Uninformed Search

Fringe: priority queue saving paths to be evaluated (initial: start)

Expand: dequeue the path with higher priority, get paths = path + each successor

Strategy: grant these paths with priorities according to our strategy, enqueue them

Loop until: goal in the pre_enqueued path

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

Informed Search

Heuristics: estimated cost from this node to the goal (information)

Greedy: No fringe, always select path whose new-added node has the smallest heuristic

A* Search: UCS + Greedy

Strategy: f(n) = g(n) (cumulative cost -> optimal) + h(n) (heuristics -> fast)

Loop until: goal in the dequeued path

Tree: critical h guaranteeing optimality ≤ actual least cost to the goal

Graph: h(A) – h(C) ≤ actual least cost from A to C (never search a node twice, must reach the optimal node first)

Constraint Satisfaction Problems

Backtrack (Filter reduces backtrack)

1. Assign a variable (with min remaining value) at a time (to the least constraining value)
2. Check consistency: if this assignment violates the rule, backtrack
3. Filter: check arc-consistency from every neighbor to this variable (X->Y consistent iff for every x in X, there is y in Y that satisfies the rule) If not, delete from the tail, that is, delete x from X, recheck neighbors after every deletion

Tree structure for acyclic CSP (Structure prevents backtrack)

1. Build a search tree with whatever node as root
2. Remove backward: for i= n : 2, enforce arc-consistency Parent(Xi)->Xi, delete from the tail (parent) and recheck!
3. Assign forward: for i= 1 : n, assign Xi consistent with Parent(Xi)

Local Search

Adopt a random assignment and gradually reduce violation with strategies

Strategies: Hill climbing: gradient ascend
Simulated annealing: reducing learning rate of gradient ascend
Genetic algorithm: select by fitness, cross over, mutation

Adversarial Search

Minimax & α-β prune

1. Initialize the α = -inf and β = +inf for the root node.
2. Pass down the α and β down to the searching node A (with leaves as children).
3. Search every child of A (M stands for max or min).
4. After searching a child, update values for A
5. If A is MAX: v = max (values searched), α = max(v, α), If A is MIN: v = min (values searched), β = min(v, β)
6. If (A is MAX and v ≥ β) or (A is MIN and v ≤ α), prune the remaining branches of A.
7. After searching (or pruning) every child A, pass its v up to A's parent and loop 3-7 for A = A's parent
9. Keep doing the steps above until every branch of the root node has been searched or pruned.

Expectimax: chance node's utility = weighted sum of its children, no prune

Bayes Network

D-Seperation

X and Y is conditionally independent iff all paths from X to Y are inactive, a path is active iff all triples along the path is active

Inference (variable elimination)

1. Instantiate evidences, delete lines violating the evidences
2. While there's still hidden variables, select one, join all tables with it (left or right), sum over it, get P(lefts | rights)
3. Join the remaining tables and normalize

Sampling

Prior sampling: without evidence, sample from root to leaves, count and get p

Rejection sampling: with evidence, sample from root to leaves, delete samples violating the evidence, count and get p

Likelihood weighting: sample from root to leaves, w = w* p's of evidences, count and get p, p = p * w

Gibbs sampling: fix evidence, randomly assign all variables, randomly choose a variable, resample it given its CPT

MDP

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

$$V^*(s) = \max_a Q^*(s,a) \qquad \pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s,a)$$

Value iteration: start with zero values, loop

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

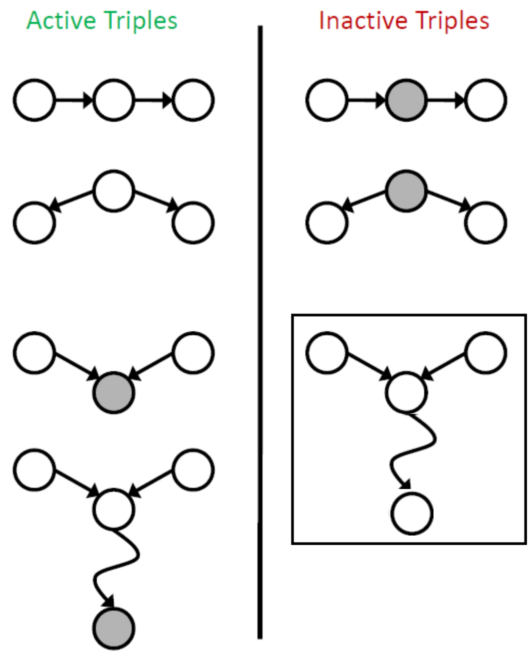
Policy iteration: start with zero values and random policy π_i, loop

