人工智能实验

project2:使用强化学习的一些方法优化黑白棋

16 级计科教务 2 班

16337327

郑映雪

实验题目

使用强化学习的一些方法优化黑白棋

（分工：我负责实现遗传算法，队友负责实现 Qlearning 和 sarsa）

实验内容

算法原理

1、遗传算法

顾名思义，遗传算法的灵感来源于生物界的遗传规律。遗传算法现在是解决最优化问题的一种强化学习的算法。下面我用生物学上的一些词汇来分步解释这个算法。

①初始化

随机生成一些问题的解，每个节为一个向量，这个向量可以类比为生物学上的染色体，向量中的每一个元素可以类比为染色体上的基因。

对每条染色体，使用适应度函数计算各自的适应度值。（适应度函数根据算法的应用场景设计）

②复制

与自然界中的基因复制一样，此处的复制就是选择直接进入下一次迭代的向量组合。如下式所示：

$$[a,b,c,d,e]_{上一次} \rightarrow [a,b,c,d,e]_{下一次}$$

③交叉

与自然界的交配一样，此处的交配是进行在两个向量之间的。两个染色体的等位基因，随机选择一个作为新的染色体该基因点上的基因，最终组成一个新的染色体进入下一代循环。在向量上的操作是在两个向量的相同位置上随机选择一个向量元素作为新向量该位置上的元素，新向量进行归一化后，就此进入下一次迭代。如下式所示：

$$v_1 = [a,b,c,d,e]$$

$$v_2 = [a_1,b_1,c_1,d_1,e_1]$$

随机生成：

$$v_{new} = [a,b_1,c_1,d,e_1]$$

④突变

自然界中存在着基因突变。遗传算法中，也有基因突变的存在。对每一次迭代，随机选择一条染色体，随机改变向量上的某一个元素为一个随机值，得到的新向量进行归一化后，就此进入下一次迭代。如下式所示：

$$v = [a, b, c, d, e]$$

随机选择一个向量，随机选择一个值改变：

$$v_{new} = [a, b, c, d, e_2]$$

⑤结束的标志

可以采用收敛或人工限定迭代次数来使算法结束。在本项目中，为了游戏的可玩性，不想耽误太多时间，所以人工限定了迭代的次数。

2、遗传算法在本次实验中的应用。

在本实验中，我是采取遗传算法与博弈树结合的方法。在大体上采取博弈树的情况下，评估函数成了算法“抗打”的一个重要方面。在评估函数里，每个计算方面如何取权重是一个问题。所以我采取遗传算法在这些权重的选择上。下面是各步骤在本实验中实现方法。

首先，我采取“随机下”的方式，随机生成了棋面，又因为刚开场的棋面计算评估函数意义不大，快结束的棋面步骤有限，所以我选择了 20 个中间状态的棋面。对于每一个棋面，进行下面的操作：

