中山大学数据科学与计算机学院

计算机科学与技术专业-人工智能

本科生实验报告

（2018-2019 学年秋季学期）

课程名称：**Artificial Intelligence**

教学班级	计科 2 班	专业（方向）	计算机科学与技术
学号	16337341	姓名	朱志儒

实验题目

期末项目

实验内容

· 算法原理

1. Epsilon-greedy 算法

Epsilon-greedy 算法是在“exploit”（利用）和“explore”（探索）之间权衡以尽可能实现收益的最大化。假设当前有 n 台老虎机，每台老虎机吐钱的概率不一样，也不清楚每台老虎机吐钱的概率分布，如何实现收益最大化呢？通常有两种方案，第一种就是在已经探索过的老虎机中找到吐钱最多的那台老虎机，然后坚持摇它，这就是“exploit”（利用）；第二种就是尝试不断探索新的老虎机，在探索过程中可能会找到更好的老虎机，也可能找到吐钱很

少的老虎机，这就是“explore”（探索）。Epsilon-greedy 算法是每次以 epsilon 概率进行“explore”（探索），也就是说在所有的老虎机中选择一个进行探索；每次以 1 – epsilon 概率进行“exploit”（利用），也就是在已经探索过的老虎机中选择收益最大的那个。

Epsilon 设置的越高，收敛到最佳收益的速度越快，因为随机搜索的概率越大，越能更快地探索到最好的老虎机。当 $0.1 < \text{Epsilon} < 0.5$ 时，Epsilon 设置的越低，最终的平均收益越高，因为每次以较大概率随机探索，将不能有效利用收益最大的老虎机。

2. Q-learning

1） 初始化状态 S_1 和表 Q。

2） 第一次循环，根据 Epsilon-greedy 算法在状态 S_1 中选择动作，选择并执行动作 A_1 后得到状态 S_2 和 S_2 的即时回报 R_2，然后更新表 Q，即

$$Q(S_1, A_1) = Q(S_1, A_1) + a\big(R_2 + \lambda \max Q(S_2, a_2) - Q(S_1, A_1)\big)$$

其中$\max Q(S_2, a_2)$表示在状态S_2下的动作中找到具有最大 Q 值的动作所对应的 Q 值。

3） 第二次循环，根据 Epsilon-greedy 算法在状态 S_2 中选择动作，选择并执行动作 A_2 后得到状态 S_3 和 S_3 的即时回报 R_3，然后更新表 Q，即

$$Q(S_2, A_2) = Q(S_2, A_1) + a\big(R_3 + \lambda \max Q(S_3, a_3) - Q(S_2, A_2)\big)$$

其中$\max Q(S_3, a_3)$表示在状态S_3下的动作中找到具有最大 Q 值的动作所对应的 Q 值。

4） 不断循环直至 S 到达最终状态。

3. Sarsa

1） 初始化状态 S_1 和表 Q。

2） 第一次循环，根据 Epsilon-greedy 算法在状态 S_1 中选择动作，选择并执行动作 A_1 后得到状态 S_2 和 S_2 的即时回报 R_2，再根据 Epsilon-greedy 算法在状态 S_2 中选择动作 A_2，然后更新表 Q，即

$$Q(S_1, A_1) = Q(S_1, A_1) + a\big(R_2 + \lambda Q(S_2, A_2) - Q(S_1, A_1)\big)$$

3） 第二次循环，根据 Epsilon-greedy 算法在状态 S_2 中选择动作，选择并执行动作 A_2 后得

到状态 S_3 和 S_3 的即时回报 R_3，再根据 Epsilon-greedy 算法在状态 S_3 中选择动作 A_3，然后更新表 Q，即

$$Q(S_2, A_2) = Q(S_2, A_2) + a\big(R_3 + \lambda Q(S_3, A_3) - Q(S_2, A_2)\big)$$

4） 不断循环直至 S 到达最终状态。

4. 遗传算法（队友实现）

队友使用的是遗传算法，我并未参与实现，所以在此只介绍相关原理。

1) 初始化

对于某个问题，随机生成该问题的一些解，每个解是一个向量代表一条染色体，一个向量中的不同元素代表一条染色体中的不同基因。根据具体的应用场景设计相应的适应度函数，每条染色体可通过该适应度函数计算出其自身的适应度。

