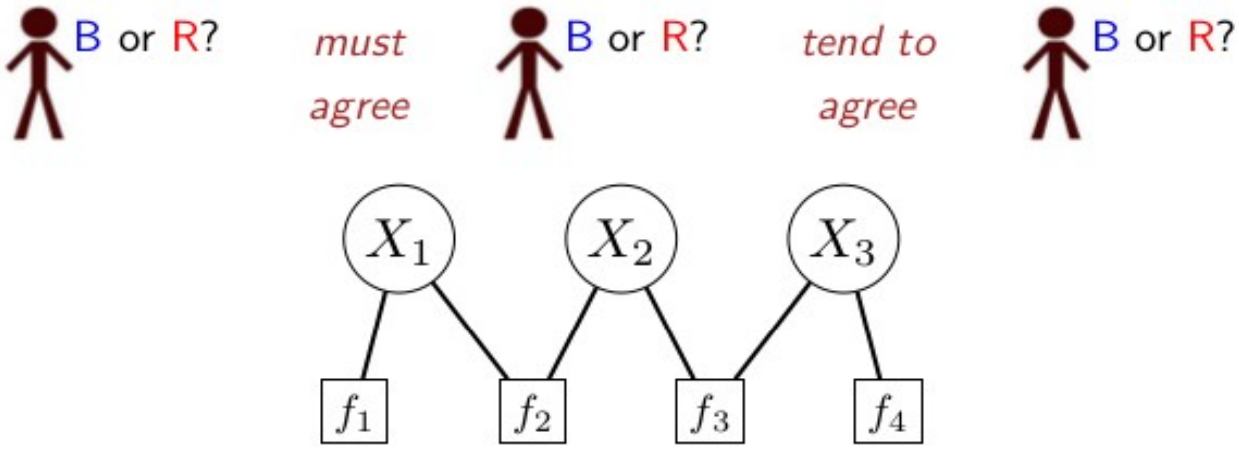
# Variables based Models

- Also known as graphical models
- Basic idea:
  - Modeling: solutions to problems are assignments of values to variables
  - Inference: All the details about how to find the assignment (in particular, which variables to try first)
  - Advantage : Compared to state-based models, it's making the algorithms do more work
- Examples:
  - Bayesian Networks (Directed Graphical Models)
  - Markov Networks (Undirected Graphical Models)
  - Constraint Satisfaction Problems (CSPs)

# Factor Graph

- Example:
  - Suppose there are three people, each of which will vote for a color, red or blue
  - Preferences/Conditions/Constraints:
    - We know that Person 1 is leaning pretty set on blue, and Person 3 is on red
    - Person 1 and Person 2 must have the same color
    - Person 2 and Person 3 would weakly prefer to have the same color
  - One can model this as a factor graph consisting of three variables, X1, X2, X3, each of which must be assigned red (R) or blue (B)
  - One can encode each of the constraints/preferences as a factor, which assigns a non-negative number based on the assignment to a subset of the variables
  - We can either describe the factor as an explicit table, or via a function (e.g., [x1 = x2])

# Factor Graph




| $x_1$ | $f_1(x_1)$ |
|---|---|
| R | 0 |
| B | 1 |

| $x_1$ | $x_2$ | $f_2(x_1, x_2)$ |
|---|---|---|
| R | R | 1 |
| R | B | 0 |
| B | R | 0 |
| B | B | 1 |

| $x_2$ | $x_3$ | $f_3(x_2, x_3)$ |
|---|---|---|
| R | R | 3 |
| R | B | 2 |
| B | R | 2 |
| B | B | 3 |

| $x_3$ | $f_4(x_3)$ |
|---|---|
| R | 2 |
| B | 1 |

$$f_2(x_1, x_2) = [x_1 = x_2] \qquad f_3(x_2, x_3) = [x_2 = x_3] + 2$$

# Factor Graph

| $x_1$ | $f_1(x_1)$ |
|---|---|
| R | 0 |
| B | 1 |

| $x_1$ | $x_2$ | $f_2(x_1, x_2)$ |
|---|---|---|
| R | R | 1 |
| R | B | 0 |
| B | R | 0 |
| B | B | 1 |

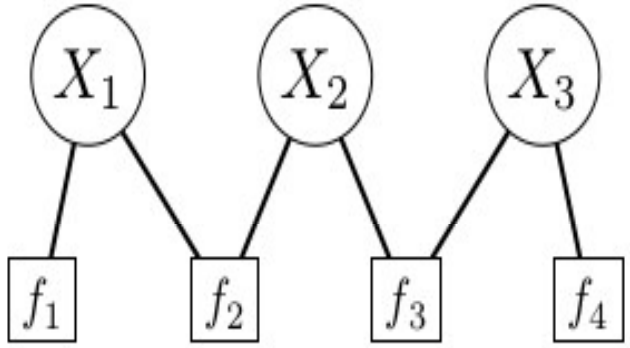
| $x_2$ | $x_3$ | $f_3(x_2, x_3)$ |
|---|---|---|
| R | R | 3 |
| R | B | 2 |
| B | R | 2 |
| B | B | 3 |

| $x_3$ | $f_4(x_3)$ |
|---|---|
| R | 2 |
| B | 1 |

| $x_1$ | $x_2$ | $x_3$ | Weight |
|---|---|---|---|
| R | R | R | $0 \cdot 1 \cdot 3 \cdot 2 = 0$ |
| R | R | B | $0 \cdot 1 \cdot 2 \cdot 1 = 0$ |
| R | B | R | $0 \cdot 0 \cdot 2 \cdot 2 = 0$ |
| R | B | B | $0 \cdot 0 \cdot 3 \cdot 1 = 0$ |
| B | R | R | $1 \cdot 0 \cdot 3 \cdot 2 = 0$ |
| B | R | B | $1 \cdot 0 \cdot 2 \cdot 1 = 0$ |
| B | B | R | $1 \cdot 1 \cdot 2 \cdot 2 = 4$ |
| B | B | B | $1 \cdot 1 \cdot 3 \cdot 1 = 3$ |

# Factor Graph Definition

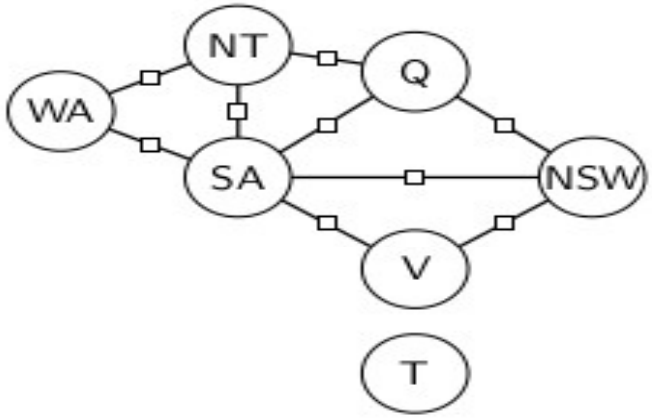
- A factor graph consists of a set of variables and a set of factors:
  - n variables $X_1, \ldots, X_n$, which are represented as circular nodes in the graphical notation
  - m factors/potentials $f_1, \ldots, f_m$, which are represented as square nodes in the graphical notation




  - Each variable $X_i$ can take on values in its domain Domain *i*
  - Each factor $f_j$ is a function that takes an assignment x to all the variables and returns a non-negative number representing how good that assignment is (from the factor's point of view)
  - Usually, each factor will depend only on a small subset of the variables

# Factor Graph: Map Coloring

- Variables: X = (WA, NT, SA, Q, NSW, V, T)
- $Domain_i \in \{R, G, B\}$
- Factors:
  - $f_1(X) = [WA \neq NT]$
  - $f_2(X) = [NT \neq Q]$




- Scope of a factor $f_j$ : set of variables it depends on
- Arity of $f_j$ is the number of variables in the scope
  Unary factors(arity 1); Binary factors(arity 2)

# Factor Graph: Map Coloring

- A factor graph specifies all the local interactions between variables
- We wish to find a global solution i.e. an assignment, which specifies a value for each variable
- Each assignment is associated with a weight, which is just the product over each factor evaluated on that assignment
- Intuitively, each factor contributes to the weight and any factor has veto power: if it returns zero, then the entire weight is irrecoverably zero
- In this setting, the maximum weight assignment is (B, B, R), which has a weight of 4
- One can think of this as the optimal configuration or the most likely outcome

# Factor Graph

| $x_1$ | $f_1(x_1)$ |
|---|---|
| R | 0 |
| B | 1 |

| $x_1$ | $x_2$ | $f_2(x_1, x_2)$ |
|---|---|---|
| R | R | 1 |
| R | B | 0 |
| B | R | 0 |
| B | B | 1 |

| $x_2$ | $x_3$ | $f_3(x_2, x_3)$ |
|---|---|---|
| R | R | 3 |
| R | B | 2 |
| B | R | 2 |
| B | B | 3 |

| $x_3$ | $f_4(x_3)$ |
|---|---|
| R | 2 |
| B | 1 |

| $x_1$ | $x_2$ | $x_3$ | Weight |
|---|---|---|---|
| R | R | R | $0 \cdot 1 \cdot 3 \cdot 2 = 0$ |
| R | R | B | $0 \cdot 1 \cdot 2 \cdot 1 = 0$ |
| R | B | R | $0 \cdot 0 \cdot 2 \cdot 2 = 0$ |
| R | B | B | $0 \cdot 0 \cdot 3 \cdot 1 = 0$ |
| B | R | R | $1 \cdot 0 \cdot 3 \cdot 2 = 0$ |
| B | R | B | $1 \cdot 0 \cdot 2 \cdot 1 = 0$ |
| B | B | R | $1 \cdot 1 \cdot 2 \cdot 2 = 4$ |
| B | B | B | $1 \cdot 1 \cdot 3 \cdot 1 = 3$ |

# Assignment Weights

- Each assignment $x = (x_1, \ldots, x_n)$ has a weight:

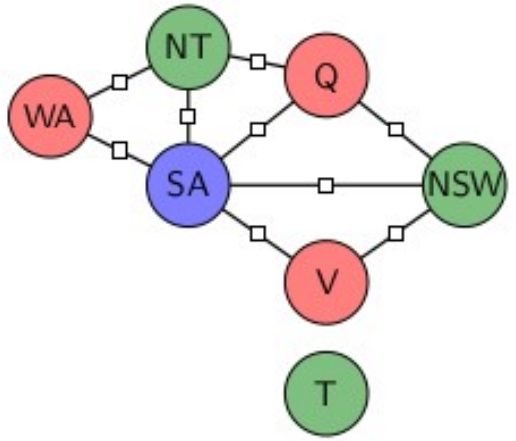
$$Weight(x) = \prod_{j=1}^{m} f_j(x)$$

- Obejctive: Find the maximum weight assignments:

$$\operatorname*{argmax}_x Weight(x)$$

# Constrained Satisfaction Problem (CSP): Example

- Map-Coloring:
  - Assignment: x = {WA : R, NT : G, SA : B, Q : R, NSW: G , V : R, T : G}
  - Weight: $1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 = 1$
  - Assignment: x = {WA : R, NT : R, SA : B, Q : R, NSW: G , V : R, T : G}
  - Weight: $0 \cdot 0 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 = 1$
- Each factor only looks at the variables of two adjacent provinces and checks if the colors are different (returning 1) or the same (returning 0)
- Modeling : Specifying local interactions in a modular way




- Inference: Multiply all the factors to achieve notion of global consistency
- factors are multiplied, i.e. any factor has veto power: a single zero causes the entire weight to be zero

# Constraint satisfaction problems

- A CSP is a factor graph where all factors are constraints:

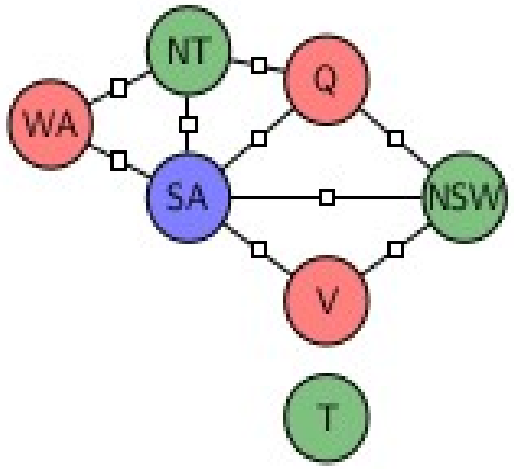
$$f_j(x) \in \{0,1\} \forall j = \{1, \ldots, m\}$$

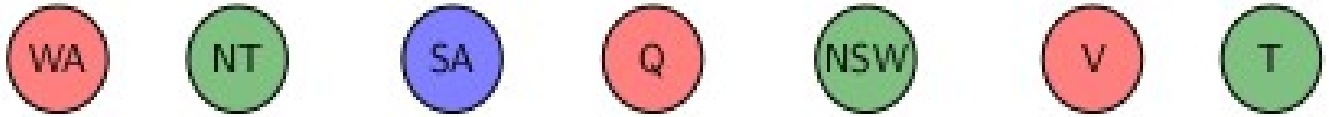
  - The constraint is satisfied iff $f_j(x) = 1$
- Constraint satisfaction problems are just a special case of factor graphs where each of the factors returns either 0 or 1. Such a factor is a constraint, where 1 means the constraint is satisfied and 0 means that it is not
- In a CSP, all assignments have either weight 1 or 0. Assignments with weight 1 are called consistent (they satisfy all the constraints), and the assignments with weight 0 are called inconsistent. Our goal is to find any consistent assignment (if one exists)

# Partial Assignments

- Weight of a full assignment is the product of all the factors applied to that assignment
- One can extend this definition to partial assignments: The weight of a partial assignment is defined to be the product of all the factors whose scope includes only assigned variables
- For example, if only WA and NT are assigned, the weight is just value of the single factor between them
- Weight of the new extended assignment: is defined to be the original weight times all the factors that depend on the new variable and only previously assigned variables

# Extending Partial Assignment : Example




[WA ≠ NT]

[WA ≠ SA]
[NT ≠ SA]

[NT ≠ Q]
[SA ≠ Q]

[SA ≠ NSW]
[Q ≠ NSW]

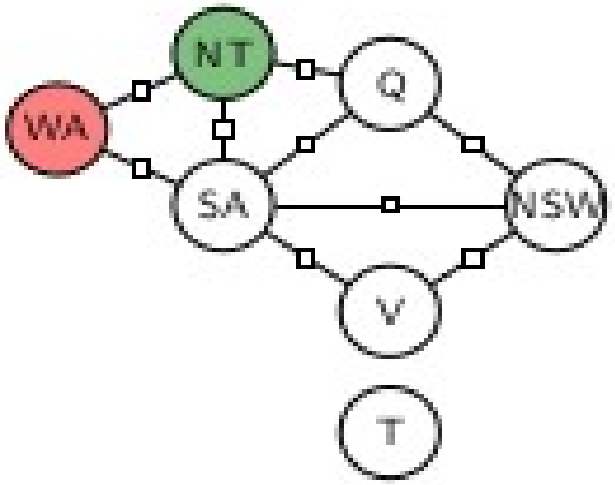
[SA ≠ V]
[NSW ≠ V]

# Dependent Factors

- Dependent factors denoted as $D(x, X_i)$ be set of factors depending on the chosen unassigned variable $X_i$ and partial assignment x
- Partial assignment

  (e.g., x = {WA : R, NT : G})
- D({WA : R, NT : G}, SA) =

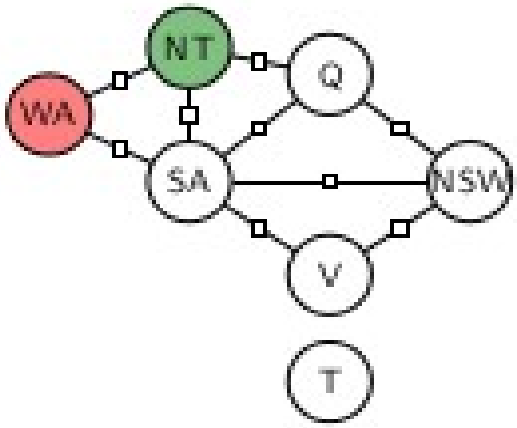
  {[WA ≠ SA], [NT ≠ SA]}



# Backtracking Search Algorithm

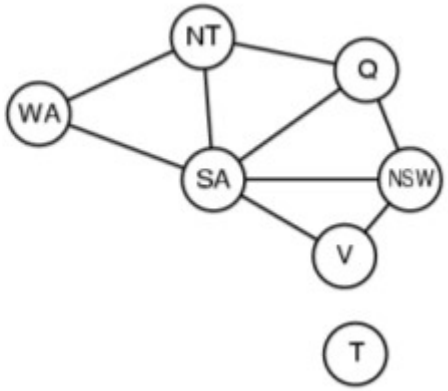
- Backtrack(x, w, Domains):
  - If x is complete assignment: update best and return
  - Choose unassigned VARIABLE $X_i$
  - Order VALUES $Domain_i$ of chosen
  - For each value v in that order:

$$\delta \leftarrow \prod_{f_j \in D(x,X_i)} f_j(x \cup \{X_i : v\})$$

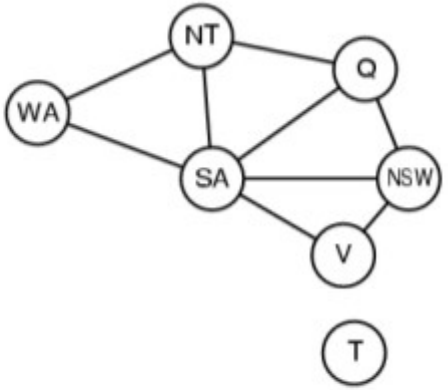



  - ❑If δ = 0: continue
  - ❑Domains' ← Domains via LOOKAHEAD
  - ❑Backtrack(x ∪ {Xi : v}, wδ, Domains')

