- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)
- Consistency implies admissibility

Strategies: DFS: stack BFS: queue UCS: cumulative cost

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal

- Admissibility: heuristic cost ≤ actual cost to goal

  h(A) ≤ actual cost from A to G

- Consistency: heuristic "arc" cost ≤ actual cost for each arc

  h(A) – h(C) ≤ cost(A to C)

  - The f value along a path never decreases

  h(A) ≤ cost(A to C) + h(C)

---

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

---

Variables: WA, NT, Q, NSW, V, SA, T

Domains: $D = \{\text{red}, \text{green}, \text{blue}\}$

Constraints: adjacent regions must have different colors

Implicit: $\text{WA} \neq \text{NT}$

Explicit: $(\text{WA}, \text{NT}) \in \{(\text{red}, \text{green}), (\text{red}, \text{blue}), \ldots\}$

Solutions are assignments satisfying all constraints, e.g.:

{WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}

Discrete Variables
- Finite domains
  - Size $d$ means $O(d^n)$ complete assignments
  - E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)
- Infinite domains (integers, strings, etc.)
  - E.g., job scheduling, variables are start/end times for each job
  - Linear constraints solvable, nonlinear undecidable

Continuous variables
- Linear constraints solvable in polynomial time by Linear Programming methods

- Varieties of Constraints
  - Unary constraints involve a single variable (equivalent to reducing domains), e.g.: $\text{SA} \neq \text{green}$
- Binary constraints involve pairs of variables, e.g.:

  $$\text{SA} \neq \text{WA}$$

- Higher-order constraints involve 3 or more variables:
  e.g., cryptarithmetic column constraints