①初始化

原本的评估函数考虑了 7 个方面。

a. 随机生成 7 个数组成一个向量，并进行归一化处理。重复生成 10 个向量。

b. 对这 10 个向量适应度初始为 0，然后进行适应度函数计算。在本实验中我简化成按原函数 7 个方面相乘的权重之和。

c. 对 10 个向量两两 PK，胜方适应度+1。

d. PK 完成后，按适应度大小对 10 个向量进行排序。

②复制

当然是适应度最高的三个有资格进入下一轮 PK，所以直接复制前三名的向量到下一轮的向量组里。

③交叉

在向量排名上取前四名，进行 12、13、14、23、24、34 交叉。交叉过程如下：

a. 定义一个新向量。

b. 先生成一个模 7 随机数，表示从哪里开始截取，再生成一个模 2 随机数，表示是先截取 i 向量还是先截取 j 向量，做到模拟。

c. 对该向量进行归一化处理。

d. 将该向量加入下一轮的向量组里。

④变异

变异的过程如下：

a. 在上一轮的 10 个向量中，生成一个随机数并模 10，该数即为随机决定的突变向量的名次。

b. 随机生成一个数并模 7，得到修改向量的位置。

c. 随机生成一个数，得到赋给选中位置的新值。

d. 对该向量重新进行归一化处理。

e. 将该向量加入下一轮向量组里。

将以上变化之后的新向量组再次重复 PK。

⑤结束迭代的标志

本实验中我设置的结束迭代的的标志为自己设定迭代次数。

至此，比较适合一个棋面的权重向量组合得到。最终选择的组合为在 20 个最优权重向量中随机选择。

⑥适应函数的选择

在适应函数的选择上，我是有些伤脑筋的。因为对棋盘当前局面的评价函数，本身就对适应函数有着诸多限制。适应函数目前有两种方向可以考虑：

a. 对我方有利的适应函数

如果适应函数选取单纯看当前棋局是否对下棋方有利（即单纯选取最有利的下法进行评估），那么在看顾我方胜率的同时也有风险——很可能造成对方先赢、我方后赢的局面。

b. 对弈质量优先的适应函数

上面说到，如果在选择适应函数时，重点对当前棋局是否对下棋方有利而设计，则可能有对方先赢的风险。所以我采取了“往高质量的对弈”来设计。先机器模拟自己与自己

随机下（不会有生成不存在的的情况），这样就避免了从选择上往高质量棋局上偏的情况，然后在该棋面进行遗传算法的迭代使得当前评价函数最高，一个回合后取均值放入向量组里。

如此循环，每个回合的最优权重向量取均值，即为最终的评价函数的向量。

3、本小组实现的其他方法（队友实现）

队友以 epsilon_greedy 算法为基础，实现了 Q-learning 和 Q-learning 的变种 Sarsa。

epsilon_greedy 算法是一种权衡的算法，它在利用和探索之间进行权衡，来实现收益的最大化，是贪心法的一种。它从两个方向出发，一个是利用已有的情况，一个是开发新的情况。对这两种情况分别施加概率，这样可以找到一个最佳收益。

Q-learning 是在 e-greedy 的基础上构造的。它在当前状态执行一个动作后，得到新的状态和及时汇报，并通过当前情况返回更新原表。如下式所示：

$$Q(S_1, A_1) = Q(S_1, A_1) + \alpha\left(R_2 + \lambda \max Q(S_2, a_2) - Q(S_1, A_1)\right)$$

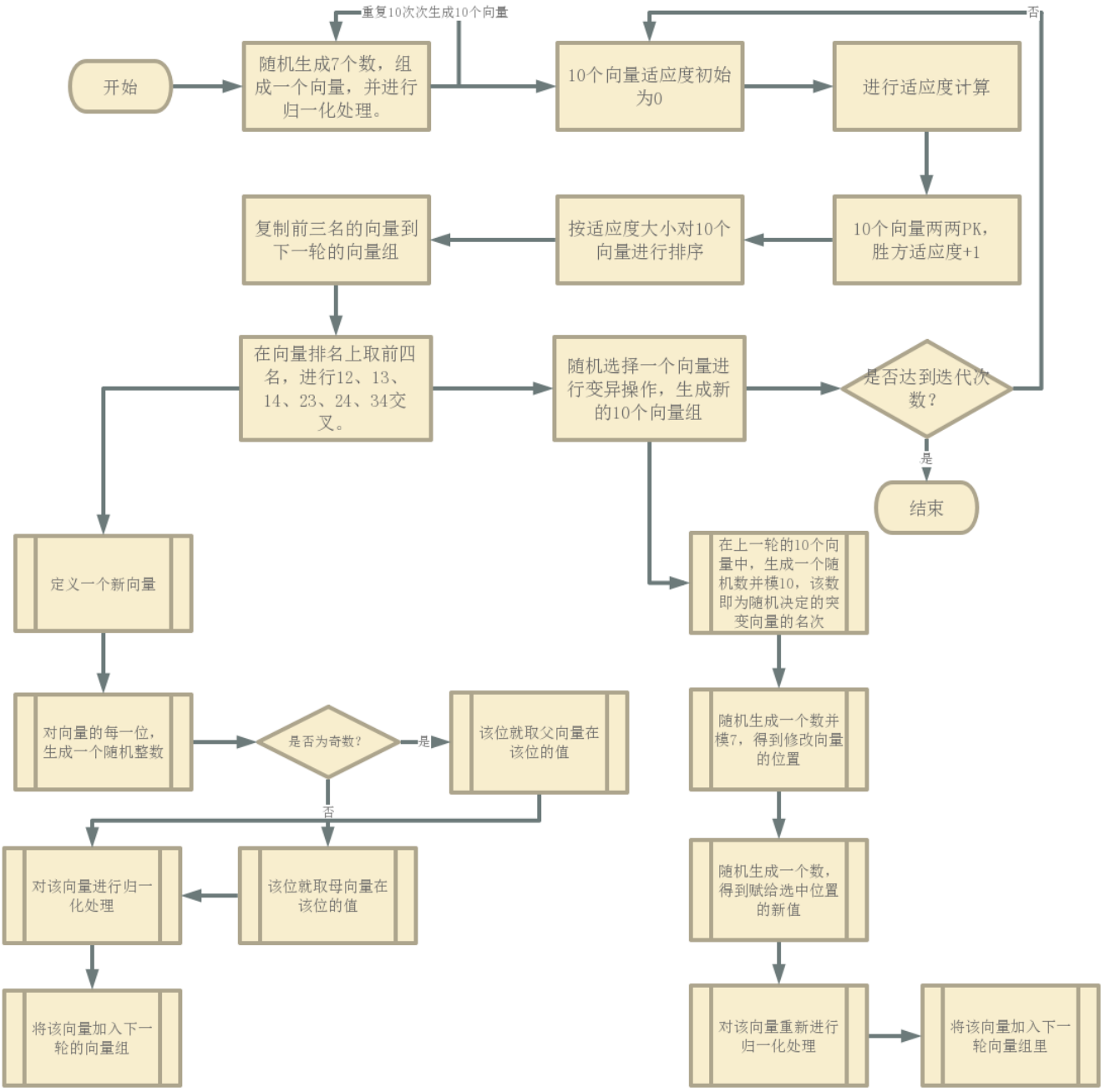
其中$\max Q(S_2, a_2)$表示在状态S_2下的动作中找到具有最大 Q 值的动作所对应的 Q 值。如此反复循环，直到最终状态。

