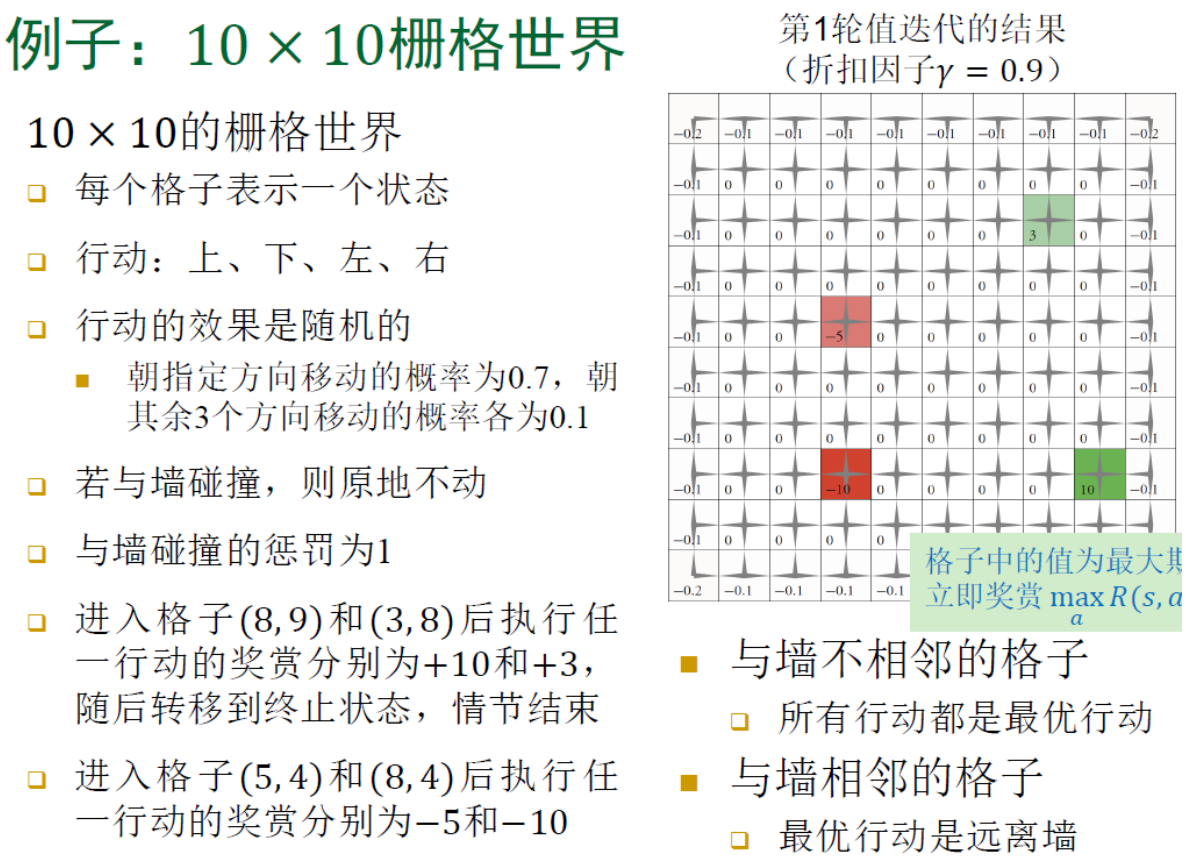
智能系统作业

人工智能学院　丁豪
181220010@smail.nju.edu.cn

问题描述



对问题进行建模，我们发现需要建模的地方主要有两个方面，对于坐标、行动-下一状态的匹配（比如游戏结束、是否撞墙等的特殊处理）以及对于坐标、行动-奖赏的描述，因此在我们的程序中分别使用这两个函数来描述他们。在S_next函数中，如果走到墙边，将不改变坐标，如果走到约定的4个结束为止，将返回负值，用于后面判断。

```
def R(x,y,a):
def S_next(x,y,a):
```

