Research on Average Reward Reinforcement Learning Algorithms

Dr. Yang Gao

National Laboratory for Novel Software Technology, Nanjing University

Nov. 5, 2006 MLA06@Nanjing, China
Outline

- **Case Study:** an Access-Control Queuing Task
- **Motivation and Background**
- **DP Methods for Average Rewards MDP**
- **RL Methods for Average Rewards MDP**
- **Our RL Method:** an Average Reward Reinforcement Learning Algorithm based on the Performance Potential of a Reference State
- **Future Works**
Case: an Access-Control Queuing Task

- \( N \) servers \((N=10)\)
- Customers with four different priorities
  - The customers with priority 1, 2 and 4 arrives the queue server with the a probability of 0.2
  - The customers with priority 8 arrives with a probability of 0.4
  - When access to a server, the system gets a reward of 1, 2, 4, 8

- When a customer arrives at the queuing, servers decide to accept it or reject it?
Motivation: why is not discounting

- Purpose of discounting
  - Such as economics, discounting can be used to represent “interest” earned on rewards, so that an action that generates an immediate reward will be preferred over one that generates the same reward some steps into the future.
  - In some sequence decision task, the goal can be transformed using the discounting method.
Background: Average Reward Markov Decision Processes

- MDP
  - $S, A, P_{xy}(a), r(x,a)$
  - Glossary
    - Communicate, Recurrent
    - Transient: non-recurrent state

Any finite MDP must have recurrent states, since not all states can be transient.

If a recurrent state $x$ communicates with another state $y$, then $y$ has to be recurrent also.
MDP: glossary

- **Ergodic**: recurrent class of states.
- **Irreducible**: all states form an ergodic class.
- **Period**: States in a given recurrence class all have the same period. / *aperiodic*

- **An ergodic or recurrent MDP**: The transition matrix corresponding to every policy has a single recurrent class.

- **Unichain**:

- **Multichain**: At least one policy whose transition matrix has two or more recurrent classes.
Example 1

- State A is transient under either policy (doing action $a_1$ or $a_2$ in state A).
- State B is recurrent.
- Such recurrent single-state classes are often called absorbing states.
Example 2

• If action \textbf{a1} is taken in state \textbf{A}, the recurrent class is formed by \textbf{A} and \textbf{B}, and is periodic with period 2.

• If action \textbf{a2} is taken in state \textbf{A}, the recurrent class is formed by \textbf{A} and \textbf{C}, and is periodic with period 2.

A1(2,1)  A1(0,1)
Gain Optimality

\[
\rho^\pi (x) = \frac{1}{N} \lim_{N \to \infty} E \left( \sum_{t=0}^{N-1} R_t^\pi (x) \right)
\]

**Gain-optimal:** policy \( \pi^* \) is one that maximizes the average reward over all states, that is \( \rho^\pi^* (x) \geq \rho^\pi (x) \) over all policies and states.
Gain Optimality

- Since states in the recurrent class will be visited forever, the expected average reward cannot differ across these states.
- Since the transient states will eventually never be reentered, they can at most accumulate a finite total expected reward before entering a recurrent state, which vanishes under the limit.

- For the example 1, the gain of the two policies are both -1;
- For the example 2, the gain of the two policies are both 1.
Bias Optimality

- Example 3
  - Both policies yield the same average reward.
  - Doing action \( a_1 \) is clearly preferable to doing \( a_2 \) in state \( A \).

- Bias Optimality
  - Define a value function \( V: S \rightarrow R \).

\[
V_\gamma^\pi(x) = \lim_{N \rightarrow \infty} E \left( \sum_{t=0}^{N-1} \gamma^t R_t^\pi(x) \right)
\]

But, in average MDP, \( \gamma = 1 \) ?
Bias Optimality

- **Average adjusted** sum of rewards

\[ V^\pi (x) = \lim_{N \to \infty} \mathbb{E} \left( \sum_{t=0}^{N-1} \left( R^\pi_t (x) - \rho^\pi \right) \right) \]

- **Bias value (relative value)**: represents the relative difference in total reward gained from starting in state \( x \) as opposed to some other state \( s \).

\[ V^\pi (x) - V^\pi (s) = \lim_{N \to \infty} \left\{ \mathbb{E} \left[ \sum_{t=0}^{N-1} R^\pi_t (x) \right] - \mathbb{E} \left[ \sum_{t=0}^{N-1} R^\pi_t (s) \right] \right\} \]
Bias Optimality