Sarsa 是 Q-learning 的一个变种。它不去寻找当前状态下具有最大 Q 值的结果，而是普遍地寻找动作，直到达到最终状态。

但是，据队友告知，这两种方法的效果并不如博弈树好，我想这是因为在同等规模下，博弈树相比一般的 Qlearning 算法更适合应用在棋类博弈上。

流程图

遗传算法在黑白棋中应用在加权函数部分的流程图如下，其中双线框为该步的子流程：



关键代码

1、准备工作，对当前棋面的 7 个评价方面，随机生成 10 个向量，每个向量里有 7 个元素。

```
    vector<weightstruct> randweight;
        //随机生成 10 个权重向量
        srand(time(0));

        for (int i = 0; i < 10; ++i) {
            vector<double> weighti;
            weightstruct tmp;
            double sum = 0;
            for (int k = 0; k < 7; k++) {
                double tmp = rand() / double(RAND_MAX);
                weighti.push_back(tmp);
                sum += tmp;
            }
            for (int k = 0; k < 7; k++) {
                weighti[k] /= sum;
                tmp.subweight.push_back(weighti[k]);
            }
            tmp.adapt = 0;
            randweight.push_back(tmp);
        }
```

2、10 个向量进行两两 PK，并排出顺序。

```
int n = 30;//进行 30 代迭代尝试
        while (n--) {
            //对当前 10 个向量进行 PK
            double res1, res2;
            for (int i = 0; i < 10; i++) {
                for (int j = i + 1; j < 10; j++) {
                    res1 = randweight[i].subweight[0] * result1 + randweight[i].subweight[1] * result2 + randweight[i].subweight[2] * result3 + randweight[i].subweight[3] * result4 + randweight[i].subweight[4] * result5 + randweight[i].subweight[5] * result6 + randweight[i].subweight[6] * result7;
                    res2 = randweight[j].subweight[0] * result1 + randweight[j].subweight[1] * result2 + randweight[j].subweight[2] * result3 + rand
```

```
    weight[j].subweight[3] * result4 + randweight[j].subweight[4] * result5 +
    randweight[j].subweight[5] * result6 + randweight[j].subweight[6] * result
    7;
                    if (res1 > res2) randweight[i].adapt += 1;  else randw
    eight[j].adapt += 1;

                }
            }
            //淘汰后五个
            weightstruct tmp2;
            for (int i = 0; i < 10; i++) {
                for (int j = 0; j < 9 - i; j++) {
                    if (randweight[j].adapt < randweight[j + 1].adapt) {
                        tmp2 = randweight[j];
                        randweight[j] = randweight[j + 1];
                        randweight[j + 1] = tmp2;
                    }
                }
            }
```