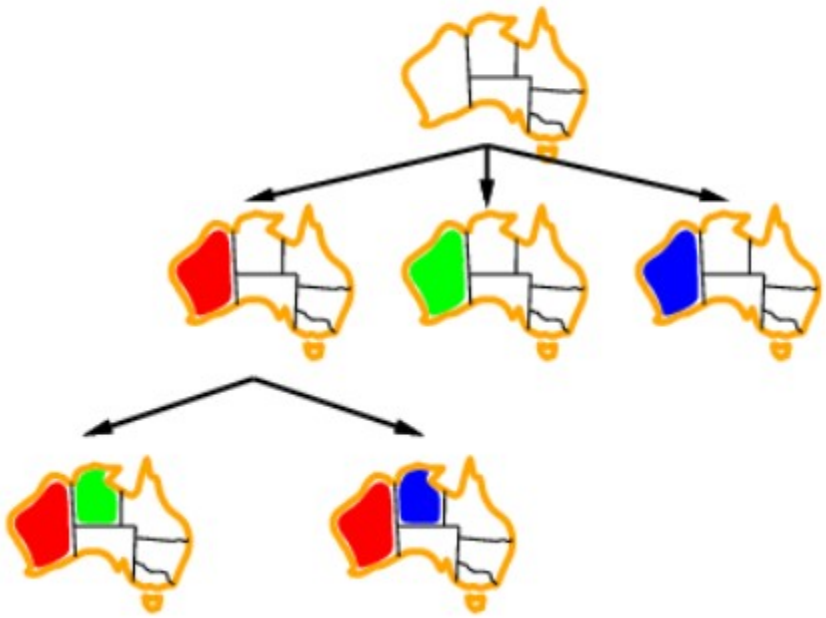
# Backtracking Example

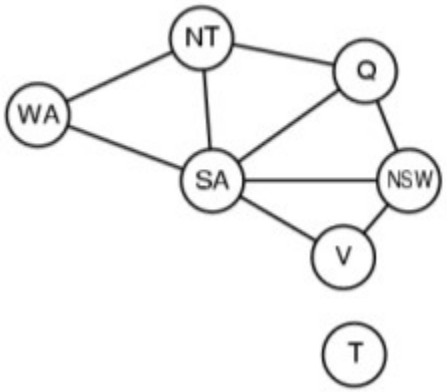


# Backtracking Example

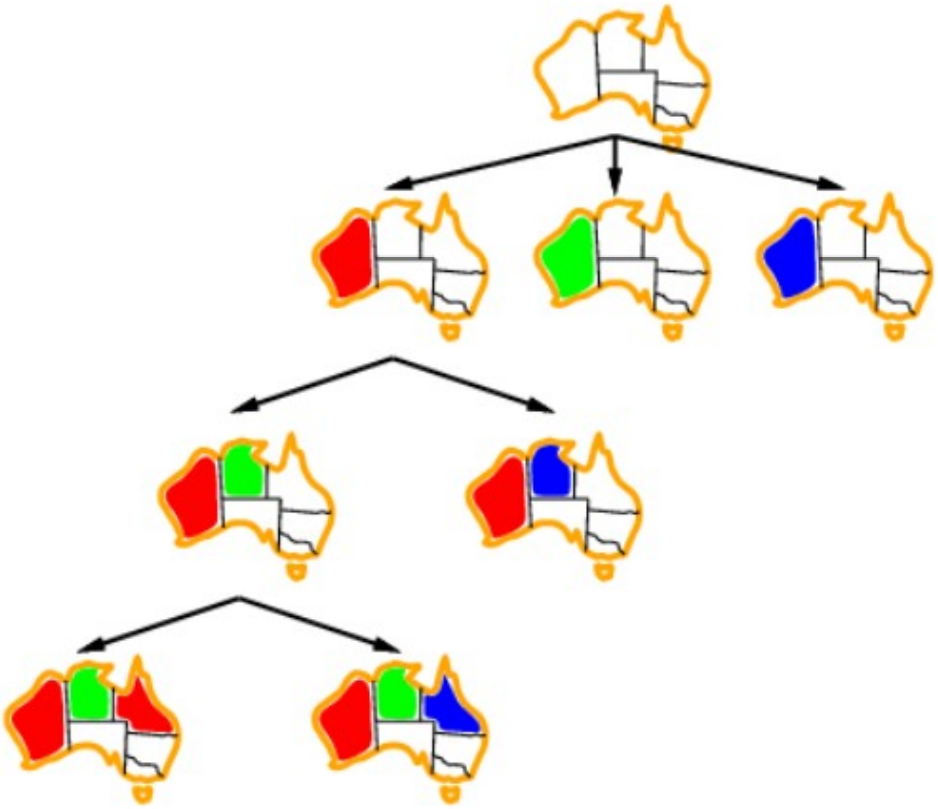


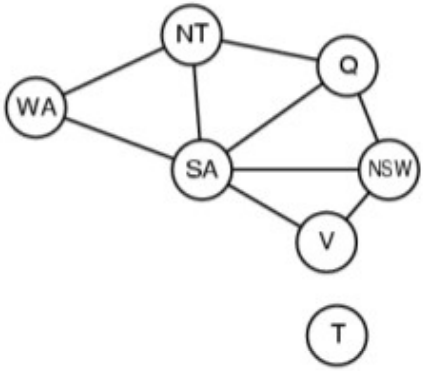
# Backtracking Example





# Backtracking Example





# Improving Backtracking

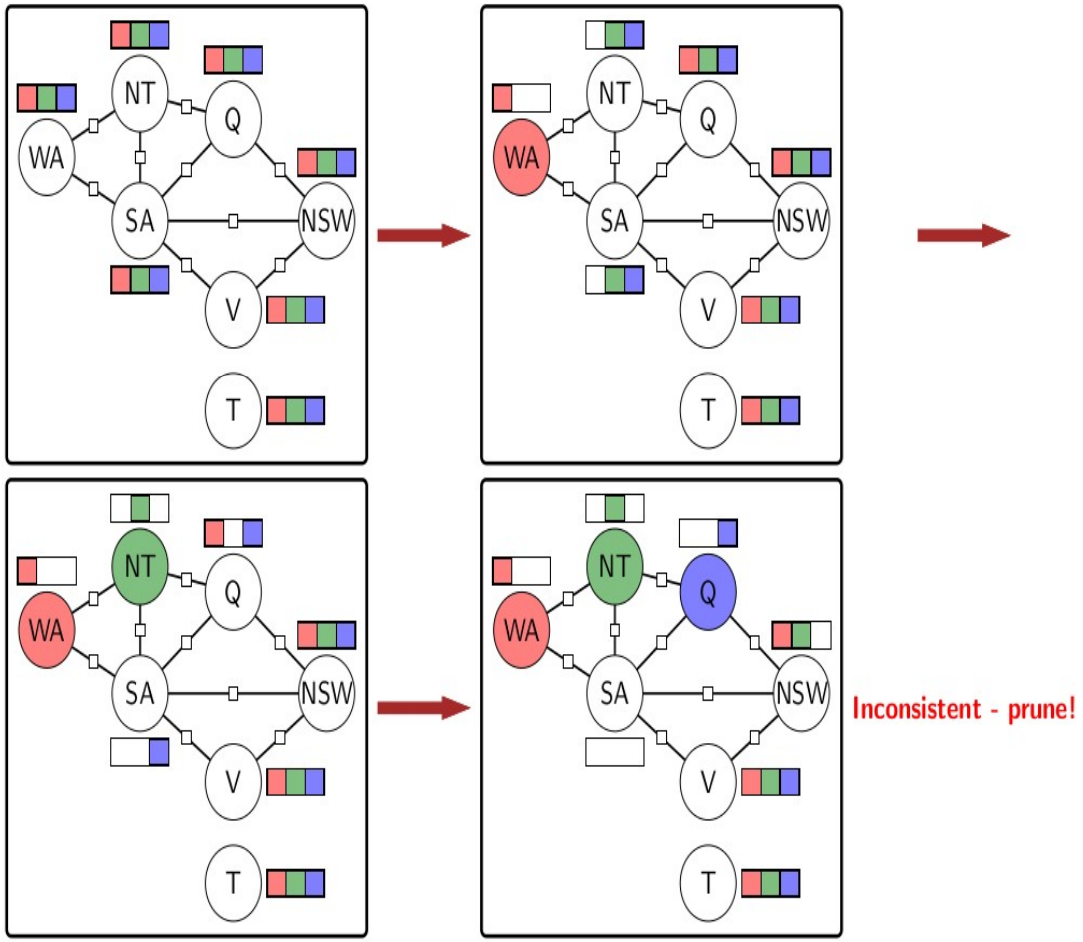
- Ordering:
  - Variable ordering
  - Value ordering
- Exploiting the “problem structure”
- Early detection of inevitable failure

# Lookadhead: Forward Checking

- Way to perform a one-step lookahead
  - As soon as one assigns a variable (e.g., WA = R), one can pre-emptively remove inconsistent values from the domains of neighboring variables (i.e., those that share a factor)
  - If one keeps on doing this and get to a point where some variable has an empty domain, then we can stop and backtrack immediately, since there's no possible way to assign a value to that variable which is consistent with the previous partial assignment
  - When unassigning a variable, remember to restore the domains of its neighboring variables

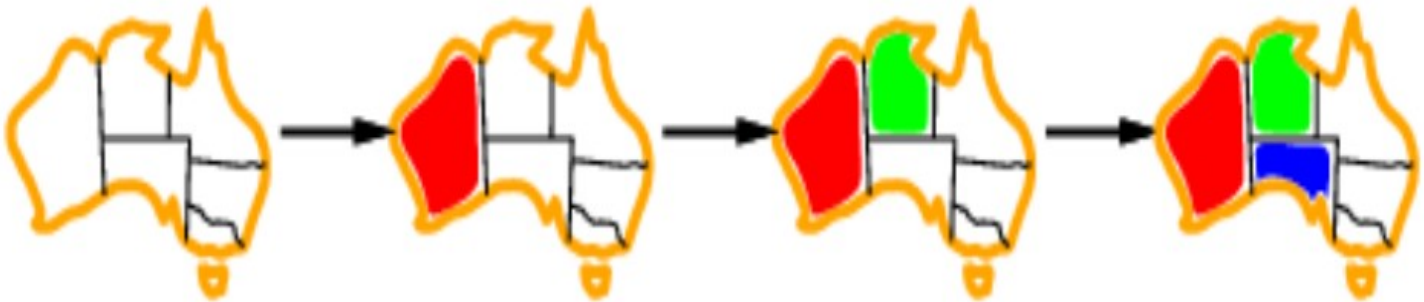
# Forward Checking: Example

- In this example, after Q is assigned blue, one can remove inconsistent values (blue) from SA's domain, emptying it. At this point, one needs not even recurse further,since there's no way to extend current assign-ment. We would then instead try assigning Q to red.



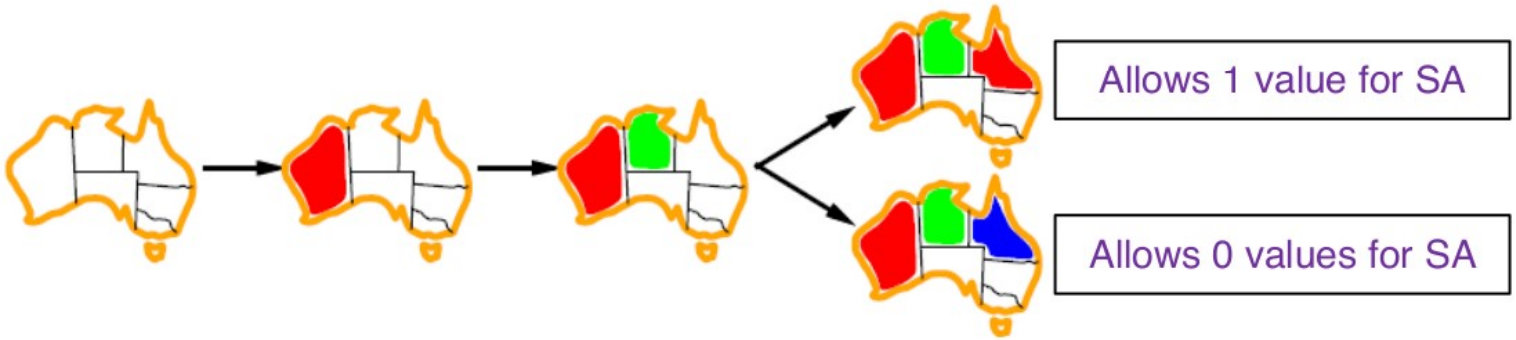
# How to Choose an Unassigned Variable

- (Unassigned) Variable Ordering
  - (Intuitively) Choose the variable which is most constrained, i.e. the variable whose domain has the fewest number of remaining valid values (based on forward checking)
  - Because those variables yield smaller branching factors



# How to Choose an Unassigned Variable

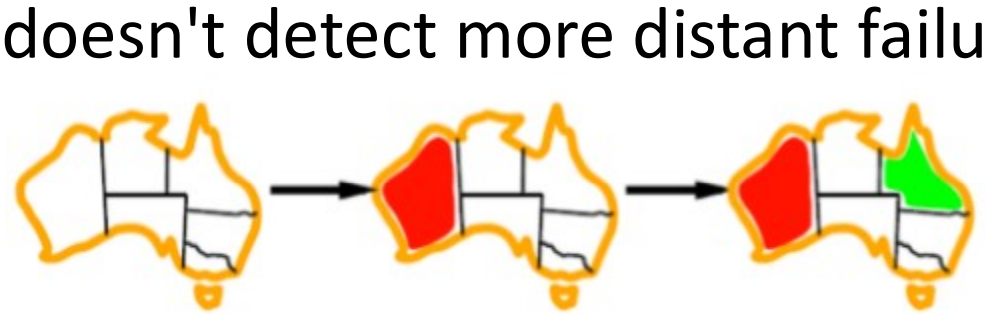
- Value Ordering
  - ➢ First try values which are less constrained
    - ❑One that rules out the fewest choices for the remaining/neighboring variables




  - ➢ Maximum flexibility for subsequent variable assignments
  - ➢ e.g. 1000-Queens problem becomes feasible using the combination of these heuristics

# Constraint Propagation

- Forward checking propagates information from assigned to adjacent unassigned variables, but doesn't detect more distant failures

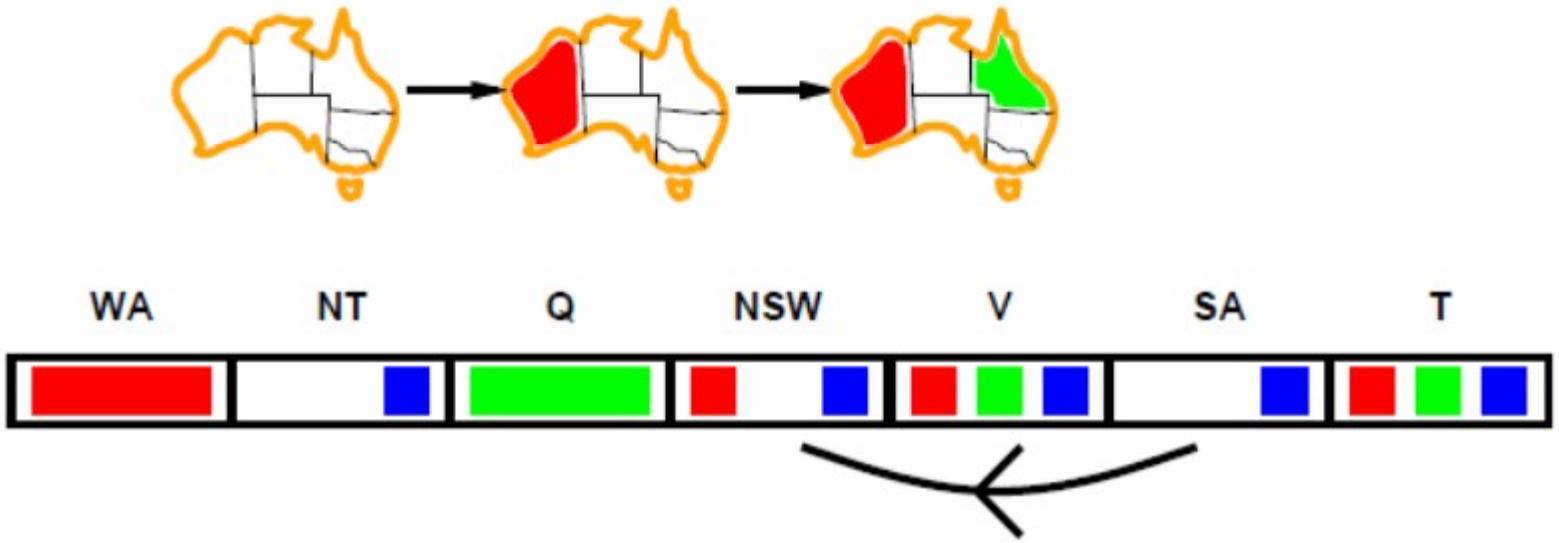



# Constraint Propagation

- NT and SA cannot be both blue
- Forward checking has not detected it yet
- Constraint propagation repeatedly enforces constraints locally