1. Policy evaluation, loop

$$V_{k+1}^{\pi_i}(s) = Q_k(s,\pi(s)) = \sum_{s'} T(s,\pi(s),s')\,[R(s,\pi(s),s') + \gamma V_k^{\pi_i}(s')]$$

2. Policy improvement

$$\pi_{k+1}(s) = \underset{a}{\operatorname{argmax}}\, Q_k(s,a) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')\,[R(s,a,s') + \gamma V^{\pi_i}(s')]$$



Reinforcement Learning

Q learning and Sarsa: choose initial s, choose a from s, loop

1. take action a, receive samples (s,a,s',r)
2. Choose a' from s' (How?)
3. Q leanring: $Q_{sample} = R(s,a,s') + \gamma \max_{a'} Q(s',a')$ Sarsa: $Q_{sample} = R(s,a,s') + \gamma Q(s',a')$
4. $Q(s,a) = (1-\alpha)Q(s,a) + \alpha Q_{sample}$
5. $s = s' \quad a = a'$

How to choose a from s

$$\varepsilon-\text{greedy:} \qquad \mathrm{P(a = random)} = \varepsilon \quad \mathrm{P}\left(\mathrm{a} = \underset{a}{\operatorname{argmax}}\, Q(s,a)\right) = 1-\varepsilon$$

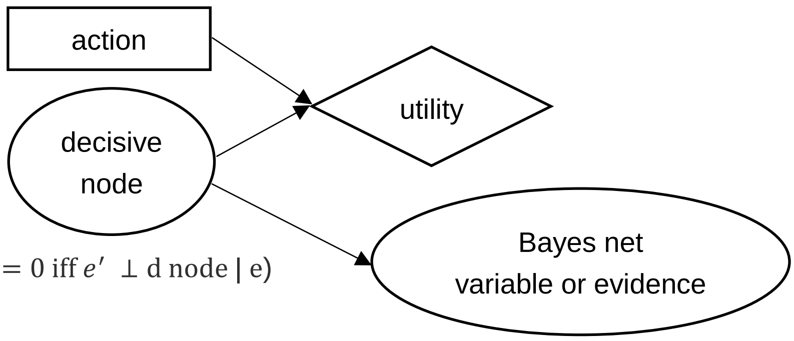
$$\text{Exploration function:} \quad \mathrm{F} = Q(s,a) + \frac{\mathrm{k}}{N(s'\ from\ s,a)\ \ (visit\ times\ sampled)} \quad \mathrm{a} = \underset{a}{\operatorname{argmax}}\, F$$

Decision Networks

$$\mathrm{EU}(\mathrm{a} \mid e) = \sum_{d\ values} P(value\ i \mid e) * U(a \mid value\ i)$$

$$\mathrm{MEU(e)} = \max_a EU(a \mid e)$$

$$\mathrm{VPI}(e' \mid e) = \textstyle\sum_{e'} P(e' \mid e) * MEU(e,e') - MEU(e) \quad (\geq 0, == 0 \text{ iff } e' \perp \text{d node} \mid \text{e})$$

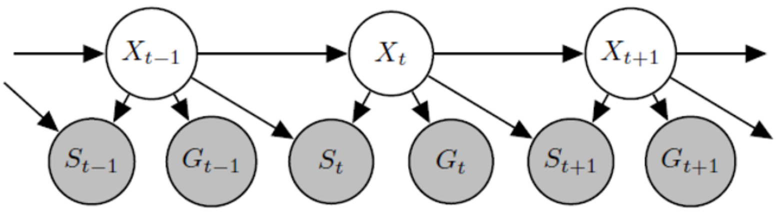


HMM

1. Passage of time $(\mathrm{P}(X_{t+1} \mid X_t))$: $\mathrm{P}(X_{t+1} \mid e_{1:t}) = \sum_{x_t} \mathrm{P}(X_{t+1} \mid x_t) * \mathrm{P}(x_t \mid e_{1:t})$
2. Observation $(\mathrm{P}(E_t \mid X_t))$: $\mathrm{P}(X_{t+1} \mid e_{1:t+1}) \propto \mathrm{P}(e_{t+1} \mid X_{t+1}) * \mathrm{P}(X_{t+1} \mid e_{1:t}) = \text{left} / P(e_{t+1} \mid e_{1:t})$