3、复制，此处选择适应度最高的 3 个进行复制。

```
            vector<weightstruct> newrandweight;
            //复制部分，此处选择适应度最高的 3 个进行复制
            newrandweight.push_back(randweight[0]);
            newrandweight.back().adapt = 0;
            newrandweight.push_back(randweight[1]);
            newrandweight.back().adapt = 0;
            newrandweight.push_back(randweight[2]);
            newrandweight.back().adapt = 0;
```

4、交叉，此处选择 12、13、14、23、24 进行交叉。先生成一个模 7 随机数，表示从哪里开始截取，再生成一个模 2 随机数，表示是先截取 i 向量还是先截取 j 向量，做到模拟。再进行归一化处理。

```
1. //交叉部分（模拟生殖），12、13、14、23、24、34 进行交叉
2.             for (int i = 0; i < 4; i++) {
3.                 for (int j = i + 1; j < 4; j++) {
4.                     weightstruct tmp;
5.                     tmp.adapt = 0;
6.                     double tmpsum = 0;
7.                     int rand2 = rand() % 2;
8.                     int cut = rand() % 7;
9.                     for (int k = 0; k < 7; k++) {
10.                        if (rand2 == 0) {
11.                            if (k <= cut) {
12.                                tmp.subweight.push_back(randweight[i].subw
   eight[k]);
13.                                tmpsum += randweight[i].subweight[k];
14.                            }
15.                            else {
16.                                tmp.subweight.push_back(randweight[j].subw
   eight[k]);
17.                                tmpsum += randweight[j].subweight[k];
18.                            }
19.                        }
20.
21.                        else {
22.                            if (k <= cut) {
23.                                tmp.subweight.push_back(randweight[j].subw
   eight[k]);
24.                                tmpsum += randweight[j].subweight[k];
25.                            }
26.                            else {
27.                                tmp.subweight.push_back(randweight[i].subw
   eight[k]);
28.                                tmpsum += randweight[i].subweight[k];
29.                            }
30.
31.                        }
32.                    }
33.                    for (int k = 0; k < 7; k++) {
```

```
                        tmp.subweight[k] = tmp.subweight[k] / tmpsum;
                    }
                    newrandweight.push_back(tmp);

                }
            }
```

5、变异，随机选择一个基因进行变异，变异的值为随机值，再进行归一化处理。

```
 //变异部分，随机选择一个向量元素给予新的随机值
            double replace = (rand() / double(RAND_MAX));
            double sum = 0;
            int rand3 = rand() % 7;
            for (int k = 0; k < 7; k++) {
                if (k == rand3)
                    sum += replace;
                else
                    sum += tmp3.subweight[k];

            }
            for (int k = 0; k < 7; k++) {
                if (k == rand3) {
                    tmp3.subweight[k] = replace / sum;
                    continue;
                }
                else {
                    tmp3.subweight[k] = tmp3.subweight[k] / sum;
                }
            }
            newrandweight.push_back(tmp3);
            randweight = newrandweight;
        }
```