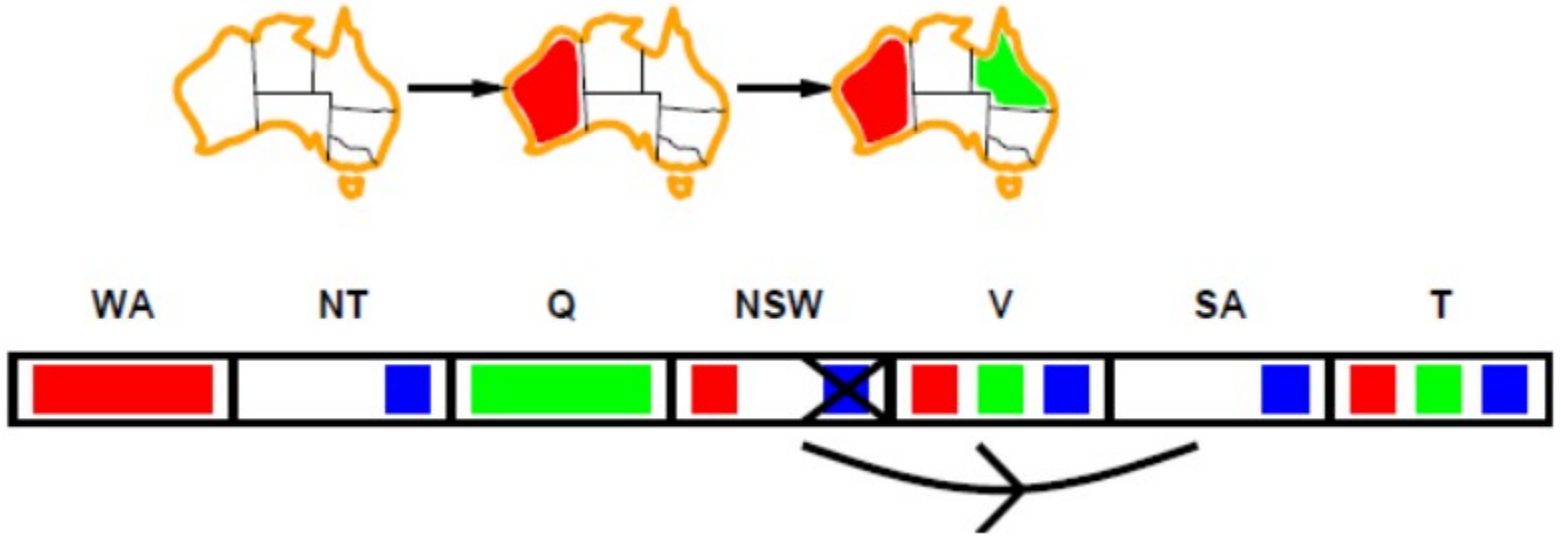
# Arc Consistency

- X $\longrightarrow$ Y is consistent iff for every value x of X there is some allowable value y of Y



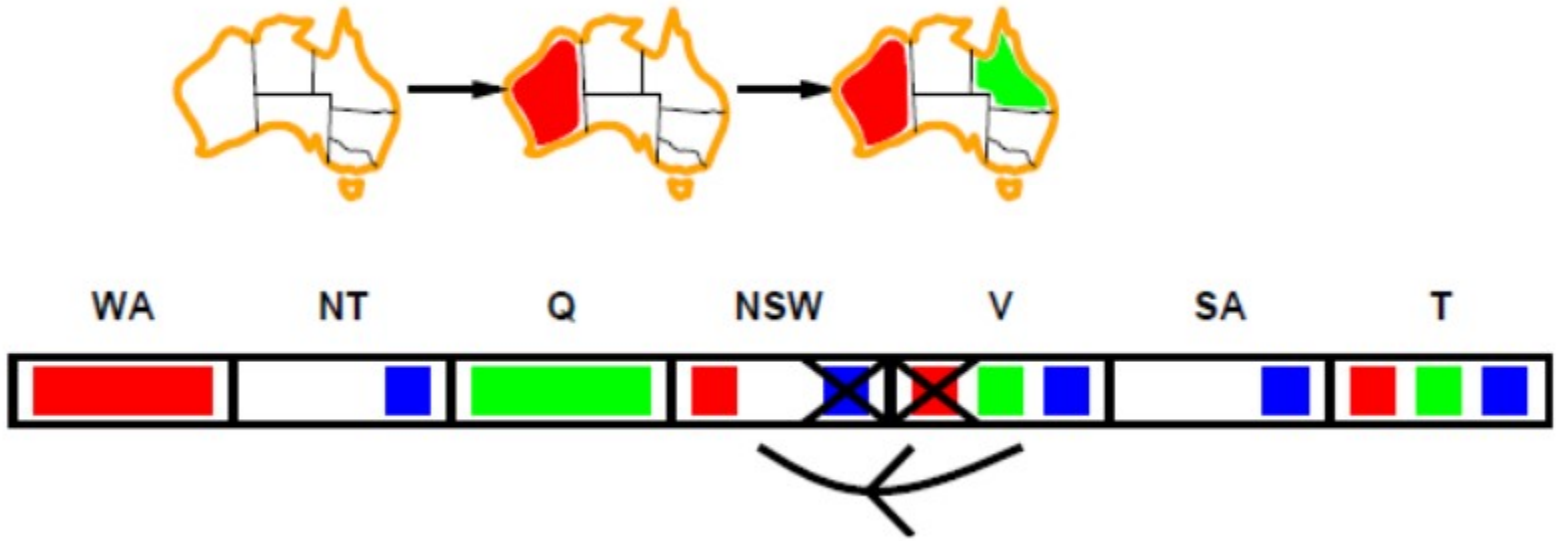
# Arc Consistency

- X $\longrightarrow$ Y is consistent iff for every value x of X there is some allowable value y of Y



# Arc Consistency

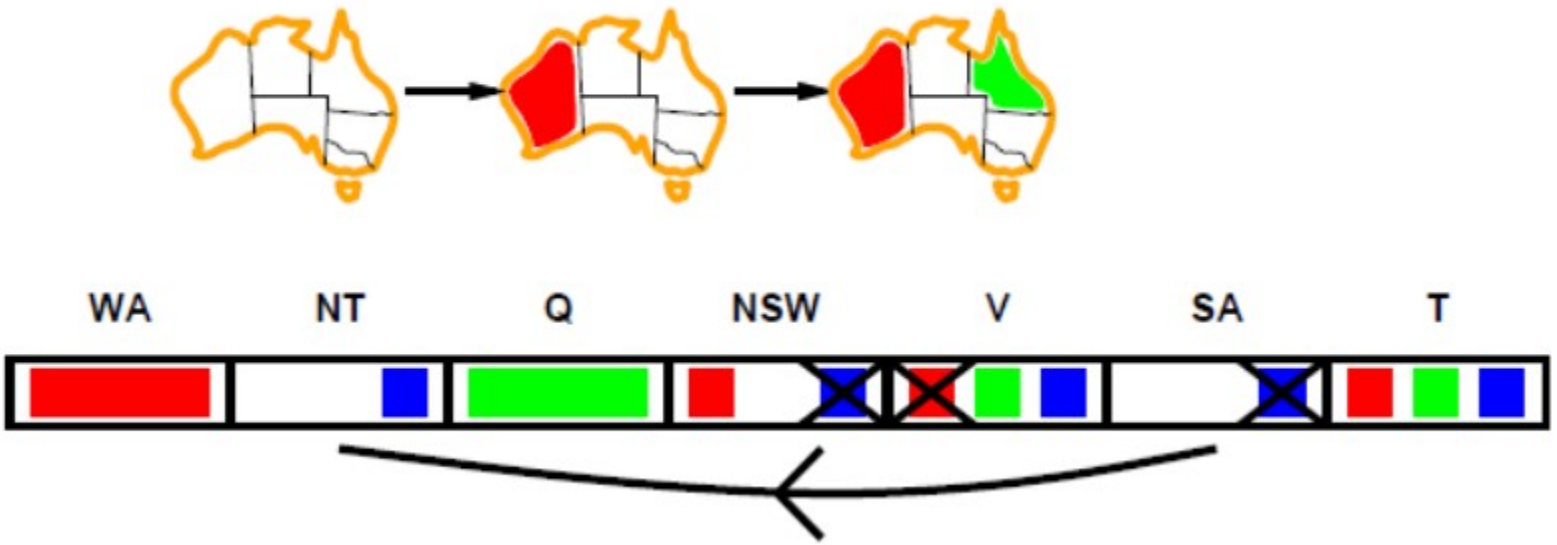
- X $\longrightarrow$ Y is consistent iff for every value x of X there is some allowable value y of Y




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

# Arc Consistency

- X $\longrightarrow$ Y is consistent iff for every value x of X there is some allowable value y of Y



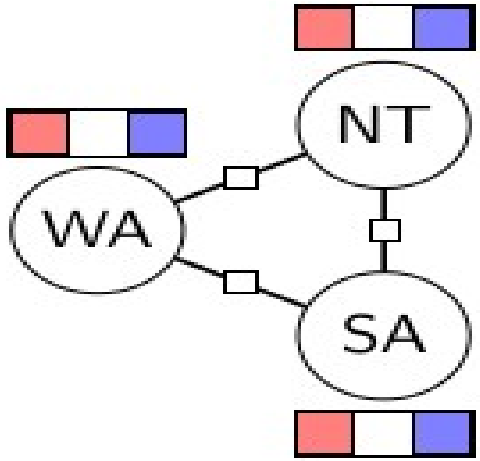

- If X looses a value, neighbors of X need to be rechecked

# Arc Consistency Algorithm

- Add $X_j$ to set
- While set is non-empty:
  - Remove any $X_k$ from set
  - For all neighbors $X_l$ of $X_k$ :
    - Enforce arc consistency on $X_l$ with respect to $X_k$
    - If Domain($X_l$) changed, add $X_l$ to set

# Limitations of Arc Consistency

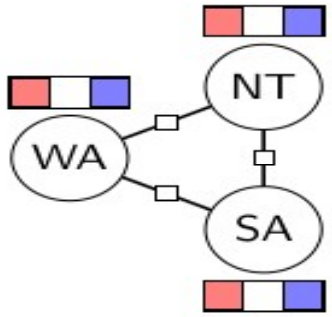
- Consider below:




- No consistent assignment, but algorithm doesn’t detect a problem

# Limitations of Arc Consistency

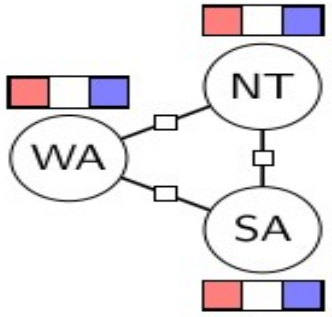
- Generalize arc consistency:
  - Instead of looking at every 2 variables and the factors between them, we could look at every subset of k variables
  - Check that there's a way to consistently assign values to all k, taking into account all the factors involving those k variables




  - e.g. k = 3, A two variable set $\{X_i, X_k\}$ is path-consistent with respect to a third variable $X_j$ if, for every assignment $\{X_i = a, X_k = b\}$ consistent with the constraints on $\{X_i, X_k\}$ , there is an assignment to $X_j$ that satisfies the constraints on $\{X_i, X_j\}$ and $\{X_j, X_k\}$
  - Path consistency because looking at a path from $X_i$ to $X_k$ via $X_j$

# Limitations of Arc Consistency

- Example:
  - Make the set {WA, SA} path consistent with respect to NT
  - Enumerate the consistent assignments: {WA = *red*, SA = *blue*} and {WA = *blue,* SA = *red*}



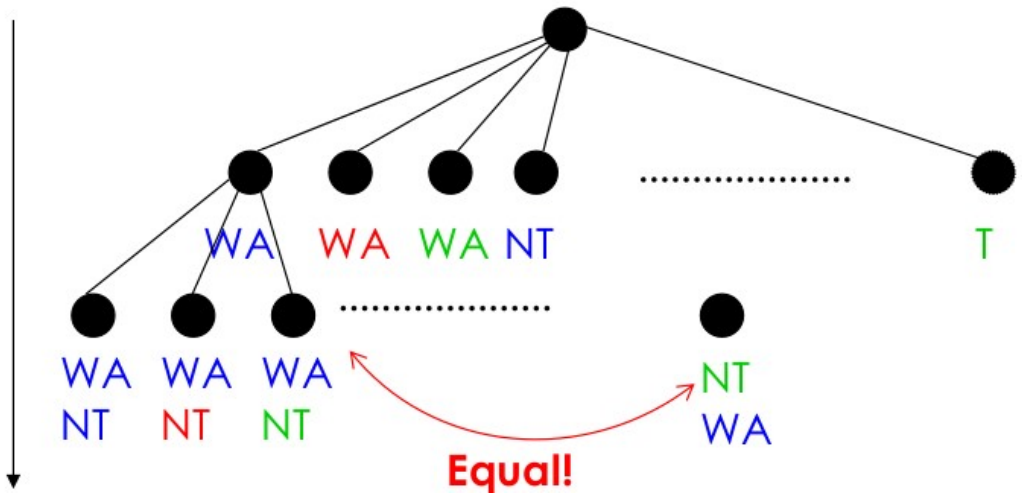

  - With both of the above consistent assignments NT can be neither *red* nor *blue*
  - Eliminate both assignments, leading to no valid assignments for {WA, SA}
- However, there is a substantial cost to doing this (the running time is exponential in k in the worst case), so generally arc consistency (k = 2) or (k = 3) is good enough

