人工智能导论搜索大作业　2048

自 55 班　刘乐章　　2015011471

2018 年 4 月 10 日

1　Readme

1.1　运行环境

- Linux Ubuntu 16.04
- Python 3.5.2
- Cython 0.28.1
- gcc 5.4.0

1.2　文件说明

压缩包中包含以下五个文件：

- PlayerAI_3.py 2048AI 主文件
- search.pyx　核心搜索算法 Cython 文件
- setup.py　Cython 配置文件
- search.c　核心搜索算法 C 文件
- search.cpython-36m-x86_64-linux-gnu.so　搜索算法动态链接库

将上述五个文件复制进入测试环境中，运行命令

```
python setup.py build_ext --inplace
```

待编译成功后，即可开始进行测试。

注　由于预先并不知道会在何种环境下进行测试，因而有可能出现超时的情况（在我的电脑上没有超时的情况发生）。如果出现程序编译错误或产生大量超时，麻烦助教立即联系我，**联系方式：13522112264**。多谢助教！

2 算法设计

算法设计主要包含三部分：启发函数、搜索算法、技巧 (Tricks)。

2.1 启发函数

启发函数设计主要考虑三点：单调性、光滑性和空格数目。另外还有防止最大数字离开的惩罚项。

2.1.1 单调性

由于只有相同的数字才能够合并，因而保持棋盘格整体的有序（即单调）非常重要。我们采用将单调递增奖励权重 S 型排列的方式实现。经过实践，我们将权重设置如下表所示。

3	1	1	1
9	3	3	3
3^5	3^6	3^7	3^8
3^5	3^6	3^7	3^8

2.1.2 平滑性

在保证单调性的基础上，进一步，我们考虑平滑性——即让相邻的数字尽可能地接近（以便于合并）。对于每个数字计算其与相邻数字的差的绝对值之和，作为惩罚项。设全部位置集合为 S，点 i 对应的值为 a_i，相邻位置集合为 D_i，惩罚项：

$$L = -\sum_{i \in S} \sum_{j \in D_i} |a_i - a_j|$$

一种具有完美平滑性的情况如下表所示。

2	2	2	2
2	2	2	2
2	2	2	2
2	2	2	2

2.1.3 空格数目

为了进一步鼓励格点合并，我们引入棋盘格上空格点的数目作为奖励项，奖励系数为。

2.1.4 最大数字离开惩罚

为了防止偶尔最大数字离开右下角点的情况，我们引入最大数字离开惩罚。当发现最大奖励权重点（右下角）的数字不是最大数字时，我们取消该点的奖励权重。即只有最大数字在最大奖励权重点时，才会产生最大奖励权重。

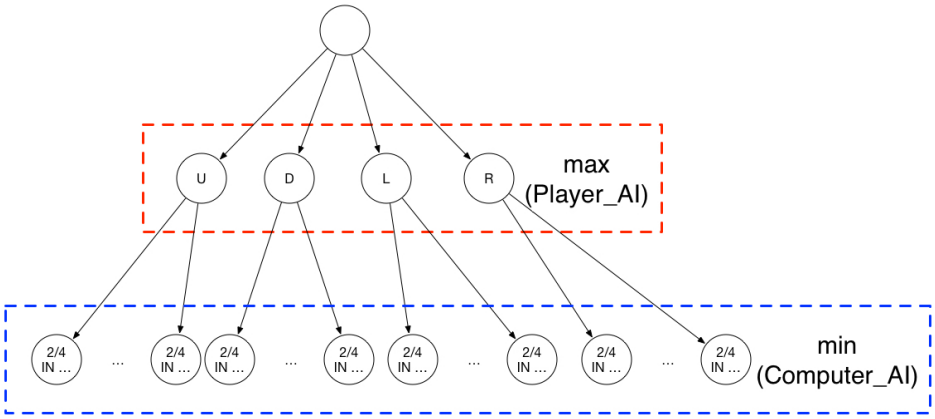

图 1: minimax 搜索

2.2 搜索算法

2.2.1 带 $\alpha - \beta$ 剪枝的 minimax 搜索

首先，根据实验要求，我们实现了经典的带 $\alpha - \beta$ 剪枝的 minimax 搜索作为搜索算法。对于 Player_AI，我们进行 max 搜索，对于 Computer_AI，我们进行 min 搜索。minimax 搜索的示意图如图 1 所示。

算法伪代码如图 2 所示。

在实验的过程中，我们发现，采用 minimax 搜索在此任务上不能达到特别好的效果。分析其原因，我认为，是 2048 游戏本身的特点所致：2048 游戏的双方并不是每步执行最优策略，Computer_AI 的行为具有随机性。因而采用 minimax 会显得太过“谨慎”，反而不能达到更好的效果。

在此基础上，我征求了张长水老师的意见，决定采用 expectimax 搜索代替 minimax。

2.2.2 expectimax 搜索

将 minimax 搜索的取最小值的 min 层用求期望的 expect 层代替，即得到 expectimax 搜索。其示意图如图 3 所示。采用 expectimax 搜索后，算法效果得到明显改善。但由于无法进行 $\alpha - \beta$ 剪枝，搜索层数被限制在了 4 层。