2) 复制

模仿自然界基因复制和染色体复制，对问题解的复制就是复制其向量，以进入下一次迭代，过程如下：

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix}_{下一次} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}_{上一次}$$

3) 交叉

模仿自然界的一对染色体的等位基因相互交换，在两个解向量之间，随机选择几对相同位置的元素相互交换，再进行归一化后就可以进入下一次迭代，过程如下：

$$\begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} \times \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix} \rightarrow \begin{bmatrix} a_1 \\ b_2 \\ c_1 \end{bmatrix}$$

4) 突变

模仿自然界的基因突变，在每次迭代中，随机选择一个解向量，在该解向量上随机选择一个位置的元素，将其值改变为一个随机数，再进行归一化后得到一个新的解向量，然后进

入下一次迭代，过程如下：

随机选择一个解向量：

$$\mathrm{v} = [a, b, c, d]$$

在该向量中随机选择一个元素，将其改为一个随机数，再进行归一化：

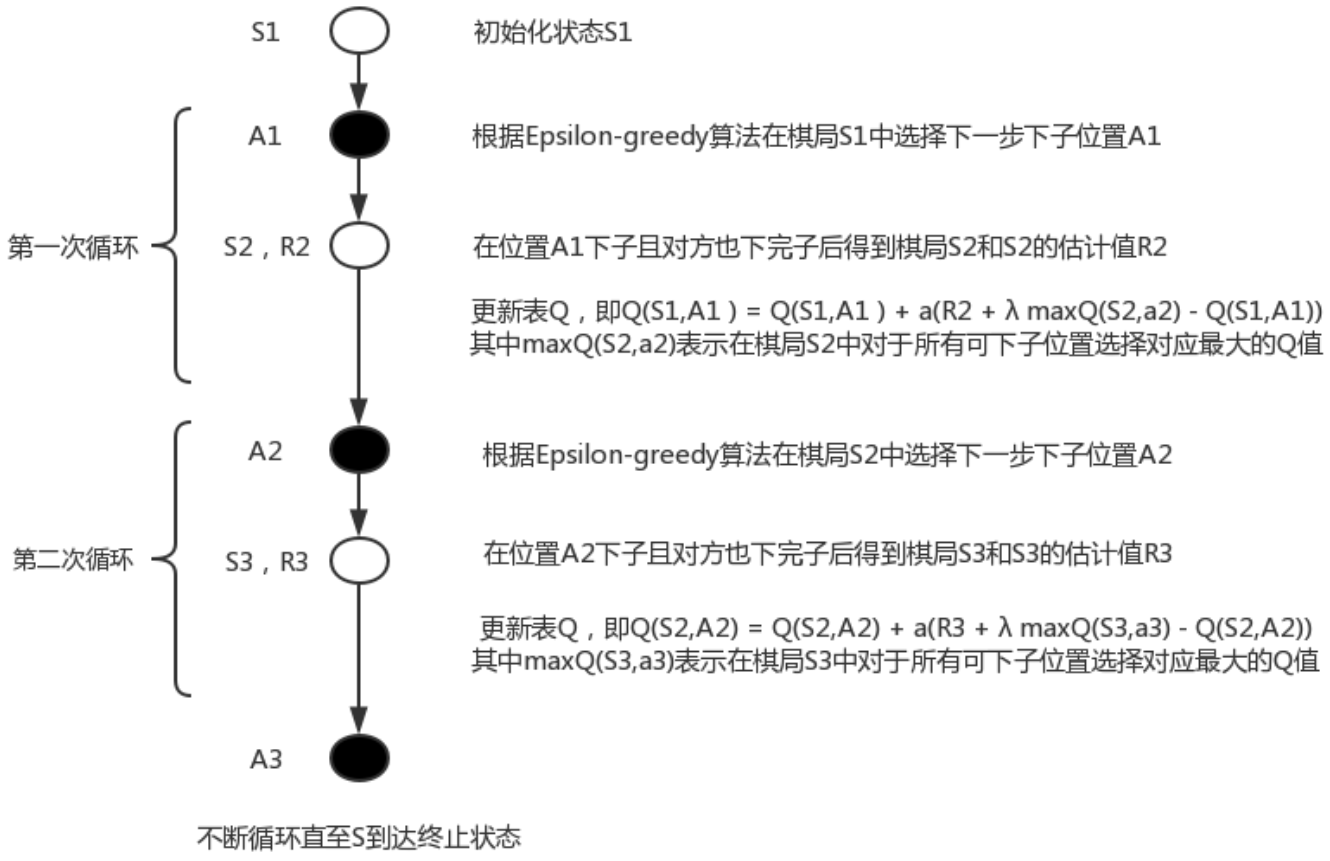
$$\mathrm{v}_{\mathrm{new}} = [a, b, c_{new}, d]$$

