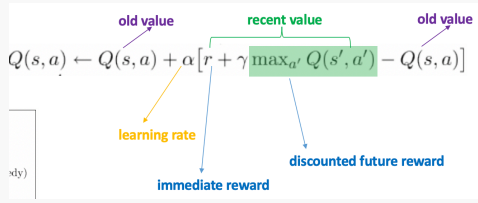
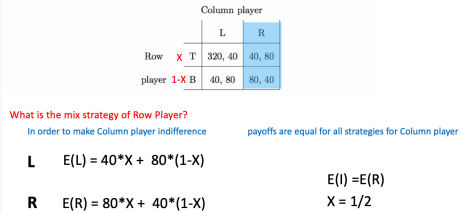


# 算法

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 Deepening	- 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 $IW(2)$ , ... - <b>novetly</b> : smallest subset of atoms (which is first showing up) size of the new state
MDP	$V(s) = \max_{a \in A(s)} \sum_{s' \in S} P_a(s' s) [r(s, a, s') + \gamma V(s')]$ <p>Or</p> $Q(s, a) = \sum_{s' \in S} P_a(s' s) [r(s, a, s') + \gamma V(s')]$ $V(s) = \max_{a \in A(s)} Q(s, a)$	<p><b>fully</b> observable, <b>probabilistic</b> state models</p> <ul style="list-style-type: none"> <li>■ a state space <math>S</math></li> <li>■ initial state <math>s_0 \in S</math></li> <li>■ a set <math>G \subseteq S</math> of goal states</li> <li>■ actions <math>A(s) \subseteq A</math> applicable in each state <math>s \in S</math></li> <li>■ transition probabilities <math>P_a(s' s)</math> for <math>s \in S</math> and <math>a \in A(s)</math></li> <li>■ action costs <math>c(a, s) &gt; 0</math></li> </ul> <p>- <b>value iteration</b>:(update value via last iteration value)</p> $V_{i+1}(s) := \max_{a \in A(s)} \sum_{s' \in S} P_a(s' s) [r(s, a, s') + \gamma V_i(s')]$ <p>- <b>policy iteration</b>:(update policy via existing policy)</p>

		$V^\pi(s) = \sum_{s' \in \mathcal{S}} P_a(s' s) [r(s, a, s') + \gamma V^\pi(s')]$
MCTs		
UCT	$\pi(s) := \operatorname{argmax}_{a \in A(s)} Q(a, s) + 2C_p \sqrt{\frac{2 \ln N(s)}{N(s, a)}}$	
Q-Learning	 <p>off-policy optimistic unsafe or risky</p>	
SARSA	$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', a') - Q(s, a)]$ <p>safe</p>	
n-step SARSA	$\begin{aligned} G_t &= r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 Q(s''', a''') \\ &= r_1 + \gamma [r_2 + \gamma r_3 + \gamma^2 Q(s''', a''')] \\ &= r_1 + \gamma [r_2 + \gamma [r_3 + \gamma Q(s''', a''')]] \end{aligned}$ $Q(s, a) \leftarrow Q(s, a) + \alpha (G_t - Q(s, a))$	
Nash Equation		
Mix strategies: Indifference	 <p>What is the mix strategy of Row Player? In order to make Column player indifference      payoffs are equal for all strategies for Column player</p> <p><b>L</b>    <math>E(L) = 40 \cdot X + 80 \cdot (1-X)</math></p> <p><b>R</b>    <math>E(R) = 80 \cdot X + 40 \cdot (1-X)</math></p> <p><math>E(L) = E(R)</math> <math>X = 1/2</math></p>	<p>- think of <math>A</math> and <math>B</math>, the indifference for <math>A</math> is that the probability <math>X</math> of selecting action <math>a</math> that makes <math>B</math> have the same reward.</p>

## 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 sparse 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?