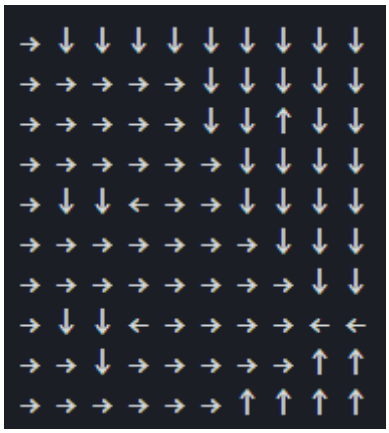
注意到行动空间为{left,right,up,down}，为了方便程序中使用，我们规定其对应0-3这4个整数。

实验的目的是求解最优值函数/最优策略，因此我们的目标就是通过特定算法，最终输出我们的最优值函数与策略图。对值函数的输出，采用保留两位小数，而最优策略直接使用字符↑↓←→来直观表示。

```
def out_U(U):
def out_policy(policy):
```

他们两者的样例输出分别为：

```
0.46 0.77 0.99 1.26 1.60 2.00 2.40 2.56 3.00 2.67
0.54 0.86 1.15 1.52 2.00 2.47 2.93 3.00 3.69 3.31
0.43 0.72 0.94 1.20 2.33 2.96 3.54 4.10 4.53 4.04
0.41 0.77 0.59 -5.00 2.24 3.49 4.20 4.93 5.50 4.87
0.66 1.11 1.48 1.94 3.35 4.18 4.97 5.85 6.68 5.84
0.66 1.06 1.37 1.80 3.82 4.88 5.85 6.92 8.15 6.94
0.53 0.86 0.22 -10.00 3.16 5.36 6.67 8.15 10.00 8.19
0.77 1.19 1.51 1.88 3.83 4.87 5.84 6.92 8.15 6.94
0.86 1.42 1.96 2.60 3.37 4.11 4.91 5.83 6.68 5.82
```



算法描述

1）策略评价：使用逐次逼近法迭代地计算值函数。注意到题目中的策略为随机策略，因此在计算值函数的时候除了策略记录的“主”方向之外，还对另外三个方向以0.1的权重进行求和。上面我们提到，如果S_next返回负值说明我们的情节结束了，因而不必再加上U(s)项。

```
# 随机策略，0.7，3个0.1
for a in range(4):
    x_next,y_next = S_next(i,j,a)
    if a==pi[i][j]:
        U_new[i][j] += 0.7*R(i,j,a)
        if x_next>=0:
            U_new[i][j] += 0.7*gamma*U_old[x_next][y_next]
    else:
        U_new[i][j] += 0.1*R(i,j,a)
        if x_next>=0:
            U_new[i][j] += 0.1*gamma*U_old[x_next][y_next]
```

2）策略迭代：初始策略可以任意选取，这里方便起见直接初始化为全0，也就是全部向左。之后我们迭代地进行如下操作：由上一次产生的π构造此次使用的U,求解最大化行动值函数的新π, 不断重复直到π不再变化。

```
gamma = 0.9 # 衰减系数
pi_0 = [[0 for j in range(10)] for i in range(10)] # 初始策略函数，采取主方向全部向左
def policy_iteration():
```

3）值迭代：此方法迭代的计算考虑k步收益情况下的值函数，值函数最终将收敛于最优值函数，而最优策略可以通过最优值函数导出。这里使用$\delta = \frac{\epsilon(1-\gamma)}{\gamma}$，作为收敛条件的判别。

```
epsilon = 1e-6
gamma = 0.9
U_0 = [[0 for j in range(10)] for i in range(10)] # 初始值函数
def get_pi_from_U(U):
def value_iteration():
```