5) 结束

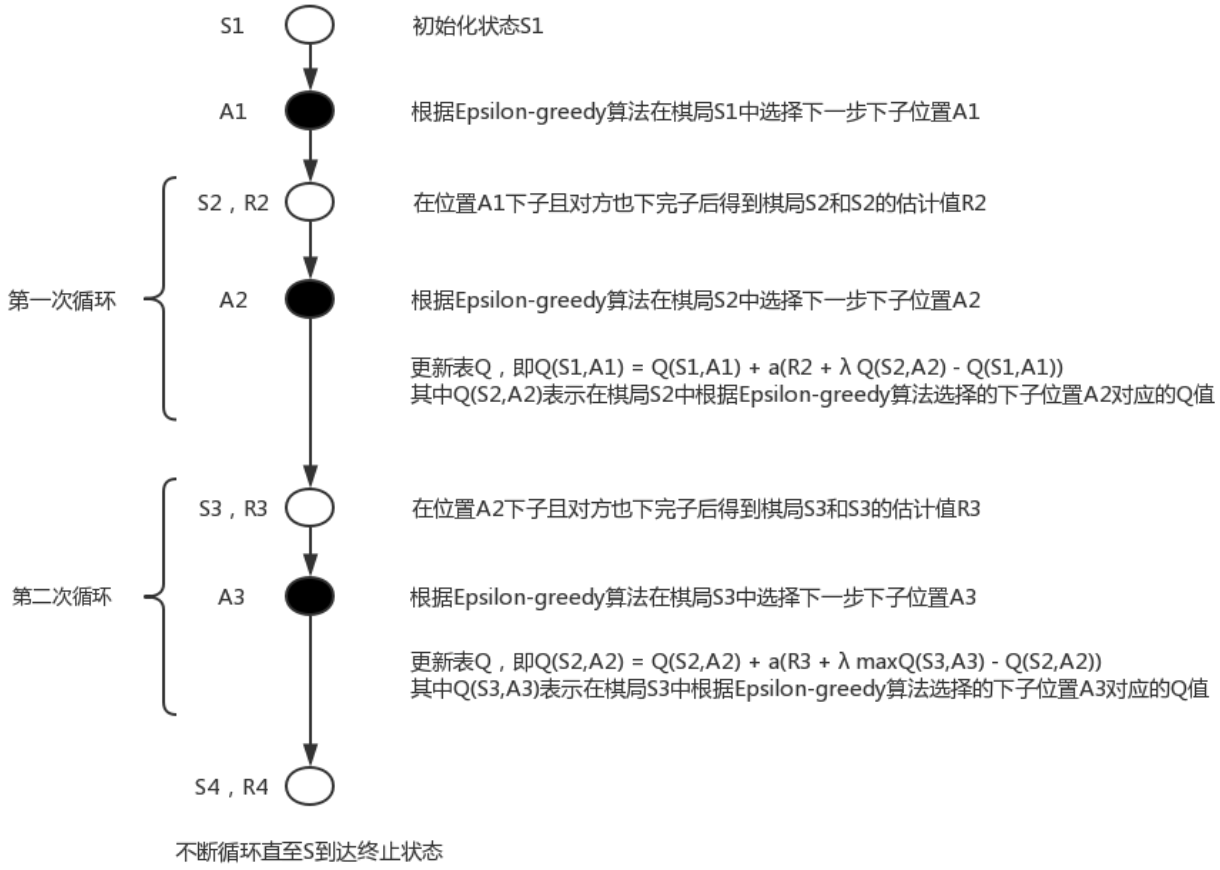
当解向量收敛时，迭代就可以结束，当然也可以直接限制迭代次数。

· 流程图

1、Q-learning



2、Sarsa



· 关键代码

1）Q-learning

Q-learning 在黑白棋中的应用中，每个不同的状态 S 表示每个不同的棋局，动作 a 表示下一步下子的位置，动作集 A 表示可下子位置的集合，状态 S 的即时回报 R 表示 S 所对应棋局的估计值。Q-learning 在黑白棋中的步骤为：

a. 初始化棋局 S_1 和表 Q。

b. 进入循环，根据 Epsilon-greedy 算法在棋局 S_1 中选择下一步下子位置 A_1，在位置 A_1 下子且对方也下完子后得到棋局 S_2 和 S_2 的估计值 R_2，然后更新表 Q，即

$$Q(S_1, A_1) = Q(S_1, A_1) + a\big(R_2 + \lambda \max Q(S_2, a_2) - Q(S_1, A_1)\big)$$