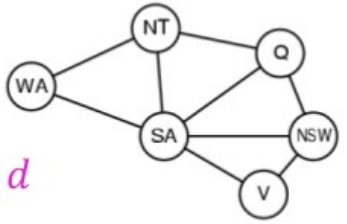
# Complexity of Arc Consistency Algorithm

- A variable could be added to the set multiple times, why?
  - ➢ As its domain can get updated more than once
- One might enforce arc consistency on $(X_i,X_j)$ up to D times in the worst case, where $D = \max_{1 \le i \le n} |Domain_i|$ is the size of the largest domain
- There will be at most of the order of $E$ different pairs $(X_i,X_j)$, where E is the number of edges
- Corresponding call to each of the above $E$ pairs takes $O(D^2)$ time to enforce arc consistency
- Therefore the running time of this algorithm is $O(ED^3)$ in the worst case

# Complexity of Backtracking Search Algorithm

- Initial State – Empty assignment
- Child generating function – assign value to an unassigned variable
  - ➢ Fail! If no legal assignments
- Goal Test – a complete consistent assignment




$n * d$

$(n * d) * (n - 1) * d$

$n! * d^n$

# Complexity of Backtracking Search Algorithm

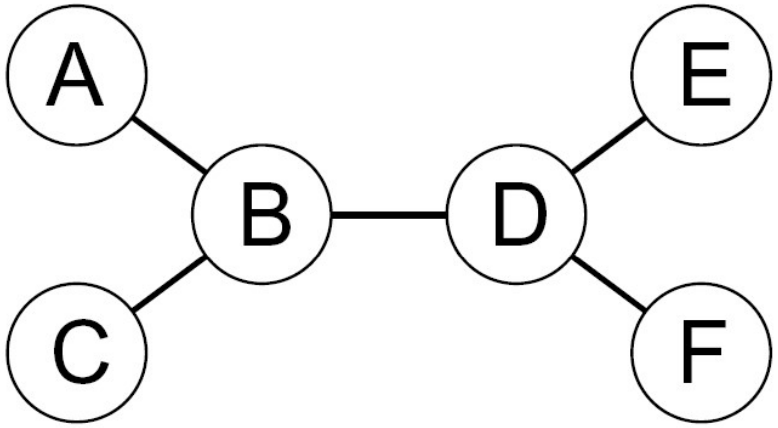
- Initial State – Empty assignment
- Child generating function – assign value to an unassigned variable
  - Fail! If no legal assignments
- Goal Test – a complete consistent assignment
- Every solution appears at depth n with n variables
  - Depth first search
- Variable assignments are commutative
  - [WA = *green* then NT = *red*] same as [NT = *red then* WA = *green*]
- Only need to consider a single variable at each node
  - b = D $\Longrightarrow$ $D^n$ leaves

# Complexity of Backtracking Search Algorithm

- Backtracking search Algorithm requires exponential time
  - One can cleverly employ lookahead and dynamic variable/value ordering
    - In some cases it can dramatically improve running time
  - But the worst case is still exponential
  - These strategies are helpful only when there are hard constraints, which allow us to prune away values which definitely are not compatible with the current assignment
  - What if the factors were strictly positive?
    - None of the pruning techniques discussed (in the previous classes) would be useful

# Tree-Structured CSPs

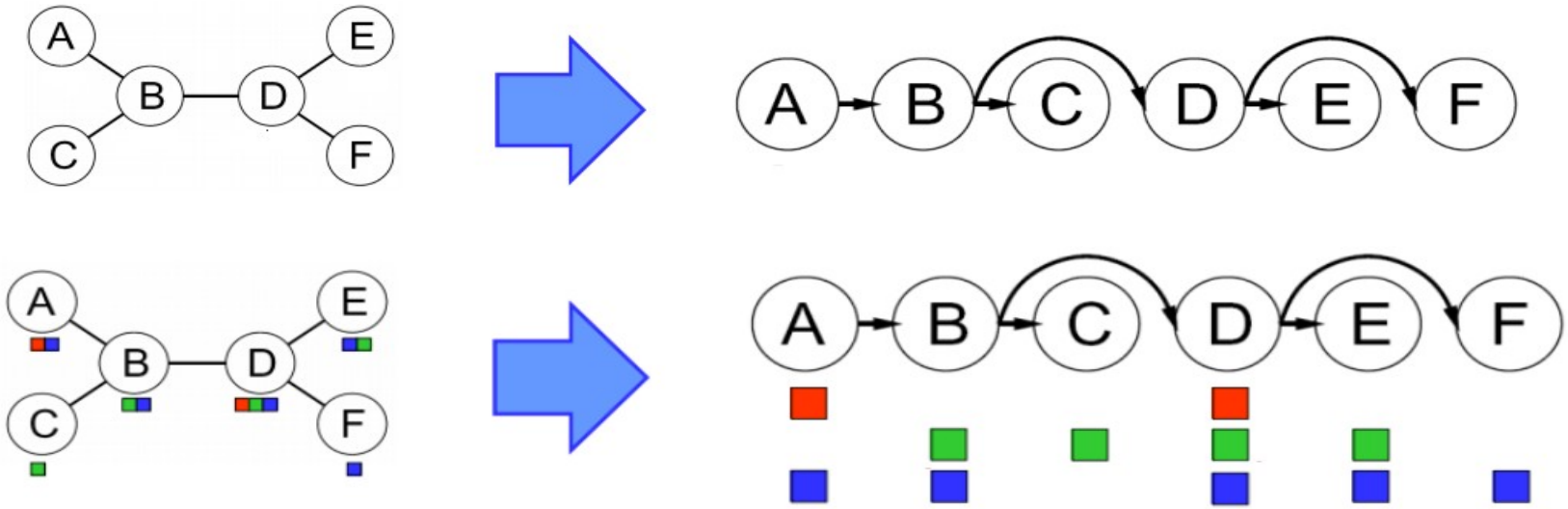
- What if constraint graph of CSP does not have any loops?
  - Problem can be solved without using back-tracking search




- Choose one variable as a root
- Order variables from root to leaves such that every node's parent precedes it in the ordering
- Every node will have only one parent

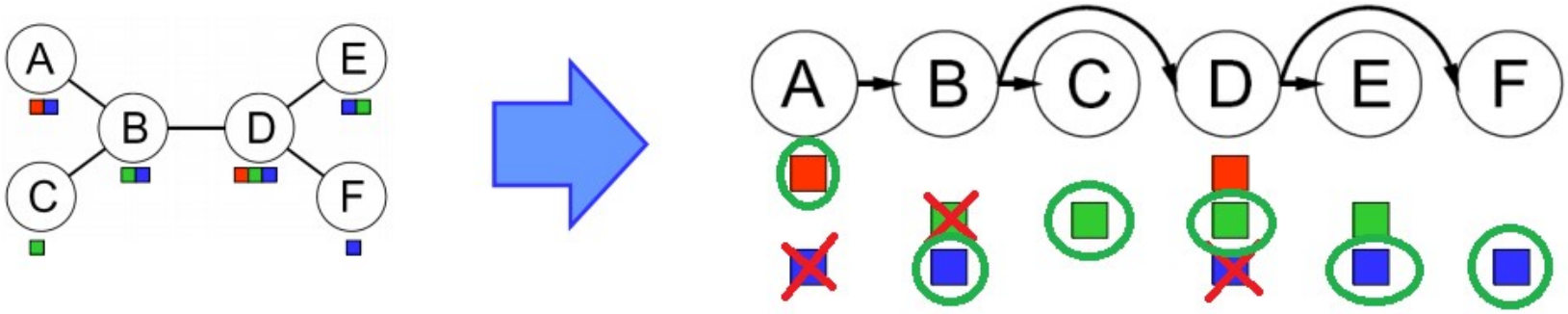
# Tree-Structured CSPs

- **Backward removal phase**: check arc consistency starting from the rightmost node and going backwards



# Tree-Structured CSPs

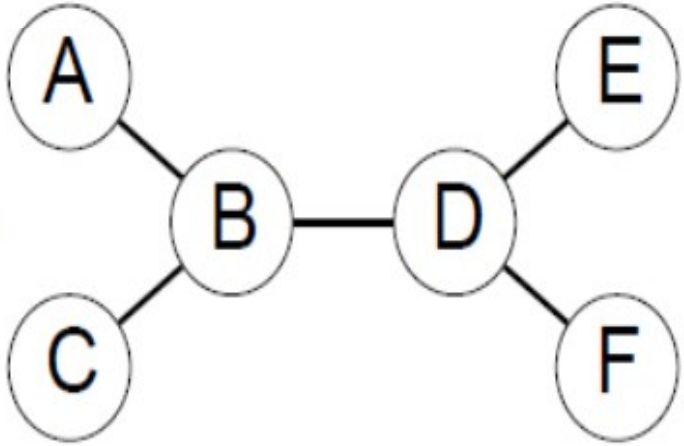
- **Backward removal phase**: check arc consistency starting from the rightmost node and going backwards
- **Forward assignment phase**: select an element from the domain of each variable going left to right. We are guaranteed that there will be a valid assignment because each arc is consistent



# Algorithm for tree-structured CSPs

**function** TREE-CSP-SOLVER(*csp*) **returns** a solution, or failure
  **inputs**: *csp*, a CSP with components $X$, $D$, $C$

  $n \leftarrow$ number of variables in $X$
  *assignment* $\leftarrow$ an empty assignment
  *root* $\leftarrow$ any variable in $X$
  $X \leftarrow$ TOPOLOGICALSORT($X$, *root*)
  **for** $j = n$ **down to** 2 **do**
    MAKE-ARC-CONSISTENT(PARENT($X_j$), $X_j$)
    **if** it cannot be made consistent **then return** *failure*
  **for** $i = 1$ **to** $n$ **do**
    *assignment*[$X_i$] $\leftarrow$ any consistent value from $D_i$
    **if** there is no consistent value **then return** *failure*
  **return** *assignment*



