四子棋实验报告

计91 刘松铭 2018011960

摘要

在重力四子棋对弈问题中，我设计并实现了四个基于蒙特卡洛搜索算法 (MCTS)的模型，即**朴素MCTS**，**改进MCTS**，**启发式模拟MCTS**，**常数优化MCTS**。上述四个模型大体框架基本一致，但在技术细节上分别采用了不同的优化方法。因此这四个模型在性能上会有所差异。

我对四个模型进行了详尽的对比测试，从网站批量测试和本地单个AI胜率 (如对决94号AI的胜率)两个维度对算法的性能进行评价。结果显示表现最好的是**常数优化MCTS**，其在网站批量测试的平均胜率达到了**99.6%**。本作业最终提交所采用的模型也因此设为**常数优化MCTS**。

注：网站测试采用Saiblo网站进行测试。本机的实验测试环境为11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 16.0GB RAM Win10 x64。

朴素MCTS

朴素MCTS模型是基于蒙特卡洛搜索算法 (MCTS)的初版模型，仅仅在算法层面上完整实现了MCTS算法，并没有加入任何的优化，是MCTS算法的最小可用模型。后续三个模型都是在该模型基础上的优化版本。

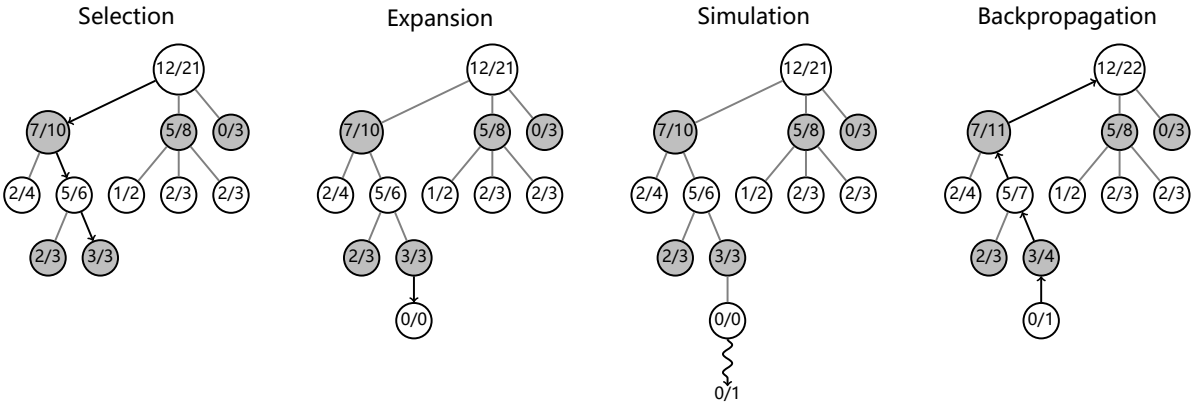
算法思路

本模型采用MCTS算法进行重力四子棋每一步的落子决策。具体算法简要介绍如下：

重力四子棋落子决策是这样一个问题：对于给定的局面S_i，给出一个最好的落子位置x_i。假设S_i中一定有位置可以落子。

蒙特卡洛搜索树是一颗多叉树，每个节点代表了一个局面，包括该节点的探索次数，该节点的胜率，该节点到子节点的引用。特别地，根节点代表了给定的局面S_i。搜索树的扩展过程就是从根节点开始沿着树向下扩展的过程，每扩展一个节点就相当于模拟了一步我方/对方的落子，这与极大极小值算法中的搜索过程是一致的。

具体来说，MCTS包含如下四步：



- 选择：从根节点R开始连续向下选择，直至抵达叶子节点L。为了更好地模拟每一个决策 (即根节点的若干子节点对应的决策)的收益，每次选择总应该选择对当前一方"最有希望"的节点。这里采用了UCT (Upper Confidence Bounds to Trees，上限置信区间算法)进行选择：

$$f_i = \frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$

上式中：

- w_i指第i个子节点的获胜次数。
- n_i指第i个子节点的模拟次数。
- c为参数，是一个平衡胜率和探索程度的因子，c越大越倾向于探索没有被完全探索的节点，反之倾向于胜率高的节点。该参数是`参数调优`部分重点优化的参数。
- t为所有子节点的模拟总次数，即$\sum n_i$。