2.2.3 带采样的 expectimax 搜索

为了增加 expectimax 的搜索深度，我考虑采用采样（Sampling）的方法来代替简单地计算期望。

事实上，expectimax 的效率瓶颈主要来源于在空格数较多的时候分支过多。而在空格数（分支数）很多时，情况并不紧急，在很多位置放置数字差别都不大。因而可以考虑采用采样的方法，在分支数较多时适当随机剪掉一些分支，从而使得深度增加。

带采样的 expectimax 搜索如图 4 所示。

在加入采样后，搜索的深度可以明显增加，可以多搜索 1-2 层（取决于采样数）。但与此同时，算法表现的方差也随之增大，会出现由于欠采样而导致过于“冒险”，发挥不稳定的情况。

```
function alphabeta(node, depth, α, β, maximizingPlayer)
     if depth = 0 or node is a terminal node
         return the heuristic value of node
     if maximizingPlayer
         v := -∞
         for each child of node
             v := max(v, alphabeta(child, depth - 1, α, β, FALSE))
             α := max(α, v)
             if β ≤ α
                 break (* β cut-off *)
         return v
     else
         v := +∞
         for each child of node
             v := min(v, alphabeta(child, depth - 1, α, β, TRUE))
             β := min(β, v)
             if β ≤ α
                 break (* α cut-off *)
         return v
```

图 2: 带 $\alpha - \beta$ 剪枝的 minimax 搜索伪代码

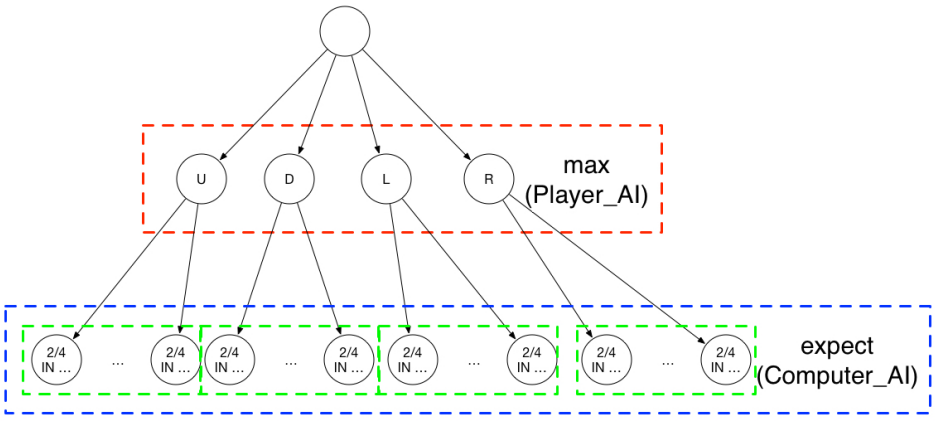

图 3: expectimax 搜索

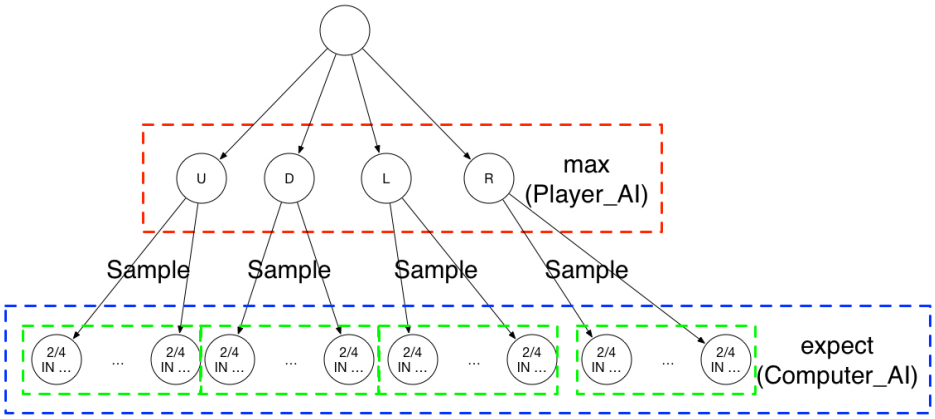

图 4: 带采样的 expectimax 搜索

2.3 Tricks

2.3.1 根据空格数目调整搜索深度

除了可以调整采样数之外，实际上类似地，也可以根据局势调整搜索深度。

由于在空格数较多的时候，局势比较缓和，各个位置放置数字，计算效果均相差不大，可以适当减少搜索深度；而在空格数较少的时候，局势比较紧张，则可以增大搜索深度。

总之，局势的乐观程度和空格数/采样数有正相关关系。并且总计算量和采样数与深度的乘积正相关，保持一定。

2.3.2 采用 Cython 加速 Python 代码

除了算法设计，在具体实现上，我改写了提供的内部代码，并用 Cython 进行加速。直观上看，相比于使用 Python，Cython 大约获得了一倍的性能提升，在常数级别上进行了优化。

3 实验结果

运行代码进行实验：多数情况下，该算法能够合成 4096；在少数情况下，该算法合成至 2048；在较为理想的情况下，该算法能合成 8192。


Computer's turn:
2 4 2 4
4 64 32 2
32 512 128 64
128 4 1024 8192
8192
Process finished with exit code 0

图 5: 实验结果