**Bias-optimal** policy $\pi^*$:

If it is gain-optimal, and it also maximizes bias values.
Related works in dynamic programming

- DP: DP Methods for Average Rewards MDP
  - The policy iteration algorithm [Howard 1960]
  - Blackwell 1962, Bias optimality
  - Veinott 1969, N-discounted-optimality
  - Puterman 1994, Important book
Theorem 1: For any MDP that is either unichain or communicating, there exists a value function $V^*$ and a scalar $\rho^*$ satisfying the equation

$$V^*(x) + \rho^* = \max_a \left[ r(x, a) + \sum_y P_{xy}(a)V^*(y) \right]$$

So the **greedy policy** achieves the optimal average reward.
Unichain Policy Iteration

- Policy iteration iterates over two phases: policy evaluation and policy improvement.

- Policy evaluation

\[ V^{\pi^k}(x) + \rho^{\pi^k} = r(s, \pi^k(x)) + \sum_y P_{xy}(\pi^k(x))V^{\pi^k}(y) \]

- Policy improvement

\[ action = \arg \max_a \left( r(s, a) + \sum_y P_{xy}V^{\pi^k}(y) \right) \]
Average Reward DP Algorithms

- Unichain Policy Iteration
  - Does it produce bias-optimal polices, or only gain-optimal policies?
  - 1. Set $V(A)=0$
    - $V(A)+1=2+V(B) \rightarrow V(B)=-1$
    - $V(A)+1=0+V(C) \rightarrow V(C)=1$
  - 2. Policy 1: select $a_2$ in $A$.
  - 3. No policy improvement.
  - 4. Policy 1 is only gain-optimal and not bias-optimal.
Average Reward DP Algorithms

- Value Iteration
  - The difficulty with policy iteration is that it requires solving \(|S|\) equations at every iteration, which is computationally intractable when \(|S|\) is large.
  - Define \(T(V)(x)\), \(T\) is a monotone mapping.

\[
T(V)(x) = \max_a \left[ r(x, a) + \sum_y P_{xy}(a)V(y) \right]
\]
Average Reward DP Algorithms

- **Value Iteration**
  - Note:
    1. The value iteration algorithm does not explicitly compute the average reward, but this can be estimated as $V^{n+1}(x) - V^n(x)$ for large $n$.
    2. **Disadvantage**: the values $V(x)$ can grow very large, causing numerical instability.
  - **White’s relative value iteration algorithm**:
    \[
    V^{k+1}(x) = T\left(V^k\right)(x) - T\left(V^k\right)(s)
    \]

Like policy iteration, value iteration cannot discriminate between the bias-optimal and gain-optimal policies in some MDP.
Average Reward DP Algorithms

- Asynchronous Value Iteration
  - Policy iteration & value iteration are both *synchronous*.
  - RL methods are *asynchronous*.

- Convergence
  - The key is to ensure that the underlying mapping remains *monotonic* or a *contraction* with respect to the maximum norm.
Average Reward DP Algorithms

- Asynchronous Value Iteration
  - Jalali & Ferguson, 1990
  
  \[
  V^{t+1}(x) = T(V^t)(x) - \rho^t \quad \forall x \neq s \\
  V^t(s) = 0
  \]

  - $\rho^t$ is the estimate of the average reward at time $t$, is independently estimated without using the relative values.

  The mapping is *monotonic*.

There are several ways of independently estimating average reward.
An Asynchronous Adaptive Control Method

- The above DP algorithms require
  - Complete knowledge of the state transition matrices
  - The expected payoffs for each action and state

- Jalali & Ferguson, 1990, A algorithm and B algorithm

1) \( \text{action} = \arg \max_a \left( r(s,a) + \sum_y P_{xy}(a)V^t(y) \right) \)

2) \( V^{t+1}(x) = T(V^t)(x) - \rho^t \)

3) \( \rho^{t+1} = \frac{K^{t+1}}{t+1} \), where \( K^{t+1} = K^t + r(x,a) \)

4) Update the probability transition matrix entry \( P_{xy}^t(a) \)
   using a maximum-likelihood estimator
An Asynchronous Adaptive Control Method