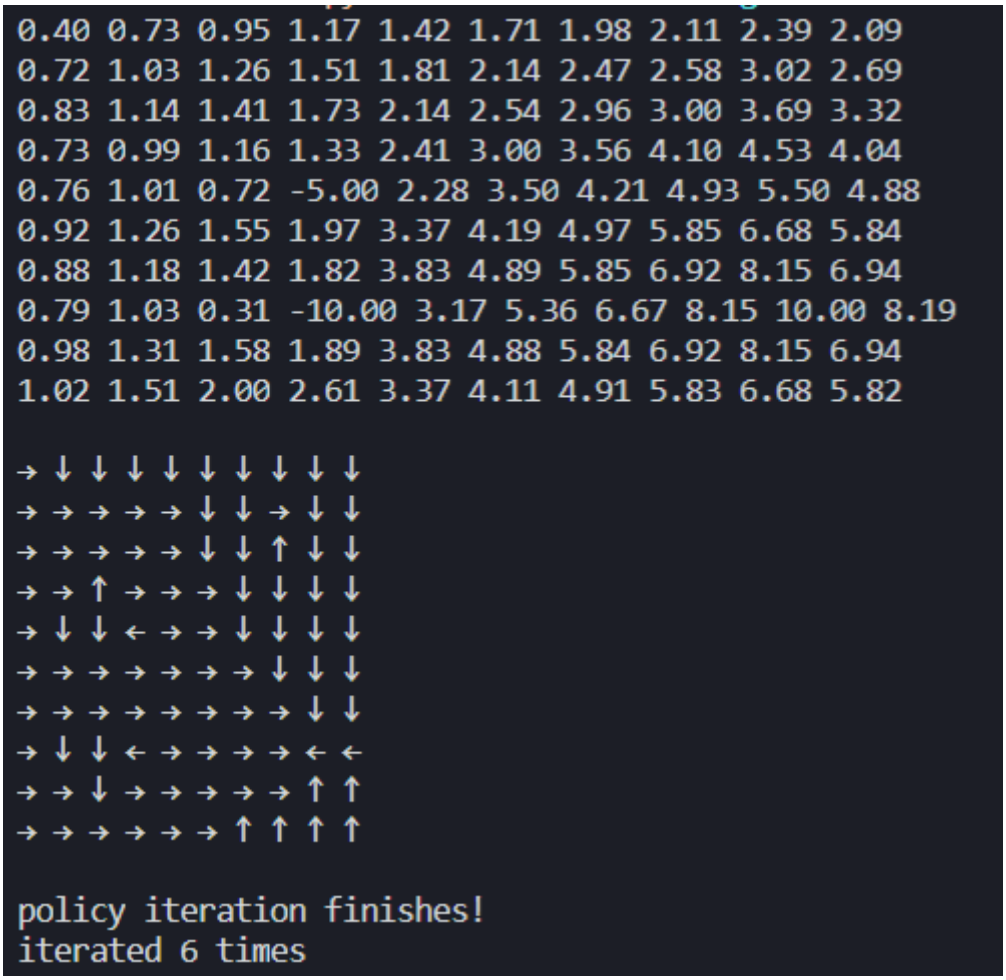
4）高斯-赛德尔值迭代：高斯赛德尔值迭代与普通值迭代的最大区别在于他是"就地"更新U值，而不需要有新老版本之分，因此基于上面的值迭代算法简单修改就得到了我们的高斯-赛德尔迭代算法，其中get_pi_from_U可以复用。

```
gamma = 0.9
U_0 = [[0 for j in range(10)] for i in range(10)] # 初始值函数
def gauss_seidel():
```