## Backtracking Search

**Filtering: Keep track of domains for unassigned variables and cross off bad options.**
**Forward checking: Cross off values that violate a constraint when added to the existing assignment.**

```
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 ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

## Arc Consistency

An arc X → Y is consistent iff for *every* x in the tail there is *some* y in the head which could be assigned without violating a constraint

*Remember: Delete from the tail!*

- A simple form of propagation makes sure all arcs are consistent:
- Important: If X loses a value, neighbors of X need to be rechecked!
- Arc consistency detects failure earlier than forward checking

## MRV

- Variable Ordering: Minimum remaining values (MRV):
  - Choose the variable with the fewest legal left values in its domain

## Tree-Structured CSPs

- Remove backward: For i = n : 2, apply RemoveInconsistent(Parent($X_i$),$X_i$)
- Assign forward: For i = 1 : n, assign $X_i$ consistently with Parent($X_i$)

- Claim1: After backward pass , all root-to-leaf arcs are consistent
- Claim2: If root-to-leaf arcs are consistent, forward assignment will not backtrack

---

## Linear Classifiers: Perceptrons

$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- Inputs are feature values
- Each feature has a weight
- Sum is the activation

- If the activation is:
  - Positive, output +1
  - Negative, output -1

## Binary Perceptron

- Start with weights = 0
- For each training instance:
- Classify with current weights

$$y = \begin{cases} +1 & \text{if} \quad w \cdot f(x) \geq 0 \\ -1 & \text{if} \quad w \cdot f(x) < 0 \end{cases}$$

- If correct (i.e., y=y*), no change!
- If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

$$w = w + y^* \cdot f$$

## Multiclass Perceptron

- Start with weights = 0
- For each training instance:

- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

$$y = \arg\max_y \; w_y \cdot f(x)$$

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$

# Markov Decision Processes (MDP)

- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ A
  - A transition function T(s, a, s')
    - Probability that a from s leads to s', i.e., P(s' | s, a)
    - Also called the model or the dynamics
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state
  - Maybe a terminal state

- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search

The value (utility) of a state s:

$V^*(s)$ = expected utility starting in s and acting optimally

The value (utility) of a q-state (s,a):

$Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

The optimal policy:

$\pi^*(s)$ = optimal action from state s

Bellman Equations:

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

Value Iteration:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

Policy Iteration:

$$V_0^\pi(s) = 0$$

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s,\pi_i(s),s') \left[R(s,\pi_i(s),s') + \gamma V_k^{\pi_i}(s')\right]$$

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi_i}(s')\right]$$

---

Sample-Based Policy Evaluation:

$$sample_1 = R(s,\pi(s),s_1') + \gamma V_k^\pi(s_1')$$

$$sample_2 = R(s,\pi(s),s_2') + \gamma V_k^\pi(s_2')$$

$$\cdots$$

$$sample_n = R(s,\pi(s),s_n') + \gamma V_k^\pi(s_n')$$

$$V_{k+1}^\pi(s) \leftarrow \frac{1}{n}\sum_i sample_i$$

## Known MDP: Offline Solution

| Goal | Technique |
| --- | --- |
| Compute V*, Q*, $\pi$* | Value / policy iteration |
| Evaluate a fixed policy $\pi$ | Policy evaluation |

## Unknown MDP: Model-Based

| Goal | Technique |
| --- | --- |
| Compute V*, Q*, $\pi$* | VI/PI on approx. MDP |
| Evaluate a fixed policy $\pi$ | Approx. MDP |

## Unknown MDP: Model-Free

| Goal | Technique |
| --- | --- |
| Compute V*, Q*, $\pi$* | Q-learning |
| Evaluate a fixed policy $\pi$ | Value Learning |

Temporal Difference Learning:

Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

Q-Learning:

$$sample = R(s, a, s') + \gamma \max_{a'} Q(s', a')$$

$$Q(s, a) \leftarrow (1-\alpha)Q(s, a) + (\alpha)\,[sample]$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a) - Q(s_t, a_t)]$$

SARSA:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Probability Distributions:

$$\forall x \quad P(X=x) \geq 0 \quad \text{and} \quad \sum_x P(X=x) = 1$$

Joint Distributions:

- Must obey: $P(x_1, x_2, \ldots x_n) \geq 0$

$$\sum_{(x_1, x_2, \ldots x_n)} P(x_1, x_2, \ldots x_n) = 1$$

Normalization Trick:

1. **SELECT** the joint probabilities matching the evidence.

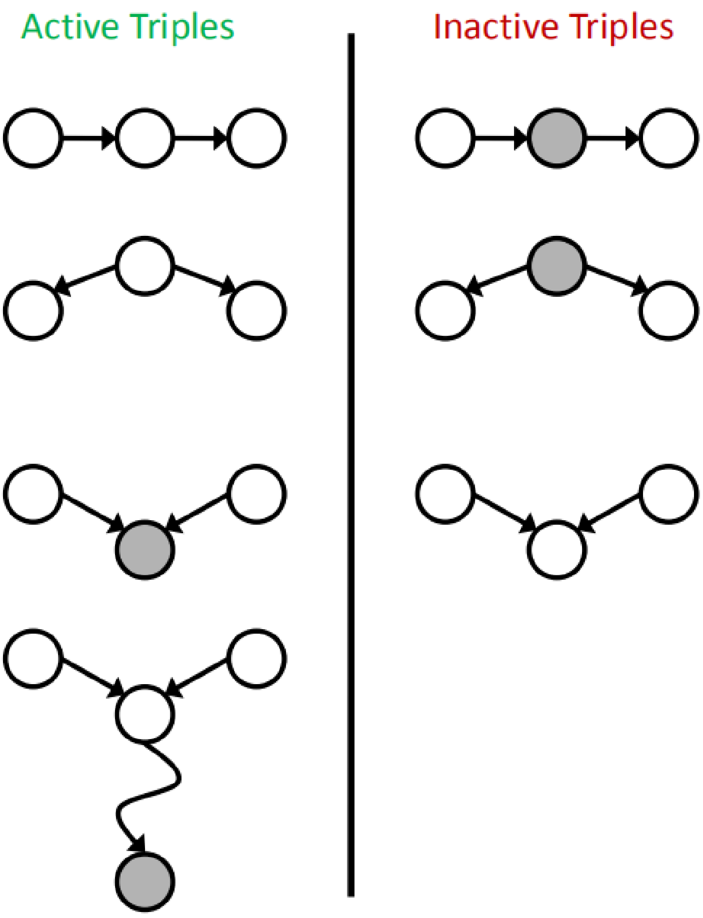
2. **NORMALIZE** the selections (make them sum to 1)

- Conditional probability $\quad P(x|y) = \dfrac{P(x,y)}{P(y)}$
- Product rule $\quad P(x,y) = P(x|y)P(y)$
- Chain rule

$$\begin{aligned} P(X_1, X_2, \ldots X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_1, X_2)\ldots \\ &= \prod_{i=1}^{n} P(X_i|X_1, \ldots, X_{i-1}) \end{aligned}$$

- X, Y independent if and only if: $\quad \forall x, y : P(x,y) = P(x)P(y)$

D-Separation:




1. Prior Sampling:

全部采样，然后算每个出现的概率

2. Rejection Sampling:

采样时把不满足 evidence 的样本舍弃，然后计算每个出现的概率

3. Likelihood Weighting（从上到下）

- if $X_i$ is an evidence variable
  - $X_i$ = observation $x_i$ for $X_i$
  - Set w = w * $P(x_i \mid \text{Parents}(X_i))$
- else
  - Sample $x_i$ from $P(X_i \mid \text{Parents}(X_i))$

4. Gibbs Sampling（上下都可）

- Choose a non-evidence variable X
- Resample X from P( X | all other variables)