- Some discussion about B algorithm
  - 1. In order to guarantee convergence, the MDP is \textit{identifiable} by an MLE-estimator.
  - 2. \textbf{One modification}: estimate the expected rewards \( r(x,a) \) from sample reward.
  - 3. \textbf{Another modification}: Take random actions in step 1 (semi-uniform exploration)
## Summary of average reward methods

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>GAIN OPTIMALITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unichain Policy Iteration</td>
<td>MDP is unichain</td>
</tr>
<tr>
<td>(Relative) Value Iteration</td>
<td>MDP is communicating</td>
</tr>
<tr>
<td>Asynchronous Relative Value Iteration</td>
<td>Dose not converge</td>
</tr>
<tr>
<td>Asynchronous Value Iteration with online Gain Estimation</td>
<td>A state $s$ is reachable under every policy</td>
</tr>
<tr>
<td>Asynchronous Adaptive Control with online Gain Estimation</td>
<td>MDP is ergodic and MLE-Identifiable</td>
</tr>
</tbody>
</table>
The study of average reward RL is currently at an early stage.

- **R-learning**, the first average-reward RL method [Schwartz, ML1993]
  - Outperform discounted methods, such as Q-learning.
- **Modified R-learning**, [Singh, AAAI1994]
- **A Model-based Algorithm for Bias-optimal**, [Mahadevan, ML1996]
- **H-Learning**, [Tadepalli, AI1998]
- **SMART**, [Das, ManagementScience1999]
- **Q-P-Learning**, [Gosavi, ML2004]
- **G-Learning**, [Gao, SMC2006]
R-learning: A Model-Free Average Reward RL Method [Schwartz, ML1993]

\[ R^\pi (x, a) = r(x, a) - \rho^\pi + \sum_{y} P_{xy}(a)V^\pi(y), \quad V^\pi(y) = \max_a R^\pi(y, a) \]

1) \[ R_{t+1}(x, a) \leftarrow (1 - \beta) R_t(x, a) + \beta (r_{imm}(x, y) - \rho_t + \max_a R_t(y, a)) \]

2) \[ \rho_{t+1} \leftarrow (1 - \alpha) \rho_t + \alpha \left[ r_{imm}(x, a) + \max_a R_t(y, a) - \max_a R_t(x, a) \right] \]

Important!
Some Variations on the basic R-learning

Singh 1994 [Singh, AAAI1994]

- Adjust Bellman equation
- Estimating average reward as the sample mean of the actual rewards
- Updating the average reward on every step

1. Like all the preceding algorithms, it is unable to differentiate the bias-optimal policy from the gain-optimal policy.

Some Variations on the basic R-learning

- Singh 1994[Singh, AAAI1994]

\[ B_\pi (V)(x_t) = R(x_t, \pi(x_t)) + V_t(x_{t+1}) \]

- Algorithm 1

\[ V_{t+1}(x_t) = (1 - \alpha_t(x_t))V_t(x_t) + \alpha_t(x_t)(B_\pi(V_t)(x_t) - \rho_t) \]

where \( \rho_0 = 0 \), and

\[ \rho_{t+1} = (1 - \beta_t)\rho_t + \beta_t[B_\pi(V_t)(x_t) - V_t(x_t)] \]

- Algorithm 2

\[ \rho_{t+1} = \frac{(t \ast \rho_t) + R(x_t, \pi(x_t))}{t + 1} \]
Some Variations on the basic R-learning

• Algorithm 3
  ▪ The difference between Algorithm 3 and R-learning is that in R-learning the estimated average payoff is updated only when the greedy action is executed.
  ▪ In Algorithm 3, the average payoff is updated with every action.

Algorithm 3 could be more efficient than R-learning since R-learning seems to waste information.

• Algorithm 4
  ▪ Estimates average payoff like algorithm 2
Model-based Average Reward Reinforcement Learning

Tadepalli & Ok: H-learning [Tadepalli, AI1998]

1. Take an exploratory action or a greedy action in the current state \( i \). Let \( a \) be the action taken, \( k \) be the resulting state, and \( r_{imm} \) be the immediate reward received.