创新点

遗传算法还可进行 2 个优化：

①陪跑员机制。可以挑选优秀的个体，作为陪跑员，但是不参与复制、交叉和变异的过程。这样可以加快收敛的速度，尽快选出优秀的个体。

②概率进化。之前进行的复制和交叉都是尽量选择优秀的个体，但是可能在收敛的过程中造成局部最小值，所以对于优秀的个体设置一定的概率再进行进化，否则再次生成随机向量。

实验结果及分析

实验结果展示

以下是部分权重向量的截取：

```
0.118715 0.0474599 0.0820541 0.406253 0.0887423 0.211214 0.0455612
0.101621 0.251132 0.187868 0.20652 0.0252135 0.122785 0.104861
0.0621216 0.260733 0.28112 0.212138 0.0545068 0.108532 0.020849
0.181935 0.272466 0.316085 0.00890415 0.0392359 0.15151 0.0298634
0.0496528 0.107295 0.121381 0.22426 0.286124 0.152228 0.0590597
0.0843375 0.247587 0.154348 0.0261199 0.301987 0.0700135 0.115607
0.109217 0.255759 0.198605 0.0298836 0.318184 0.0569039 0.0314485
0.128122 0.260306 0.198877 0.0835897 0.31501 0.0109095 0.00318459
0.18641 0.148806 0.138029 0.0922368 0.30242 0.049404 0.0826946
0.194259 0.152363 0.13287 0.092959 0.304242 0.0290752 0.0942327
0.15438 0.254647 0.0934263 0.088121 0.307803 0.0252044 0.0764182
0.155251 0.134685 0.141733 0.0930956 0.31018 0.0516368 0.11342
0.204777 0.0709432 0.161427 0.0793043 0.298624 0.0661732 0.118752
0.183891 0.301602 0.0904167 0.164266 0.0967004 0.0774991 0.0856246
0.179225 0.282235 0.14931 0.074669 0.127729 0.0839515 0.10288
0.151471 0.292817 0.0839126 0.154672 0.128928 0.0361695 0.15203
0.243581 0.271186 0.106578 0.0167714 0.133032 0.0434442 0.185407
0.167671 0.304922 0.150987 0.139182 0.10963 0.0250636 0.102545
0.148203 0.273215 0.218171 0.0991961 0.0802153 0.0315204 0.14948
0.148206 0.273752 0.218279 0.0992101 0.0802887 0.0314023 0.148862
0.148193 0.271336 0.217794 0.0991469 0.079959 0.0319329 0.151638
0.191147 0.303112 0.199846 0.015577 0.0520255 0.102891 0.135401
0.212346 0.130049 0.170609 0.0838888 0.18266 0.123715 0.0967325
0.149675 0.0707875 0.192436 0.159671 0.32292 0.0566938 0.0478169
0.213078 0.181266 0.204858 0.0204371 0.07253 0.0583557 0.249474
0.160482 0.0899875 0.150738 0.253119 0.109183 0.199308 0.037183
0.257691 0.338549 0.109043 0.100102 0.0775074 0.0679404 0.0491671
0.180888 0.250036 0.277443 0.0241455 0.0743652 0.0357808 0.157342
0.151633 0.294526 0.0903416 0.154453 0.127784 0.0349313 0.146331
0.152032 0.174689 0.125297 0.20468 0.083455 0.238842 0.0210053
0.118715 0.0474599 0.0820541 0.406253 0.0887423 0.211214 0.0455612
0.101621 0.251132 0.187868 0.20652 0.0252135 0.122785 0.104861
0.0621216 0.260733 0.28112 0.212138 0.0545068 0.108532 0.020849
0.181935 0.272466 0.316085 0.00890415 0.0392359 0.15151 0.0298634
0.0496528 0.107295 0.121381 0.22426 0.286124 0.152228 0.0590597
0.0843375 0.247587 0.154348 0.0261199 0.301987 0.0700135 0.115607
0.109217 0.255759 0.198605 0.0298836 0.318184 0.0569039 0.0314485
0.128122 0.260306 0.198877 0.0835897 0.31501 0.0109095 0.00318459
0.18641 0.148806 0.138029 0.0922368 0.30242 0.049404 0.0826946
0.194259 0.152363 0.13287 0.092959 0.304242 0.0290752 0.0942327
0.15438 0.254647 0.0934263 0.088121 0.307803 0.0252044 0.0764182
0.155251 0.134685 0.141733 0.0930956 0.31018 0.0516368 0.11342
0.204777 0.0709432 0.161427 0.0793043 0.298624 0.0661732 0.118752
0.183891 0.301602 0.0904167 0.164266 0.0967004 0.0774991 0.0856246
```

由于不同棋面的差异，所以棋盘的收敛不一定都是相同。但可以大致看出，最终迭代到一定程度后，权重的平均值为[0.15,0.25,0.2,0.1,0.3,0.05,0.1]。在均值上下波动的原因来自于评价棋盘的不同棋面。为了权衡选择一个向量组，我选择了计算均值。

评测指标展示

选取搜索层数为 4（不一定能完全下过随机数），进行多次对弈：

对弈局数	10	20	100
胜率	0.8	0.85	0.79

同样搜索层数为 4 的情况下，对一开始自己设置的权重，对弈结果如下：

对弈局数	10	20	100
胜率	1	0.85	0.75

10 局的时候我被吓到了，但是 100 局的时候情况还是好了一些（原因可能是随机数有时会出其不意，有时可能会笨笨的）。看来，遗传算法还是起了一丢丢丢丢的作用。