在每一次选择中，总是选择f_i最大的子节点进行移动。

- 扩展：若在L处胜负已分，则直接转第四步 (此时$C \equiv L$)。否则从当前节点中**随机**创建一个新的子节点C。
- 模拟：从C开始随机落子，直至胜负可分。
- 反向传播：将C模拟的结果逐层向上传播，更新每个节点信息。

实现细节

下面介绍朴素MCTS的一些实现细节：

- 存储蒙特卡洛搜索树的数据结构：

  该搜索树采用了12叉树进行存储。若某个节点不具备12个孩子，则将其余的孩子置为空。为了加速搜索过程，统一采用静态内存池进行存储，不使用`new`和`delete`关键词，不使用指针进行访问，而是使用某节点在静态内存池中的秩进行访问。

- 存储局面的数据结构：

  对于某一局面，需要**频繁地**进行如下两类操作：

  - 在某个位置落子，或者去除某个位置的落子。
  - 判断当前局面是否有四个子连起来。

  如果粗暴地采用二维数组进行存储，则第二类操作将会无法很快地完成。

  为了进一步加速模拟，我采用了**位图**存储每一个局面。位图使用一个`int`型变量存储每一行，每一列，每一左上右下斜列，每一右上左下斜列的落子情况。每两位表示一个位置的落子情况。例如：如下是表示某一列落子情况的串

  ```
  01 10 00 00 00 ...
  ```

  `01`表示该列底下是一个黑棋，`10`表示黑棋之上是一个白棋，`00`表示再往上是空，没有落子。

  这样一来，无论是落子、去除落子还是判断局面是否结束都可以很快完成

  - 落子

    ```
    line[x] |= (1 << ((y << 1) + player));
    ```

    `line[x]`表示第`x`列对应的串，`y`表示行，`player`表示黑棋或白棋。

  - 去除落子

    ```
    line[x] -= (1 << ((y << 1) + player));
    ```

  - 判断是否结束

    ```
    ((line[x] & (line[x] >> 2) & (line[x] >> 4) & (line[x] >> 6)) ? 1 : 0)
    ```

如上只是判断第`x`列是否有四个子连起来的条件。完整的判断还需考虑上一步落子位置对应的行，左上右下斜列，右上左下斜列是否有四个子连起来。只要有一个连起来则说明游戏结束，上一步落子的玩家胜利。

- 剪枝：

  对于一些特殊的局面，需要进行直接决策：

  - 若我方下某一列可以赢，则我方一定要下这一列。
  - 若对方下某一列可以赢，且这样的列只有一列，则我方一定要下这一列。
  - 若对方下某一列可以赢，且这样的列有两列及以上，则我方直接认输，停止扩展。
  - 当我方下某一列后，对方再下这一列可以赢，则我方一定不要下这样的列。

  如此进行剪枝可以避免很多无效的扩展，提高搜索的效率。

- 记录所有节点

  为了充分利用模拟的信息，在模拟过程中随机落子产生的局面也会被记录下来。因此反向传播过程需要从决定胜负的节点开始，逐层往上更新各个节点的信息，而不是从L开始更新。

- 最终落子选择

  本程序事先规定了最大搜索时长和最大局面数，分别规定为**2.7s**和**1000000**。

  在时长结束或局面数达到最大局面数后，会进入最终落子选择阶段。最终会选择根节点R的所有子节点中**探索次数最多**的节点对应的落子位置作为输出——即给定S_i局面下的最佳落子位置x_i。

