算法

Short Name	Full Name	Comments
BrFS(BFS)	Breadth-first-search	- Completeness - Optimality(if costs are uniform)
DFS	Deep-first-search	- Not completeness - Not optimality
ID	Iterative Depening	- Completeness - Optimality
GBFS	Greedy Best-First Search	
BFWS	Best-First Width Search	
h^*		- Optimal one (theoretical)
h^+		Safe, Admissible (drop deletion)
h^{add}		Safe, not admissible (based on h^+)
h^{max}		Safe, admissible (based on h^+)
$IW(k)$	- if $novetly(s) > k$, then prune	- try to solve problem first in $IW(1)$, if not solved, then $IW(2)$, ... - **novetly:** smallest subset of atoms (which is first showing up) size of the new state
MDP	$V(s) = \max_{a \in A(s)} \sum_{s' \in S} P_a(s' \vert s) \; [r(s,a,s') + \gamma \; V(s')]$ **Or** $Q(s,a) = \sum_{s' \in S} P_a(s' \vert s) \; [r(s,a,s') + \gamma \; V(s')]$ $V(s) = \max_{a \in A(s)} Q(s,a)$	**fully** observable, **probabilistic** state models - a state space S - initial state $s_0 \in S$ - a set $G \subseteq S$ of goal states - actions $A(s) \subseteq A$ applicable in each state $s \in S$ - transition probabilities $P_a(s' \vert s)$ for $s \in S$ and $a \in A(s)$ - action costs $c(a,s) > 0$ - **value iteration:**(update value via last iteration value) $V_{i+1}(s) := \max_{a \in A(s)} \sum_{s' \in S} P_a(s' \vert s) \; [r(s,a,s') + \gamma \; V_i(s')]$ - **policy iteration:**(update policy via existing policy)

		$V^{\pi}(s) = \sum_{s' \in S} P_a(s' \vert s)\ [r(s,a,s') + \gamma\ V^{\pi}(s')]$
MCTs		
UCT	$\pi(s) := \underset{a \in A(s)}{\operatorname{argmax}} Q(a,s) + 2C_p \sqrt{\frac{2 \ln N(s)}{N(s,a)}}$	
Q-Learning	$Q(s,a) \leftarrow \overbrace{Q(s,a)}^{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} [\overbrace{\underbrace{r}_{\text{immediate reward}} + \underbrace{\gamma \max_{a'} Q(s',a')}_{\text{discounted future reward}}}^{\text{recent value}} - \overbrace{Q(s,a)}^{\text{old value}}]$ off-policy optimistic unsafe or risky	
SARSA	$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$ safe	
n-step SARSA	$G_t = r1 + \gamma r2 + \gamma^2 r3 + \gamma^3 Q(s''',a''')$ $= r1 + \gamma[r2 + \gamma r3 + \gamma^2 Q(s''',a''')]$ $= r1 + \gamma[r2 + \gamma[r3 + \gamma Q(s''',a''')]]$ $Q(s,a) \leftarrow Q(s,a) + \alpha(G_t - Q(s,a))$	
Nash Equation		
Mix strategies: Indifferency	Column player: L, R Row player X T: 320, 40 | 40, 80 Row player 1-X B: 40, 80 | 80, 40 What is the mix strategy of Row Player? In order to make Column player indifference payoffs are equal for all strategies for Column player L E(L) = 40*X + 80*(1-X) R E(R) = 80*X + 40*(1-X) E(l) =E(R) X = 1/2	- think of A and B, the indifferency for A is that the probability X of selecting action a that makes B have the same reward.

Reinforcement Learning Cons

1. Unlike Monte-Carlo methods, which reach a reward and then backpropagate this reward, TD methods use bootstrapping (they estimate the future discounted reward using $Q(s, a)$), which means that for problems with spare rewards, it can take a long time to for rewards to propagate throughout a Q-function.
2. Both methods estimate a Q-function $Q(s, a)$, and the simplest way to model this is via a Q-table. However, this requires us to maintain a table of size $|A| \times |S|$, which is prohibitively large for any non-trivial problem.
3. Using a Q-table requires that we visit every reachable state many times and apply every action many times to get a good estimate of $Q(s, a)$. Thus, if we never visit a state s, we have no estimate of $Q(s, a)$, even if we have visited states that are very similar to s.
4. Rewards can be sparse, meaning that there are few state/actions that lead to non-zero rewards. This is problematic because initially, reinforcement learning algorithms behave entirely randomly and will struggle to find good rewards. Remember the Freeway demo from the previous lecture?