2. \( N(i, a) \leftarrow N(i, a) + 1; N(i, a, k) \leftarrow N(i, a, k) + 1 \)

3. \( p_{i,k}(a) \leftarrow N(i, a, k)/N(i, a) \)

4. \( r_i(a) \leftarrow r_i(a) + (r_{imm} - r_i(a))/N(i, a) \)

5. \( GreedyActions(i) \leftarrow \) All actions \( u \in U(i) \) that maximize \( \{r_i(u) + \sum_{j=1}^{n} p_{i,j}(u)h(j)\} \)

6. If \( a \in GreedyActions(i) \), then
   
   (a) \( \rho \leftarrow (1 - \alpha)\rho + \alpha(r_i(a) - h(i) + h(k)) \)
   
   (b) \( \alpha \leftarrow \alpha/\alpha+1 \)

7. \( h(i) \leftarrow \max_{u \in U(i)} \{r_i(u) + \sum_{j=1}^{n} p_{i,j}(u)h(j)\} - \rho \)

8. \( i \leftarrow k \)
A Model-based Algorithm for Bias-optimal

- Mahadevan, ML1996

1. Initialize time \( n = 0 \), bias values \( V(x) = 0 \), and bias offset values \( W(x) = 0 \). Let the initial state be \( i \). Initialize \( N(i,a) = 0 \), the number of times action \( a \) has been tried in state \( i \), and \( T(i,a,k) = 0 \), the number of times \( a \) has resulted in moving from state \( i \) to \( k \). Initialize the expected rewards \( r(i,a) = 0 \). Let \( s \) be some reference state, preferably one that is recurrent under all policies.

2. Let \( H(i,a) = r(i,a) + \sum_j P_{ij}(a)V(j) \), \( \forall a \in A(i) \).

3. Let \( h(i) = \{ a \in A(i) | a \text{ maximizes } H(i,a) \} \). Let \( A(i,\epsilon_n) \) be the set of actions that are within \( \epsilon_n \) of the maximum \( H(i,a) \) value.

4. Let \( w(i,\epsilon_n) = \{ a \in A(i,\epsilon_n) | a \text{ maximizes } \sum_j P_{ij}(a)W(j) \} \). With probability \( 1 - p_{\text{exp}} \), select action \( A \) to be some \( a_o \in w(i,\epsilon_n) \). Otherwise, let action \( A \) be any random action \( a_r \in A(i) \).
A Model-based Algorithm for Bias-optimal

6. Carry out action \( A \), and let the next state be \( k \), and the immediate reward received be \( r(i, a) \).

7. \( N(i, A) \leftarrow N(i, A) + 1 \).

8. \( T(i, A, k) \leftarrow T(i, A, k) + 1 \).

9. \( P_{ik}(A) \leftarrow \frac{T(i,k,A)}{N(i,A)} \).

10. \( r(i, A) \leftarrow r(i, A)[1 - \frac{1}{N(i,A)}] + \frac{1}{N(i,A)} r(i, a) \).

11. \( V(i) \leftarrow \max_{a \in A(i)}(H(i,a)) - \max_{a \in A(s)}(H(s,a)) \).

12. \( W(i) \leftarrow \max_{a \in A(i, c_a)} \left( \sum_j P_{ij}(a) W(j) - V(i) \right) - W(s) \), where
    \[ W(s) = \max_{a \in A(s, c_a)} \left( \sum_j P_{sj}(a) W(j) - V(s) \right) \).

13. If \( n < MAX\_STEPS \), set \( n \leftarrow n + 1 \), and \( i \leftarrow k \) and go to step 5.

14. Output \( \pi(i) \in w(i, c_n) \).
A Reinforcement Learning Algorithm based on Policy Iteration for Average Reward

- Gosavi, ML2004: Q-P-Learning

**ALGORITHM Q-P-Learning:**

Step 1: Initialize the vector \( P(i, a) \) for all states \( i \in S \) and all \( a \in U(i) \) (the set of permissible actions in state \( i \)) to random values. Set \( E = 1 \) (where \( E \) denotes the number of phases) and initialize \( E_{\text{max}}, m_{\text{max}}, h_{\text{max}}, \) and \( T_{\text{max}} \) to large numbers.

Step 2: Set \( Q(i, a) = 0 \) for all \( i \in S \) and all \( a \in U(i) \). Simulate the system for a time interval of \( T_{\text{max}} \), using the action given by \( \arg\max_{b \in U(i)} P(i, b) \) in every \( i \in S \). At the end of the simulation, divide the total of immediate rewards by \( T_{\text{max}} \) to obtain an estimate of the average reward \( \rho \). Use averaging with \( h_{\text{max}} \) replications to obtain a good estimate of \( \rho \). Set \( m \), the iteration number within a phase, to 0.