Particle Filtering: number of particles stands for p, loop

1. Passage of time: transfer particles with $\mathrm{P}(X_{t+1} \mid X_t)$
2. Observation: multiply $\mathrm{P}(e_{t+1} \mid X_{t+1})$ for each particle and get smaller ones
3. Resample: turn weights of particles to number of them



$$P(x_t|s_{1:t},g_{1:t}) = \underbrace{\frac{1}{P(s_t,g_t|s_{1:t-1},g_{1:t-1})}}_{\text{Normalize}} \sum_{x_{t-1}} \underbrace{P(s_t|x_{t-1},x_t)P(g_t|x_t)}_{\text{Observation}} \underbrace{P(x_t|x_{t-1})}_{\text{Elapse of time}} P(x_{t-1}|s_{1:t-1},g_{1:t-1}).$$

Machine Learning

Naive Bayes

Maximum likelihood estimation

$$\hat{y} = \underset{y}{\operatorname{argmax}} P(y) \prod_i P(x_i \mid y) \quad x_i: sample\ component$$

Laplace smoothing

$$\mathrm{P(x)} = \frac{c(x)+k}{N+k|x|} \quad \mathrm{P(x \mid y)} = \frac{c(x,y)+k}{c(y)+k|x|}$$

$$c(x): number\ of\ samples\ in\ catogory\ x \quad N: sample\ num \quad |x|: catagory\ num$$

Perceptron

Binary class: a weight vector w

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

$$\mathrm{if\ y} != y^*: \quad w = w + y^* * f \quad (f: sample)$$

Multiclass: a weight vector w_y for each class

prediction highest score wins: $y = \underset{y}{\operatorname{argmax}} w_y * f(x)$

$$\mathrm{if\ y} != y^*: \quad w_y = w_y - y^* * f \quad w_{y^*} = w_{y^*} + y^* * f$$

Logistic Regression

Binary class

$$p(y=1 \mid \mathbf{x}) = \frac{e^{\mathbf{w}^T\mathbf{x}+b}}{1+e^{\mathbf{w}^T\mathbf{x}+b}} \qquad p(y=0 \mid \mathbf{x}) = \frac{1}{1+e^{\mathbf{w}^T\mathbf{x}+b}}$$

$$l(\mathbf{w},b) = \sum_{i=1}^{n} \ln p(y_i = j \mid \mathbf{x}_i, \mathbf{w}, b) = \sum_{i=1}^{n}\left[y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) - \ln\left(1+e^{\mathbf{w}^T\mathbf{x}_i+b}\right)\right]$$

$$\begin{cases} \mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \lambda\Delta\mathbf{w} = \mathbf{w}^{(t)} - \lambda \left.\dfrac{\partial l(\mathbf{w},b)}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{(t)}, b=b^{(t)}} \\ b^{(t+1)} = b^{(t)} - \lambda\Delta b = b^{(t)} - \lambda \left.\dfrac{\partial l(\mathbf{w},b)}{\partial b}\right|_{\mathbf{w}=\mathbf{w}^{(t)}, b=b^{(t)}} \end{cases}$$

$$式中 \begin{cases} \dfrac{\partial l(\mathbf{w},b)}{\partial \mathbf{w}} = -\sum_{i=1}^{n}\left[\mathbf{x}_i y_i - \mathbf{x}_i p(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}, b)\right] \\ \dfrac{\partial l(\mathbf{w},b)}{\partial b} = -\sum_{i=1}^{n}\left[y_i - p(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}, b)\right] \end{cases}$$

Multiclass

$$\mathrm{P}(\mathrm{y} = y^i \mid x) = softmax(score_{y^i})$$

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

with: $$P(y^{(i)}|x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$