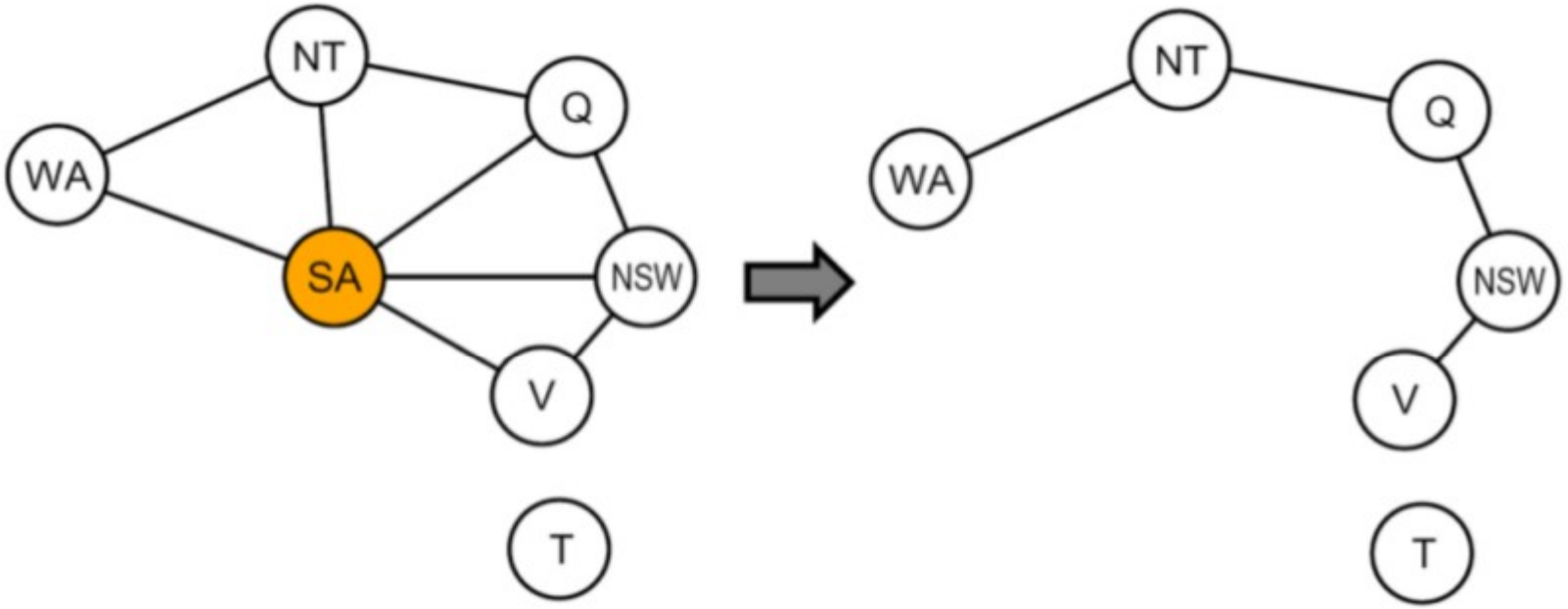
# Algorithm Complexity for tree-structured CSPs

- If $n$ is the number of variables and $D$ is the domain size, what is the running time of this algorithm?
  - $O(nD^2)$: One has to check arc consistency once for every node in the graph (every node has one parent), which involves looking at pairs of domain values
- Backtracking search for general CSPs?
  - Worst case $O(D^n)$

# Nearly tree-structured CSPs

- Can the tree-structured algorithm be extended to general constraint graph?
  - Yes. Its efficient to the CSPs, whose constraint graphs are quite close to being tree-structured
- Cutset-Conditioning: find the smallest subset of variables, the removal (from constraint graph) of which results in a tree
- Such a subset is known as smallest cutset
- Example?

# Nearly tree-structured CSPs : Example

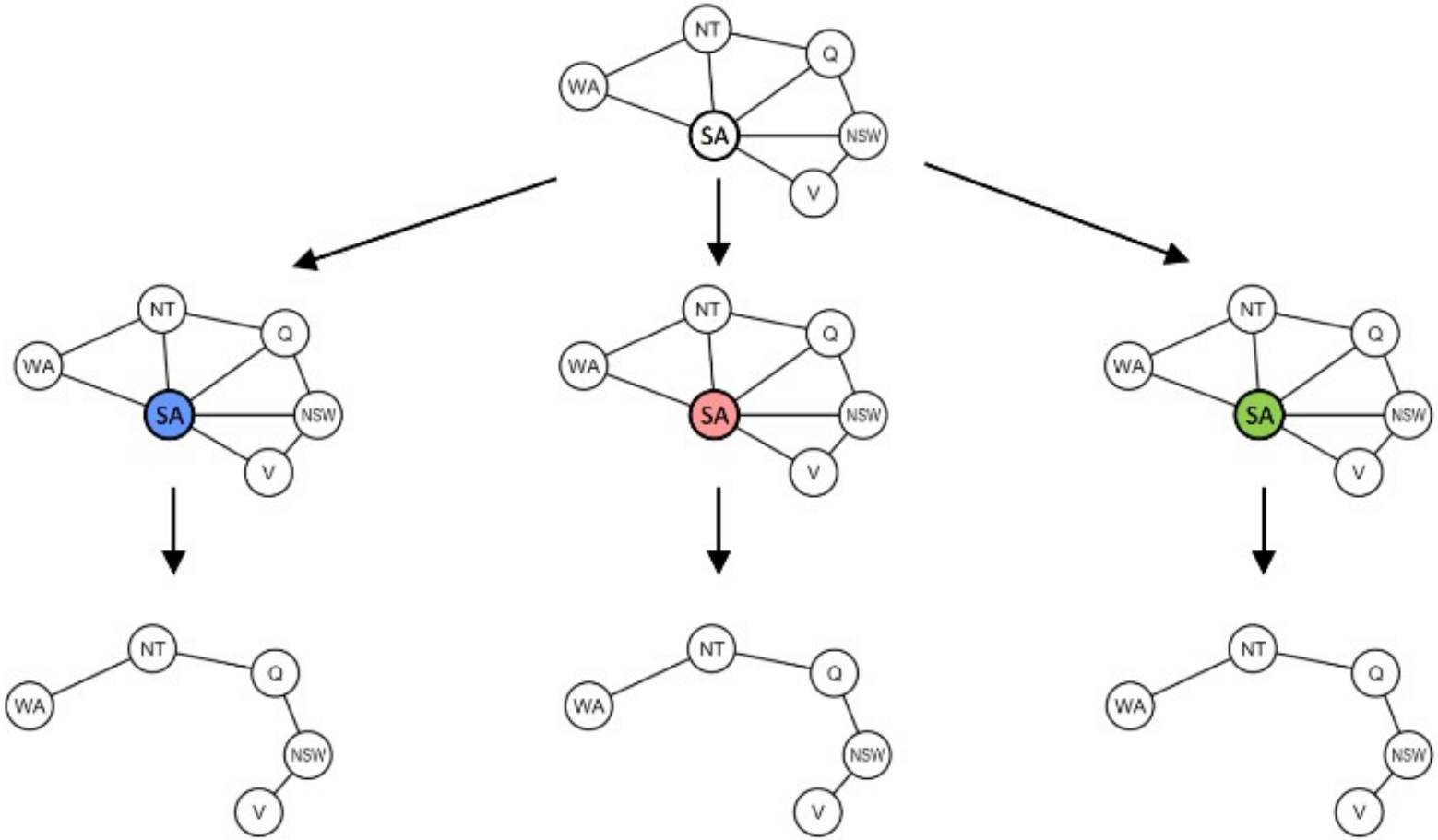



- In the map colouring problem SA can be the smallest cutset

# Nearly tree-structured CSPs : Example

- After finding the smallest cutset
  - Assign all variables in it
  - Remove the inconsistent values from the domains of all neighbouring nodes(variables)
- Running time?
  - Cutset of size c is of $O(D^c(n-c)D^2)$, Why?
  - Very fast for small c

# Nearly tree-structured CSPs : Example

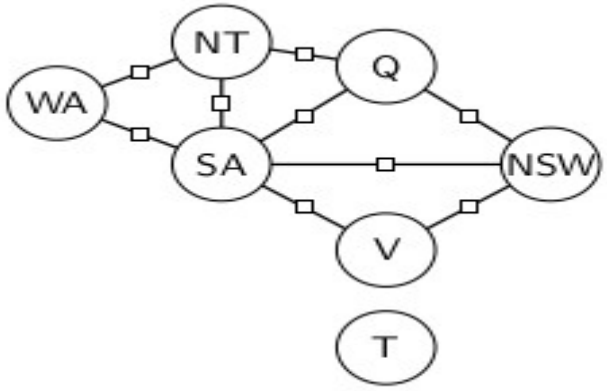


# Problem Structure

- Independent sub-problems
  - ➢Connected components
- Suppose each sub problem has c variables
  - ➢Worst case n/c problems
  - ➢Complexity $O(n/c*d^c)$ – linear in n
    - ❑E.g. n = 80, d =2, c = 20

      $2^{80}$ = 4 billion years at 10 million nodes per second

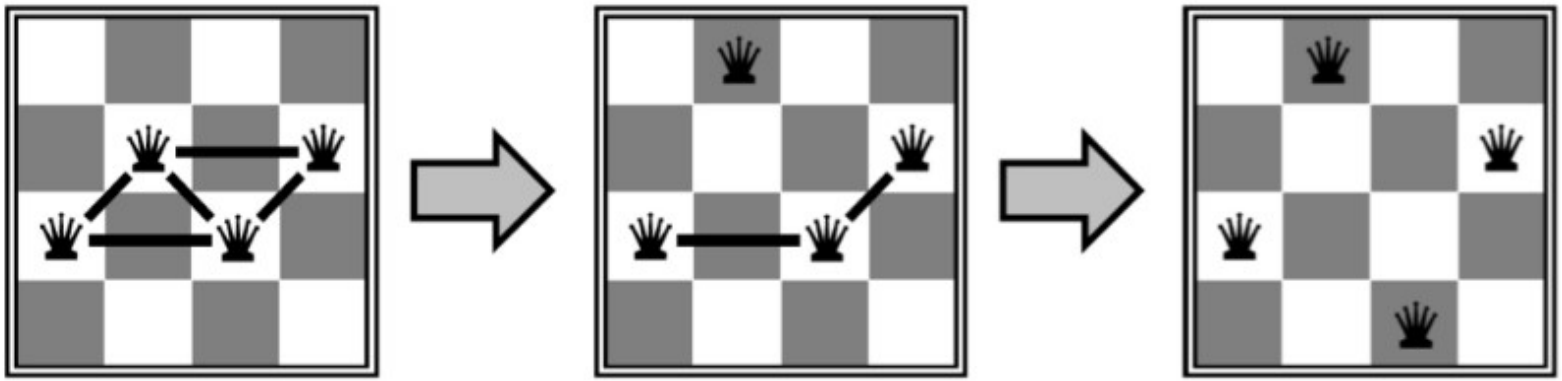
      $4*2^{20}$ = 0.4 seconds at 10 million nodes per second



# Local Search for CSPs

- Begin with “complete”-states, i.e., all variables assigned
- Initial state may be randomly chosen or by a greedy assignment process
- Search  changes the value of one variable at a time
- In order to apply to CSPs:
  - Allow states with unsatisfied constraints
  - Operators try to improve the sates by reassigning variable values
- Value selection:
  - Select any conflicted variable
  - by min-conflicts heuristic: Choose value that violates the fewest constraints
  - attempt to greedily minimize total number of violated constraints i.e. h(n) = total number of violated constraints

# Local Search for CSPs: Example




- States: 4 queens in 4 columns ($4^4$ = 256 states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: h(n) = number of attacks

# Local Search for CSPs: Example

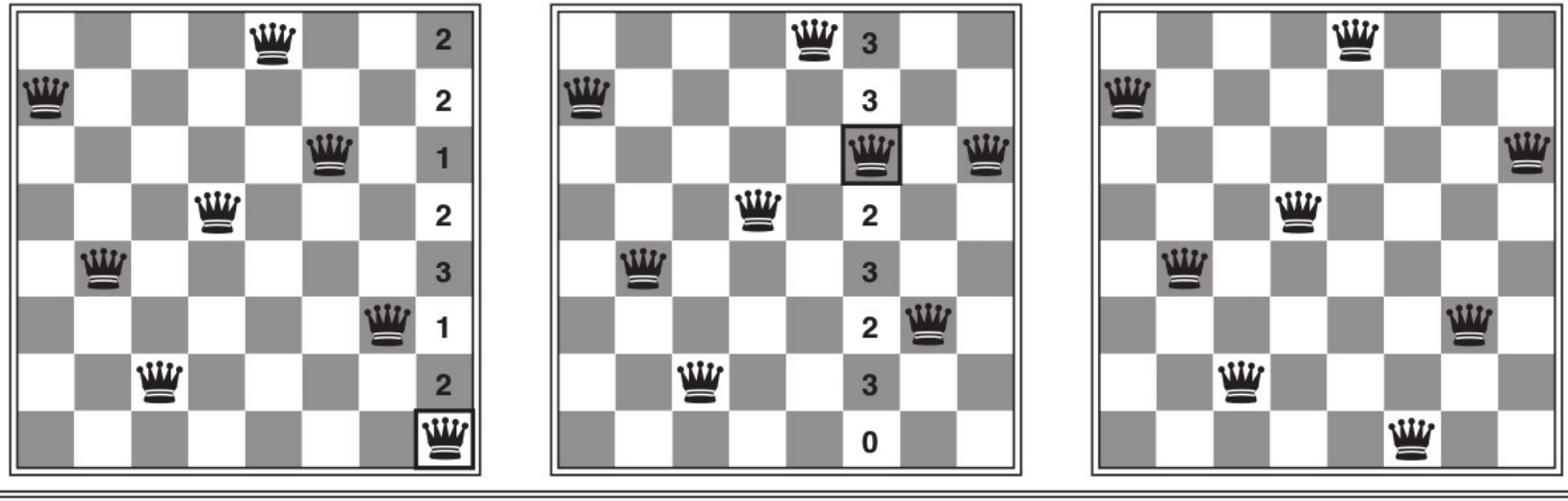
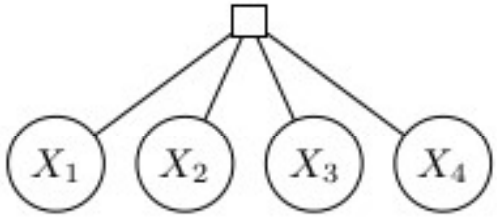



**Figure 6.9** A two-step solution using min-conflicts for an 8-queens problem. At each stage, a queen is chosen for reassignment in its column. The number of conflicts (in this case, the number of attacking queens) is shown in each square. The algorithm moves the queen to the min-conflicts square, breaking ties randomly.