Step 3: (Policy evaluation) Start fresh simulation. Let the system state be \( i \in S \).

Step 3a: Simulate action \( u \in U(i) \) with probability \( 1/|U(i)| \).

Step 3b: Let the next decision-making state encountered in the simulator be \( j \). Also, let \( t(i, u, j) \) be the transition time (from state \( i \) to state \( j \)) and let \( r(i, u, j) \) be the immediate reward.

Step 3c: Calculate \( \beta \) using \( m \) (see Comment 1 below). Then change \( Q(i, u) \) using:

\[
Q(i, u) \leftarrow (1 - \beta)Q(i, u) + \beta[r(i, u, j) - \rho t(i, u, j) + Q(j, \arg\max_{v \in U(j)} P(j, v))].
\]

Step 3d: Increment \( m \) by 1. If \( m < m_{\text{max}} \) set current state \( i \) to new state \( j \) and then go to Step 3a; else go to Step 4.

Step 4: (Q to P conversion - policy improvement) Set \( P(i, a) \leftarrow Q(i, a) \) for all \( i \) and \( a \in U(i) \). Set \( E \leftarrow E + 1 \). If \( E \) equals \( E_{\text{max}} \) terminate learning; else go back to Step 2.
Average Reward Reinforcement Learning
Algorithm based on a Reference State

- Our RL algorithm

\[ \begin{align*}
  b_1 &= 1 - \rho + b_2 \\
  b_2 &= 2 - \rho + b_1
\end{align*} \]

\[ \begin{align*}
  g_1 &= 1 - g_1 + g_2 \\
  g_2 &= 2 - g_1 + g_1
\end{align*} \]

\[ b_1 - b_2 = -\frac{1}{2} \quad \rightarrow \quad g_1 = \frac{3}{2}, \quad g_2 = 2 \]
Off-policy G-learning

PSEUDO-CODE OF THE OFF-POLICY G-LEARNING ALGORITHM.

- Choose an reference state $s_0$, initialize $G(s, a)$, the learning rate $\alpha$ and $\epsilon$.
- Repeat forever
  - $s \leftarrow$ current state.
  - Choose an action for $s$ with a behavior policy ($\epsilon$-greedy), receive a cost $r(s, a)$, and next state $s'$.
  - $G(s, a) \leftarrow G(s, a) + \alpha[r(s, a) - g(s_0) + \max_{a'} G(s', a') - G(s, a)]$.
  - $\alpha \leftarrow \frac{\alpha}{\alpha + 1}$.
  - If $s = s_0$, $g(s_0) = \max_a G(s_0, a)$.
On-policy G-learning

PSEUDO-CODE OF THE ON-POLICY G-LEARNING ALGORITHM.

- Choose an reference state $s_0$, initialize $G(s, a)$, the learning rate $\alpha$ and $\epsilon$.
- Repeat forever
  - Choose an action for $s$ with a behavior policy ($\epsilon$-greedy), receive a cost $r(s, a)$, and next state $s'$.
  - Choose an action $a'$ for $s'$ with a behavior policy ($\epsilon$-greedy).
  - $G(s, a) \leftarrow G(s, a) + \alpha [r - g(s_0) + G(s', a') - G(s, a)]$.
  - $\alpha \leftarrow \frac{\alpha}{\alpha + 1}$.
  - $s \leftarrow s', a \leftarrow a'$.
  - If $s = s_0$, $g(s_0) = \max_a (G(s_0, a))$. 

Dec. 5, 2006
MLA06@Nanjing
Case Study: The learnt policy using G-learning

- P1: (2,1)→1 Reject
- P2: (4,1)→0 Accept
- P3: (8,4)→1 Reject
- P4: (9,4)→0 Accept
Comparison (1)