其中$\max Q(S_2, a_2)$表示在棋局S_2中对于所有可下子位置选择对应最大的 Q 值。

c. 不断循环直至游戏结束。

```
void Q_learning(char board[8][8], char color, void(*function)(char board[8][8], char)) {
    char opp = color == '@' ? 'O' : '@';
    double alpha = 0.8, gama = 0.5;
    state pre = state(board);
    default_random_engine e;
    uniform_real_distribution<double> u(0.0, 1.0);
    double epsilon = u(e);
    auto next_step = show_places(board, color);
    // 根据 Epsilon-greedy 算法选择动作
    if (1 - epsilon < 0.5) {
        // 以 1 - epsilon 概率进行“exploit”（利用），在已经探索过的动作中选择收益最大的
        double best_value = -10000000;
        pre.action = next_step[0];
        for (int i = 0; i < next_step.size(); ++i) {
            auto index = state(pre.board, next_step[i]);
            if (Q.find(index) == Q.end())
                continue;
            else if (Q[index] > best_value) {
                best_value = Q[index];
                pre.action = next_step[i];
            }}}
    else
        // 以 epsilon 概率进行“explore”（探索），在所有的动作种随机选择一个
        pre.action = next_step[rand() % next_step.size()];
    //进入循环
    while (!is_over(board)) {
        // 执行选好的动作
        move(board, pre.action, color);
        if (!show_places(board, opp).empty() && !show_places(board, opp).empty())
            function(board, opp);
        while (!is_over(board) && show_places(board, color).empty())
            function(board, opp);
        // 得到下一个状态
        state S = state(board);
        // 得到下一个状态的即时回报
        double reward = evaluate(board, color);
        next_step = show_places(board, color);
        // 在下一个状态下找到具有最大 Q 值的动作
        if (!next_step.empty()) {
            double best_value = -10000000;
            S.action = next_step[0];
```

```
            for (int i = 0; i < next_step.size(); ++i) {
                auto index = state(board, next_step[i]);
                if (Q.find(index) == Q.end())
                    continue;
                else if (Q[index] > best_value) {
                    best_value = Q[index];
                    S.action = next_step[i];
                }}}
        double new_q = 0, old_q = 0;
        if (Q.find(pre) != Q.end())
            old_q = Q[pre];
        if (Q.find(S) != Q.end())
            new_q = Q[S];
        // 更新表 Q
        Q[pre] = old_q + alpha * (reward + gama * new_q - old_q);
        // 以 epsilon 概率进行“explore”（探索）
        epsilon = u(e);
        if (!next_step.empty() && epsilon < 0.5)
            S.action = next_step[rand() % next_step.size()];
        pre = S;}}
```

2）Sarsa

Sarsa 在黑白棋的应用中，状态 S、动作 a、动作集 A、即时回报 R 所表示的意义与 Q-learning 在黑白棋的应用中的相同，Sarsa 在在黑白棋中的步骤为：

a. 初始化棋局 S_1 和表 Q。

b. 进入循环，根据 Epsilon-greedy 算法在棋局 S_1 中选择下一步下子位置 A_1，在位置 A_1 下子且对方也下完子后得到棋局 S_2 和 S_2 的估计值 R_2，再根据 Epsilon-greedy 算法在棋局 S_2 中选择下一步下子位置 A_2，然后更新表 Q，即

$$Q(S_1, A_1) = Q(S_1, A_1) + a\big(R_2 + \lambda Q(S_2, A_2) - Q(S_1, A_1)\big)$$