# Example: N-ary Constraints

- Consider the simple problem: given n variables $X_1, \ldots, X_n$, where each $X_i \in \{0, 1\}$
- Impose the requirement that at least one $X_i = 1$
- Consider the case of n = 4




Variables: $X_1, X_2, X_3, X_4 \in \{0, 1\}$

Factor: $[X_1 \vee X_2 \vee X_3 \vee X_4]$

Examples:

$\text{Weight}(\{X_1 : 0, X_2 : 0, X_3 : 0, X_4 : 0\}) = 0$

$\text{Weight}(\{X_1 : 0, X_2 : 1, X_3 : 0, X_4 : 0\}) = 1$

$\text{Weight}(\{X_1 : 0, X_2 : 1, X_3 : 0, X_4 : 1\}) = 1$

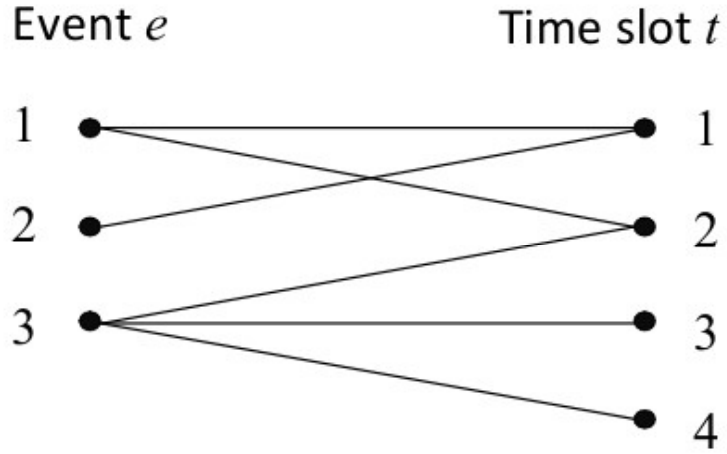
# Example: N-ary Constraints

- The idea is to break down the computation of the n-ary constraint into n simple steps
- Introduce an auxiliary variable $A_i$ for i = 1, . . . , n which represents the OR of variables $X_1$, . . . , $X_i$
- A simple recurrence updates $A_i$ with $A_{i-1}$ as:
  - $A_i = A_{i-1} \vee X_i$
- The constraint $[A_n = 1]$ enforces that the OR of all the variables is 1
- So, auxiliary variables hold the intermediate computation
- Steps: [$X_1$: 0  $X_2$: 1  $X_3$: 0  $X_4$ : 1]
  - Initialization: $[A_0 = 0]$
  - Processing:$[A_i = A_{i-1} \vee X_i]$
  - Final output: $[A_4 = 1]$

| i | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $X_i$: | | 0 | 1 | 0 | 1 |
| $A_i$: | 0 | 0 | 1 | 1 | 1 |

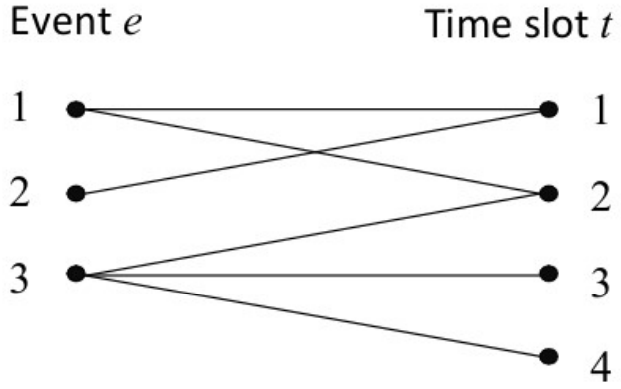
# Example: Event Scheduling

- E (distinct)events that has to be scheduled into T time slots
- Three conditions:
  - Every (and exactly one) event e must be scheduled into a time slot t
  - Every time slot t can have at most one event (zero is possible)
  - One is given a fixed set A of (event, time slot) pairs which are allowed i.e. Event e only allowed at time slot t if (e, t) is in A



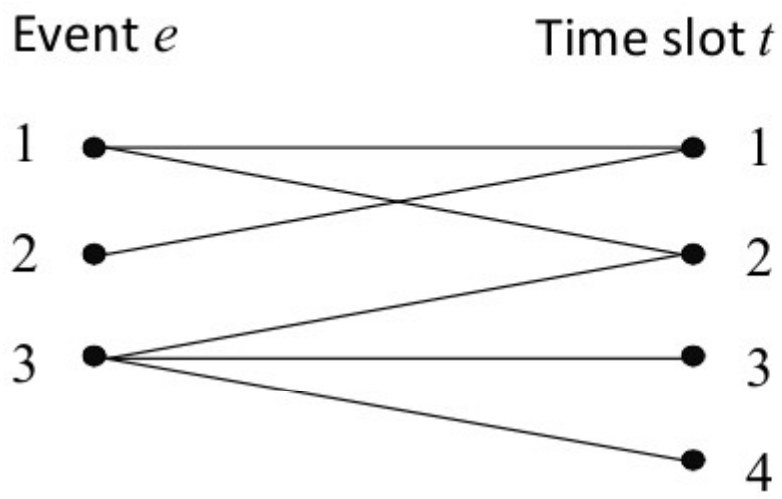
# Example: Event Scheduling

- Approach 1:
  - Variables: for each event e, $X_e \in \{1, \ldots, T\}$
  - Constraints (only one event per time slot): for each pair of events $e \neq e'$, enforce $[X_e \neq X_{e'}]$

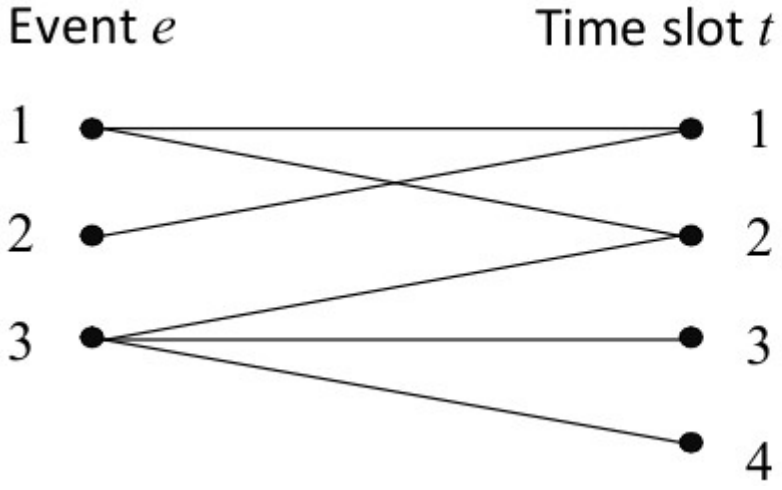



  - Constraints (only scheduled allowed times): for each event e, enforce $[(e, X_e) \in A]$
  - Note that we end up with E variables with domain size T, and $O(E^2)$ binary constraints

# Example: Event Scheduling

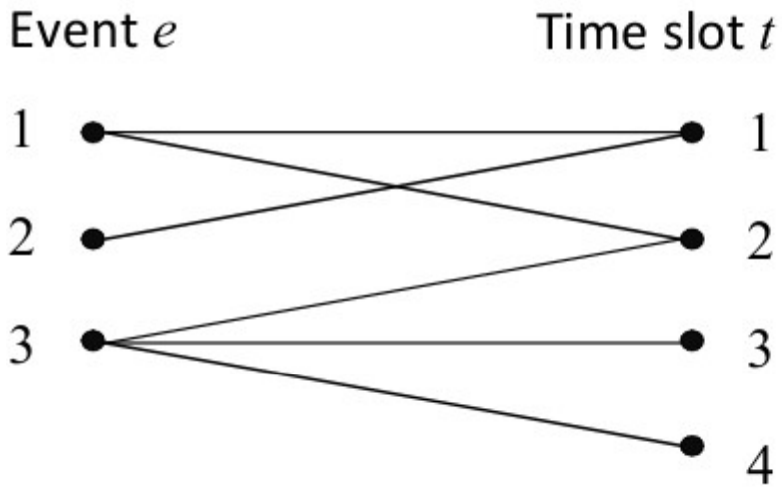



$\{X_1 : 1, X_2 : 1, X_3 : 3\}$

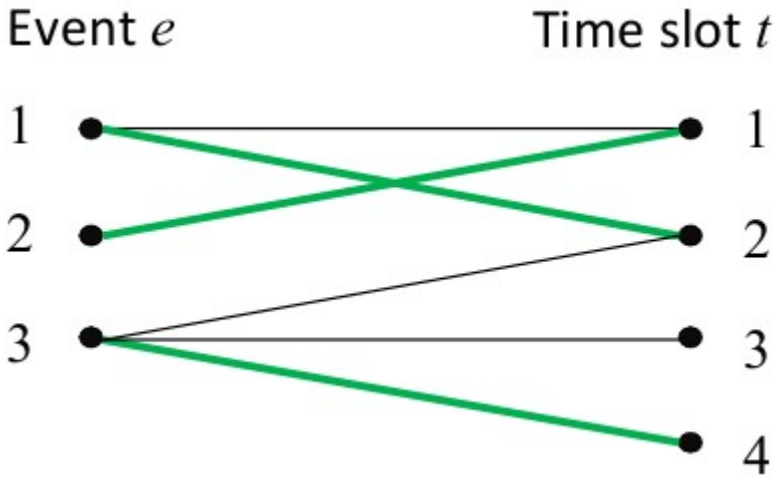



$\{X_1 : 1, X_2 : 4, X_3 : 3\}$

# Example: Event Scheduling




$\{X_1 : 2, X_2 : 1, X_3 : 4\}$



# Example: Event Scheduling

- Approach 2:
  - Variables for each time slot t, $Y_t \in \{1, \ldots, E\} \cup \{\emptyset\}$
  - Constraints (each event is scheduled exactly once): for each event e, enforce [$Y_t$ = e for exactly one t]
  - Constraints (only schedule allowed times): for each time slot t, enforce [$Y_t = \emptyset$ or $(Y_t, t) \in A$]
  - How many variables, constraints? What is the domain size?