- Comparison of average rewards between the learnt policy and P1, P2, P3 & P4

<table>
<thead>
<tr>
<th></th>
<th>The learnt policy</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trial 1</td>
<td>3.38775</td>
<td>3.38156</td>
<td>3.37336</td>
<td>3.34708</td>
<td>3.32774</td>
</tr>
<tr>
<td>Trial 2</td>
<td>3.3738</td>
<td>3.3834</td>
<td>3.3733</td>
<td>3.35493</td>
<td>3.32778</td>
</tr>
<tr>
<td>Trial 3</td>
<td>3.38092</td>
<td>3.38104</td>
<td>3.37973</td>
<td>3.35146</td>
<td>3.33767</td>
</tr>
<tr>
<td>Trial 4</td>
<td>3.40323</td>
<td>3.38127</td>
<td>3.38875</td>
<td>3.36622</td>
<td>3.35285</td>
</tr>
<tr>
<td>Trial 5</td>
<td>3.38612</td>
<td>3.38127</td>
<td>3.37837</td>
<td>3.3541</td>
<td>3.34253</td>
</tr>
<tr>
<td>Trial 6</td>
<td>3.38797</td>
<td>3.3855</td>
<td>3.39189</td>
<td>3.34519</td>
<td>3.34411</td>
</tr>
<tr>
<td>Trial 7</td>
<td>3.38354</td>
<td>3.39242</td>
<td>3.37124</td>
<td>3.36909</td>
<td>3.33946</td>
</tr>
<tr>
<td>Trial 8</td>
<td>3.37812</td>
<td>3.39584</td>
<td>3.37091</td>
<td>3.35436</td>
<td>3.33957</td>
</tr>
<tr>
<td>Trial 9</td>
<td>3.38309</td>
<td>3.39623</td>
<td>3.38047</td>
<td>3.35065</td>
<td>3.34287</td>
</tr>
<tr>
<td>Trial 10</td>
<td>3.38626</td>
<td>3.37011</td>
<td>3.39015</td>
<td>3.34853</td>
<td>3.34972</td>
</tr>
<tr>
<td>MEAN</td>
<td>3.38508</td>
<td>3.384864</td>
<td>3.379817</td>
<td>3.354161</td>
<td>3.34043</td>
</tr>
<tr>
<td>SD</td>
<td>±0.0074</td>
<td>±0.0076</td>
<td>±0.0076</td>
<td>±0.0074</td>
<td>±0.0077</td>
</tr>
</tbody>
</table>
Comparing the on-line learning performance between G-learning and R-learning

G-learning has the optimal learning performance than R-learning
Comparison (3)

- Comparing the learning performance between different reference states

Choose the frequent visited states as reference state.
Comparison (4)

- Comparing the g-value of different reference states
## Summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Model-free</th>
<th>On-policy</th>
<th>Value Iteration</th>
<th>Directly Measure $\rho$</th>
<th>When Update $\rho$</th>
<th>Bias-optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-learning</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>Adaptive</td>
<td>Greedy action</td>
<td>X</td>
</tr>
<tr>
<td>Variations on R-learning</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Adaptive or directly</td>
<td>Every action</td>
<td>X</td>
</tr>
<tr>
<td>H-learning</td>
<td>X</td>
<td>N/A</td>
<td>✓</td>
<td>Adaptive</td>
<td>Greedy action</td>
<td>X</td>
</tr>
<tr>
<td>A Model-based Algorithm for Bias-optimal</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>N/A</td>
<td>N/A</td>
<td>✓</td>
</tr>
<tr>
<td>Q-P-learning</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>N/A</td>
<td>N/A</td>
<td>?</td>
</tr>
<tr>
<td>G-learning</td>
<td>✓</td>
<td>Both</td>
<td>✓</td>
<td>Reference state</td>
<td>Visit reference state</td>
<td>?</td>
</tr>
</tbody>
</table>
Direction for Future Research

- Bias-optimal RL Algorithms
- Value Function Approximation in Average Reward MDP
- Multi-Chain Average Reward Algorithms
- Modular Average Reward Methods
- Average Reward Methods in SMDP
- Difference between Every algorithm
Future works: Exploration Strategies

- Undirected
  - Undirected exploration methods do not use the results of learning to guide exploration; they merely select a random action some of the time.
  - Semi-Uniform Exploration
  - Boltzmann Exploration

- Directed
  - Use the results of learning to decide where to concentrate the exploration efforts.
  - Recency-based Exploration
  - Uncertainty estimation Exploration
References

References


References

  Solving semi-Markov decision problems using average reward reinforcement

  Reinforcement Learning Algorithm Based on the Performance Potential of a
Thanks!
Any Questions?