实验

实验的参数设置在上一节已经讲过了，这里直接来看结果

1）策略迭代



2）值迭代

```
0.40 0.73 0.95 1.17 1.42 1.71 1.98 2.11 2.39 2.09
0.72 1.03 1.26 1.51 1.81 2.14 2.47 2.58 3.02 2.69
0.83 1.14 1.41 1.73 2.14 2.54 2.96 3.00 3.69 3.32
0.73 0.99 1.16 1.33 2.41 3.00 3.56 4.10 4.53 4.04
0.76 1.01 0.72 -5.00 2.28 3.50 4.21 4.93 5.50 4.88
0.92 1.26 1.55 1.97 3.37 4.19 4.97 5.85 6.68 5.84
0.88 1.18 1.42 1.82 3.83 4.89 5.85 6.92 8.15 6.94
0.79 1.03 0.31 -10.00 3.17 5.36 6.67 8.15 10.00 8.19
0.98 1.31 1.58 1.89 3.83 4.88 5.84 6.92 8.15 6.94
1.02 1.51 2.00 2.61 3.37 4.11 4.91 5.83 6.68 5.82

→ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
→ → → → → ↓ ↓ → ↓ ↓
→ → → → → ↓ ↓ ↑ ↓ ↓
→ → ↑ → → → ↓ ↓ ↓ ↓
→ ↓ ↓ ← → → ↓ ↓ ↓ ↓
→ → → → → → → ↓ ↓ ↓
→ → → → → → → → ↓ ↓
→ ↓ ↓ ← → → → → ← ←
→ → ↓ → → → → → ↑ ↑
→ → → → → → ↑ ↑ ↑ ↑

value iteration finishes!
iterated 67 times
```

3）高斯-赛德尔值迭代

```
0.13 0.48 0.69 0.91 1.17 1.50 1.86 2.06 2.35 2.04
0.46 0.77 0.99 1.26 1.60 2.00 2.40 2.56 3.00 2.67
0.54 0.86 1.15 1.52 2.00 2.47 2.93 3.00 3.69 3.31
0.43 0.72 0.94 1.20 2.33 2.96 3.54 4.10 4.53 4.04
0.41 0.77 0.59 -5.00 2.24 3.49 4.20 4.93 5.50 4.87
0.66 1.11 1.48 1.94 3.35 4.18 4.97 5.85 6.68 5.84
0.66 1.06 1.37 1.80 3.82 4.88 5.85 6.92 8.15 6.94
0.53 0.86 0.22 -10.00 3.16 5.36 6.67 8.15 10.00 8.19
0.77 1.19 1.51 1.88 3.83 4.87 5.84 6.92 8.15 6.94
0.86 1.42 1.96 2.60 3.37 4.11 4.91 5.83 6.68 5.82

→ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
→ → → → → ↓ ↓ ↓ ↓ ↓
→ → → → → ↓ ↓ ↑ ↓ ↓
→ → → → → → ↓ ↓ ↓ ↓
→ ↓ ↓ ← → → ↓ ↓ ↓ ↓
→ → → → → → → ↓ ↓ ↓
→ → → → → → → → ↓ ↓
→ ↓ ↓ ← → → → → ← ←
→ → ↓ → → → → → ↑ ↑
→ → → → → → ↑ ↑ ↑ ↑

Gauss-Seidel finishes!
iterated 14 times
```

我们可以发现，策略迭代与值迭代的最优值函数是完全相同的，而高斯赛德尔迭代则在某些部分和他们有些差异。这是因为在算法中高斯-赛德尔值迭代的终止条件是策略不再变化，而非值函数不再变化，在此基础上如果继续迭代下去值函数还有可能继续收敛，并在无限步后与上两种算法的最优值函数相等。

同时三种算法的最优策略是完全相同的，这由两点保证。首先算法的正确性保证了输出的一定是最优策略，其次因为我们在遍历 left right up down四个方向的时候总是使用相同的顺序0-3, 因而等价的最优策略总会选取到相同的那一个优先解作为最终输出。有了这两点进行保证，最终的最优策略方阵就应当是完全相同的。

观察迭代次数，我们可以发现策略迭代和高斯-赛德尔迭代的次数都比值迭代要小很多。值迭代依靠最优质函数的收敛性来间接推定最优策略函数，这里我们的ϵ设置较小，因而收敛所需迭代次数较多，同时在收敛后期有一些迭代可能并不会改变最优策略函数而只是单纯的收敛最优值函数，因此他的终止条件相比策略迭代和高斯赛德尔值迭代是更加苛刻的，所以需要更多迭代次数。