# Home Work Exercise

- Three sculptures (A, B, C) are to be exhibited in rooms 1, 2 of an art gallery
- The exhibition must satisfy the following conditions:
  - Sculptures A and B cannot be in the same room
  - Sculptures B and C must be in the same room
  - Room 2 can only hold one sculpture

# Why CSPs

| Atomic State Space | CSP |
| --- | --- |
| ○ Highly problem specific | General Purpose CSP Solver |
| ○ State space is large e.g.{SA = blue} - $3^5$ = 243 | Large state space can be eliminated e.g.{SA = blue} - $2^5$ = 32 |
| ○ Search is slower | Search is faster |
| ○ If state is not a goal state, we explore it | If partial assignment is not a legal assignment, stop further exploration |
| ○ Intractable problems | Locally optimal solution |

# Real World CSPs

- Scheduling of events:
  - Flight/Train scheduling
  - Staff rostering at a company
  - Course/Class time table at a university
  - In an operating system
- Games/Puzzles:
  - n-Queens problem, Knights tour problem, Sudoku, Crosswords etc
- Finance
  - Linear Programming

# Summary

- In CSPs :
  - States defined by values of a fixed set of variables
  - Goal test defined by constraints on variable values
- Backtracking Search:
  - Like DFS
  - May be improved further by incorporating:
    - Variable ordering and value selection heuristics
    - Forward checking prevents assignments that guarantee later failure
    - Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect early failure
- Local Search:
  - Alternative to backtracking search
  - Fast
  - Suboptimal
- Complexity of CSP
  - NP-complete in general (exponential worst-case running time)
  - Efficient solutions possible for special cases (e.g., tree-structured CSPs)

# References

- CS-188 UCB lecture notes/slides
- Professor Mausam AI course
- “Artificial Intelligence – A Modern Approach” Russell and Norvig

# Probability

- Set = A collection of unique objects
  - Example:
    - A finite set: {Sun, Mon, Tue, Wed, Thu, Fri, Sat}
    - An infinite set of abstract things: {1, 2, 3, ...}
  - Order doesn't matter
  - Important Sets: Natural numbers, integers, rational numbers, real numbers
- Random Experiment
  - An experiment whose outcome is uncertain
  - Example:
    - Coin Toss
    - Die roll
    - Weather (temperature, rain) tomorrow
    - Lifetime of a laptop battery
    - Intensity of a randomly chosen pixel in an image

# Probability

- Sample Space
  - A set Ω of all possible outcomes of a random experiment
  - Examples:
    - Discrete: Coin toss Ω = {*H,T*}, Die roll Ω = {1,2,3,4,5,6}
    - Discrete: Number of tosses required to get a "head" (This discrete set is a set of natural numbers, which has cardinality infinite)
    - Continuous: Temperature tomorrow: Ω = (0, 1.416785 x $10^{32}$) Kelvin. Absolute zero to Plank temperature

# Probability

- Event
  - A subset of the sample space Ω
  - Example
    - Event A = The die roll produces an even number
    - Event A = The temperature is between 298K and 300K (25 and 30 degree Celsius)
    - Event A = Intensity at a chosen pixel (in a grayscale image) falls between 10 and 20

# Probability

- Operations on Events:
  - Union = "or" = $A \cup B$
  - Intersection = "and" = $A \cap B$
  - Complement = "negation" = $\overline{A}$
  - Subtraction = $A$ happens but $B$ doesn't happen = $A$-$B$

# Probability

- Event Space:
  - Space of all possible events (that one likes to consider/model) $F \subseteq 2^{\Omega}$
  - Given a sample space Ω, the event space *F* must satisfy several rules (for consistency and to deal with sample spaces that are infinite)
    - Null Set $\emptyset \in F$
    - Closed under Countable Union: if $A_1, A_2, A_3, \ldots \in F$, then $A_1 \cup A_2 \cup A_3 \cup \ldots \in F$
      (We can have a union of only a countably infinite set here)
    - Closed under Complementation: if $A \in F$ then $\overline{A} \in F$
      (complementation is with respect to Ω, i.e. $A = \Omega - \overline{A}$ )
  - A set *F* that satisfies the above rules is called a σ-algebra

# Event Space

- Note: $2^{\Omega}$ is the power-set notation
  - The number of all possible subsets of a finite set of size n is $2^{n}$
- The term algebra is used because the theory defines an algebra on sets (union, intersection , differences), similar to the algebra on numbers
- Example: Ω = {a, b, c, d}. One possible event space on Ω is *F* = {ø, {a, b}, {c, d}, {a,b,c,d}}

# Probability Measure

- Gives a precise notion of "size" or volume of the sets
- A probability measure on an event space $F$ is a function $P$: $F \rightarrow [0,1]$ such that:
  - $P(\emptyset) = 0$
  - $P(\Omega) = 1$ (this needn't hold for a general (non-probability) measure)
  - For pairwise disjoint sets(events) $A_1, A_2, A_3, \ldots \epsilon F$, we have $P(A_1 \cup A_2 \cup A_3 \cup \ldots) = P(A_1) + P(A_2) + P(A_3) + \ldots$
  - In simple words, the probability measure on a sample space $\Omega$ assigns every event $A \subseteq \Omega$, a number in [0,1], such that $P(\Omega) = 1$ and $P(A \cup B) = P(A) + P(B)$ for disjoint $A, B$
  - This number $\epsilon$ [0, 1] assigned to an event is nothing but the probability/chance that the event occurs

# Properties of Probability Measure

- Complement of an event A: $P(A) = 1 - P(A)$
- Union of two overlapping events:
  - $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- Difference of events:
  - $P(\text{A-B}) = P(\text{A}) - P(A \cap B)$

# Probability Space

- A triple ($\Omega$, $F$, $P$) where:
  - $\Omega$ is the sample space (space of all possible outcomes)
  - $F$ is the event space (space of all possible events)
  - $P$ is the probability measure on ($\Omega$, $F$) with $P(\Omega) = 1$
- Example:
  - Toss a fair coin : What is $\Omega$ ? What is *F?* What is *P?*
  - Roll a fair die: $P(\{1\}) = 1/6$. All outcomes equally likely. $P(\{1,2,3\}) = 1/2$

# Conditional Probability

- The conditional probability of event A given event B is $P(A|B) = P(A \cap B) / P(B)$
- Example:
  - Dice example: $A = \{2\}$, $B = \{2, 4, 6\}$, $P\{A\} = 1/6$, $P(A|B) = 1/3$
  - Image Generation process: For a pixel first choose a label (e.g. sky, mountain), then choose a color given the label
  - Image example 2 (Image-Acquisition Process): Observed intensity is some randomly perturbed version of the true intensity, because of measurement errors. P ( observed intensity at pixel "p" | true intensity at pixel "p" ) = noise model.

# Random Variable

- A function defined on a probability space $(\Omega, F, P)$ (Name "random variable" is a misnomer)
- $X: \Omega \rightarrow V$ where V is the set of numeric values e.g. set of integers or real numbers
- Example:
  - Roll one die and take the number. Here $X(\cdot)$ is identity.
  - Roll one die and square the number. What is $\Omega$? What is $X(\cdot)$?
  - Image example: Pick a pixel location at random and take the intensity value at that pixel. What is $\Omega$, X?

# Discrete Random Variable

- Values taken by Random Variable form a discrete set
- The cardinality of the set of possible values can be finite or (countable) infinite
- Sample space may be discrete or continuous
- Example:
  - Roll a die and take a number on top face (finite sample space)
  - Number of packets transmitted by a computer on a network

# Continuous Random Variable

- The Random Variable takes a continuous range of values
- The cardinality of the set of possible values is (uncountable) infinite
- Cardinality of the sample space must be uncountable infinite, Why?
- Example:
  - Heights of children in school
  - Width of a leaf
  - Standard Normal random variable z

# Cumulative Distribution Function(CDF)

- For a real-valued random variable X, the CDF $f(x) = P(X \leq x)$
- $\forall x, f(x) \in [0,1]$
- Properties of CDFs:
  - $f$ monotonically non-decreasing
  - $f$ is right continuous. $\lim_{\epsilon \to 0^+} f(x+\epsilon) = f(x), \forall x \epsilon \mathrm{R}$
  - $\lim_{x \to -\infty} f(x) = 0$
  - $\lim_{x \to +\infty} f(x) = 1$

# Probability Mass Function

- $p(x) = P(X=x)$, $\forall x, p(x) \in [0,1]$
- The CDF can be defined in terms of PMF:
  - $f(x) = P(X \leq x) = \sum_{y \leq x} p(y)$
- Examples:
  - Bernoulli Distribution:
    - Radom Variable = success/failure at a task

      $p(x = 1) = \alpha$, $p(x = 0) = 1 - \alpha$
  - Binomial Distribution:
    - Random Variable = number of successes observed in multiple tries

      Probability of getting *k* success from *n* trials:

      $p(x = k) = C_k^n \alpha^k (1 - \alpha)^{n-k}$

# Probability Density Function

- $p(x) = \frac{df(x)}{dx}$, where the derivative exists
  - As the CDF $f(x)$ is non-decreasing, $p(x) \geq 0$
- The CDF can be defined in terms of the pdf $p(x)$: $f(x) = P(X \leq x) = \int_{-\infty}^{x} p(y)dy$
- $\lim_{x \to +\infty} f(x) = 1 \Rightarrow \int_{-\infty}^{\infty} p(y)dy = 1$
- PDF values are not probabilities. Probabilities are obtained by integrating the PDF

# Examples of PDF

- Exponential Distribution:
  - Consider random events occurring at a constant average rate $\lambda > 0$; the events occur continuously and independently of each other
  - Packets arriving at a router
  - Exponential distribution models the probability of the time elapsed between two consecutive events
  - PDF:

    $p(x) = 0, \forall x < 0,$

    $p(x) = \lambda \exp(-\lambda x), \forall x \geq 0$
  - CDF:

    $f(x) = 0, \forall x < 0$

    $f(x) = 1 - \exp(-\lambda x), \forall x \geq 0$

# Examples of PDF

- Gaussian(normal) Distribution
  - $p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$, with parameters μ, σ
  - A standard normal distribution has μ = 0, σ = 1
  - Prove that Gaussian integrates to 1. (Home work exercise)

# Joint, Marginal Distribution

- Joint Random Variable:
  - Given two Random Variables X and Y sharing the same probability space ($\Omega$, $F$, P), one has to construct a suitable shared probability space in order to define the joint random variables
  - Example:
    - (Independent RVs) X ≡ coin toss, Y ≡ die roll. Then one can construct the probability space as follows:
      - $\Omega = \Omega_1 \times \Omega_2 = \{(i,j): i \in \{-1, 1\}, j \in \{1, 2, 3, 4, 5, 6\}\}$
      - $F = F_1 \times F_2$
      - $P(\{(X, Y) = (i, j)\}) = P(\{X = i\})\, P(\{Y = j\})$
      - $X((i,j)) = f(i)$ (could be any function of i)
      - $Y((i,j)) = g(j)$ (could be any function of j)

# Joint, Marginal Distribution

- Joint CDF and Joint PMF/PDF
  - Joint CDF: $f(x,y) = P(X \le x, Y \le y) = P(X \le x) \cap P(Y \le y)$
  - Joint PMF(discrete): $p(x,y) = P(X = x, Y = y)$
  - Joint PDF (continuous): $p(x,y) = d^2 f(x,y) / dxdy$
- Marginal Distribution
  - Marginal PMF: $p(x) = \sum_j p(x,y_j)$
    - As $P(\{X = x\}) = P(\{X = x\}, \{Y \in \{y_1, y_2 \ldots\}\}) = \sum_j P(\{X = x\}, \{Y = y_j\})$
  - Marginal PDF : $p(x,y) = \int_{-\infty}^{\infty} p(x,y) dy$

# Example

- Consider a 1D binary image of 10 pixels with intensities [0, 0, 0, 1, 1, 1, 1, 1, 0, 0,]
  - Pick a pixel location s at random, X(s) = intensity at location s. Y(s) = intensity at location s + 1
  - $\Omega = \{1, 2, \ldots, 10\}$, $F$, $P(s \in \Omega) = 1/10$
  - Joint PMF (table): $P(x = 0, y = 0) = 0.4$, $P(x = 1, y = 1) = 0.4$, $P(x = 1, y = 0) = 0.1$, $P(x = 0, y = 1) = 0.1$
  - Marginal PMF: $P(x = 0) = 0.5$, $P(x = 1) = 0.5$;

    $P(y = 0) = 0.5$, $P(y = 1) = 0.5$