$$P(S|+c,+r,-w) = \frac{P(S,+c,+r,-w)}{P(+c,+r,-w)}$$

Markov Chain: Stationary Distributions

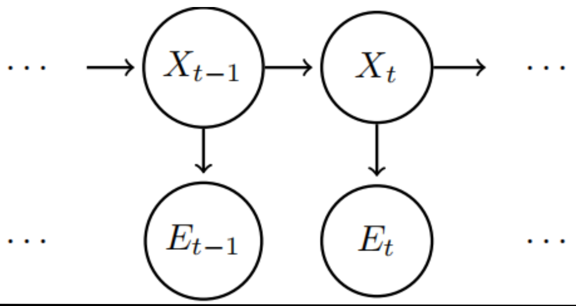
$$P_\infty(X) = P_{\infty+1}(X) = \sum_x P(X|x)P_\infty(x)$$

Page Rank:

$$PR_{t+1}(P_i) = \sum_{P_j} \frac{PR_t(P_j)}{C(P_j)},$$ 其中 $PR_t(P_j)$ 初始为 1/结点数，$C(P_j)$ 为结点出度

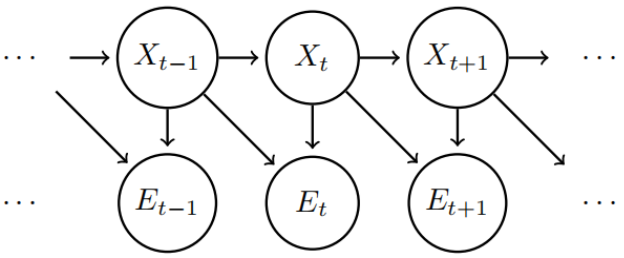
Transition Matrix: $P_{ij}$ 表示第 i 行第 j 列为第 i 个结点转移到第 j 个元素的概率

Hidden Markov Models: The Forward Algorithm



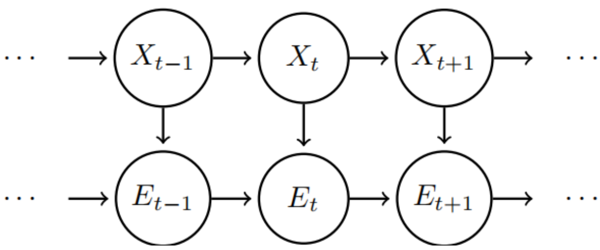

$$P(x_t|e_{1:t}) \propto P(x_t, e_{1:t})$$

$$= P(e_t \mid x_t) \sum_{x_{t-1}} P(x_t \mid x_{t-1}) P(x_{t-1}, e_{1:t-1})$$



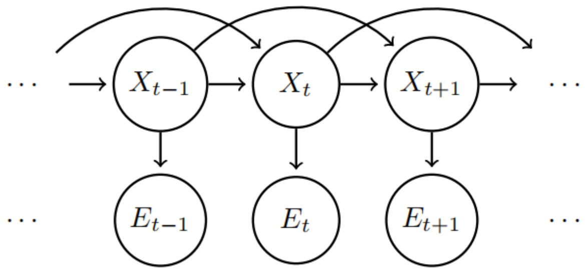

$$P(x_t|e_{1:t}) \propto P(x_t, e_{1:t})$$

$$= \sum_{x_{t-1}} P(e_t \mid x_t, x_{t-1}) P(x_t \mid x_{t-1}) P(x_{t-1}, e_{1:t-1})$$




$$P(x_t|e_{1:t}) \propto P(x_t, e_{1:t})$$

$$= \sum_{x_{t-1}} P(e_t \mid x_t, e_{t-1}) P(x_t \mid x_{t-1}) P(x_{t-1}, e_{1:t-1})$$




$$P(x_t, x_{t+1}|e_{1:t}) \propto P(x_t, x_{t+1}, e_{1:t})$$

$$= \sum_{x_{t-1}} P(e_t \mid x_t) P(x_{t+1} \mid x_t, x_{t-1}) P(x_t, x_{t-1}, e_{1:t-1})$$

$$P(X_2)$$

$$P(x_2) = \sum_{x_1} P(x_1, x_2)$$

$$= \sum_{x_1} P(x_1) P(x_2|x_1)$$

$$P(X_1|e_1)$$

$$P(x_1|e_1) = P(x_1, e_1)/P(e_1)$$

$$\propto_{X_1} P(x_1, e_1)$$

$$= P(x_1)P(e_1|x_1)$$

## Particle Filtering

- Our representation of P(X) is now a list of N particles (samples)
- Each particle is moved by sampling its next position from the transition model

$$x' = \text{sample}(P(X'|x))$$

- Similar to likelihood weighting, downweight samples based on the evidence

$$w(x) = P(e|x)$$

$$B(X) \propto P(e|X)B'(X)$$

- **Rather than tracking weighted samples, we resample;**
- **N times, we choose from our weighted sample distribution (draw with replacement);**
- **This is equivalent to renormalizing the distribution;**
- **Now the update is complete for this time step, continue with the next one.**

---

## Naïve Bayes

|Y| parameters

$$P(\mathsf{Y}, \mathsf{F}_1 \dots \mathsf{F}_n) = \quad P(\mathsf{Y}) \prod_i P(\mathsf{F}_i|\mathsf{Y})$$

|Y| x |F|$^n$ values

n x |F| x |Y| parameters

$$P(Y, f_1 \dots f_n) = \begin{bmatrix} P(y_1, f_1 \dots f_n) \\ P(y_2, f_1 \dots f_n) \\ \vdots \\ P(y_k, f_1 \dots f_n) \end{bmatrix} \Rightarrow \frac{\begin{bmatrix} P(y_1) \prod_i P(f_i|y_1) \\ P(y_2) \prod_i P(f_i|y_2) \\ \vdots \\ P(y_k) \prod_i P(f_i|y_k) \end{bmatrix}}{P(f_1 \dots f_n)} \Rightarrow P(Y|f_1 \dots f_n)$$

+

$$\textit{prediction}(f_1, \cdots f_n) \quad = \underset{y}{\operatorname{argmax}}\, P(Y=y) \prod_{i=1}^{n} P(F_i = f_i | Y = y)$$

## Laplace Smoothing

$$P_{LAP,k}(x) = \frac{c(x) + k}{N + k|X|}$$

$$P_{LAP,k}(x|y) = \frac{c(x,y) + k}{c(y) + k|X|}$$