c. 不断循环直至游戏结束。

```
void Sarsa(char board[8][8], char color, void (* function)(char board[8][8],
 char)) {
    char opp = color == '@' ? 'O' : '@';
    double alpha = 0.8, gama = 0.5;
    state pre = state(board);
```

```
5.      default_random_engine e;
6.      uniform_real_distribution<double> u(0.0, 1.0);
7.      double epsilon = u(e);
8.      auto next_step = show_places(board, color);
9.      // 根据 Epsilon-greedy 算法选择动作
10.     if (1 - epsilon < 0.5) {
11.         // 以 1 - epsilon 概率进行“exploit”（利用），在已经探索过的动作中选择收益
    最大的
12.         double best_value = -10000000;
13.         pre.action = next_step[0];
14.         for (int i = 0; i < next_step.size(); ++i) {
15.             auto index = state(pre.board, next_step[i]);
16.             if (Sarsa_Q.find(index) == Sarsa_Q.end())
17.                 continue;
18.             else if (Sarsa_Q[index] > best_value) {
19.                 best_value = Sarsa_Q[index];
20.                 pre.action = next_step[i];
21.             }}}
22.     else
23.         // 以 epsilon 概率进行“explore”（探索），在所有的动作种随机选择一个
24.         pre.action = next_step[rand() % next_step.size()];
25.     // 进入循环
26.     while (!is_over(board)) {
27.         // 执行已选好的动作
28.         move(board, pre.action, color);
29.         if (!show_places(board, opp).empty() && !show_places(board, opp).emp
    ty())
30.             function(board, opp);
31.         while (!is_over(board) && show_places(board, color).empty())
32.             function(board, opp);
33.         // 得到下一个状态
34.         state S = state(board);
35.         // 得到下一个状态的即时回报
36.         double reward = evaluate(board, color);
37.         epsilon = u(e);
38.         next_step = show_places(board, color);
39.         // 根据 Epsilon-greedy 算法选择动作
40.         if (!next_step.empty())
41.             if (1 - epsilon < 0.5) {
42.                 // 以 1 - epsilon 概率进行“exploit”（利用）
43.                 double best_value = -10000000;
44.                 S.action = next_step[0];
45.                 for (int i = 0; i < next_step.size(); ++i) {
46.                     auto index = state(board, next_step[i]);
```

```
                    if (Sarsa_Q.find(index) == Sarsa_Q.end())
                        continue;
                    else if (Sarsa_Q[index] > best_value) {
                        best_value = Sarsa_Q[index];
                        S.action = next_step[i];
                    }}}
            else
                // 以 epsilon 概率进行“explore”（探索）
                S.action = next_step[rand() % next_step.size()];
        double new_q = 0, old_q = 0;
        if (Sarsa_Q.find(pre) != Sarsa_Q.end())
            old_q = Sarsa_Q[pre];
        if (Sarsa_Q.find(S) != Sarsa_Q.end())
            new_q = Sarsa_Q[S];
        // 更新表 Q
        Sarsa_Q[pre] = old_q + alpha * (reward + gama * new_q - old_q);
        pre = S;}}
```

实验结果及分析

· 实验结果展示

1、Q-learning

将ε设置为 0.5，学习速率α设置为 0.8，折扣因子γ设置为 0.5，与随机数对局训练 50 万局，Q 表中存储了 12559838 个不同的棋局 S 和下子位置 A 以及对应的 Q 值。与随机数对战 100 局，胜率为 68%。

2、Sarsa

将ε设置为 0.5，学习速率α设置为 0.8，折扣因子γ设置为 0.5，与随机数对局训练 100 万局，Q 表中存储了 24834550 个不同的棋局 S 和下子位置 A 以及对应的 Q 值。与随机数对战 100 局，胜率为 69%。

3、Alphabeta 剪枝

Alphabeta 剪枝使用的是实验八中的代码，只不过在评价函数中新增了内部稳定子的估计值和靠近边角的估计值，设置搜索深度为 6，与随机数对战 100 局，胜率为 92%。

· 评测指标展示

从实验结果展示可以看出，强化学习中的 Q-learning 和 Sarsa 并不适合黑白棋，因为黑白棋具有太多的棋局和对应每个棋局不同的下子位置, 这导致实际运用中根本无法存储这么多种情况，再者它与随机数对局训练，这导致每次训练的棋局也会有所不同，从而 Q 表中对应的值收敛较慢，即时训练 100 万次，但对于 1 亿多个棋局和下子位置，这些训练次数无法使得它们对应的 Q 值收敛。

与强化学习中的 Q-learning 和 Sarsa 相反，Alphabeta 剪枝适合黑白棋，从实验结果展示可以看出它的胜率远远高于另外两个。