参数优化

固定$c=1$，最终落子选择**胜率最大**的节点，与天梯最强AI94号AI对决20回合结果如下 (本地测试)：

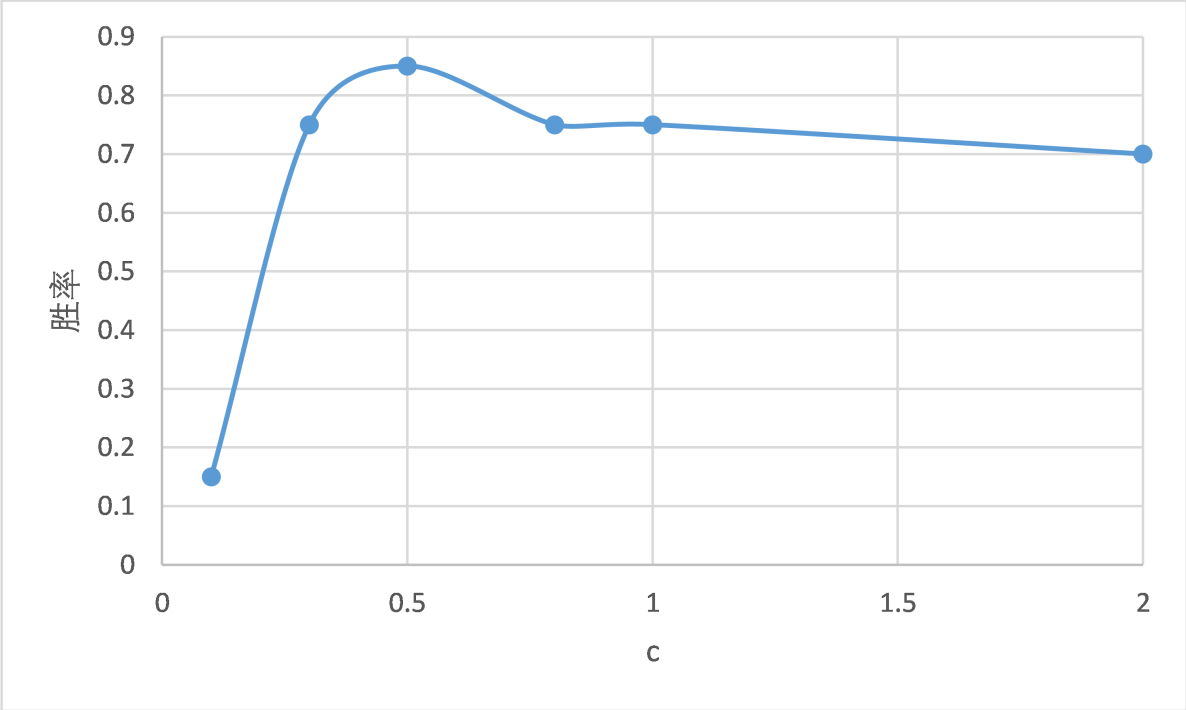
```
Stat:
ratio of A wins : 0.7
ratio of B wins : 0.3
ratio of Tie : 0

ratio of (A wins + tie) : 0.7
ratio of (B wins + tie) : 0.3
```

注：`A`为该模型，`B`为样例94号AI，**下同。**

固定$c=1$，最终落子选择**探索次数最多**的节点，与天梯最强AI94号AI对决20回合结果如下 (本地测试)：

```
Stat:
ratio of A wins : 0.75
ratio of B wins : 0.25
ratio of Tie : 0

ratio of (A wins + tie) : 0.75
ratio of (B wins + tie) : 0.25
```

固定最终落子选择**探索次数最多**的节点这一策略，下面取不同的c值与天梯最强AI94号AI对决20回合结果如下 (本地测试)：



由如上参数调优结果，最后选择$c = 0.5$，**探索次数最多**策略。

测试结果

在本机上进行测试，输出搜索次数，每次决策的平均搜索次数为**9w**次。 (除去对局开头和结尾的特殊情况)

利用网站的批量测试平行测试五次，得到如下的统计结果。

批量测试编号	胜数	负数	平数	胜率 (%)
#10238	96	4	0	96
#12487	100	0	0	100
#12483	94	6	0	94
#12477	96	4	0	96
#13053	97	3	0	97

朴素MCTS的批量测试平均胜率为**96.6%**。

改进MCTS

改进MCTS模型是基于朴素MCTS的优化模型。

优化思路

在朴素MCTS中，模拟部分的随机落子产生的局面也会被记录下来，这样虽然能够最大程度的利用模拟产生的信息，但是有两点不足：

- 记录这些信息需要浪费大量的时间，会增加模拟的开销，减小搜索次数。
- 下方的节点非常稀疏。此外由上往下节点数几乎是指数增加的，因此下方节点信息很难被真正利用到。

因此做出的优化是在模拟阶段产生所有局面都不记录下来，只是单纯地进行模拟。

参数优化

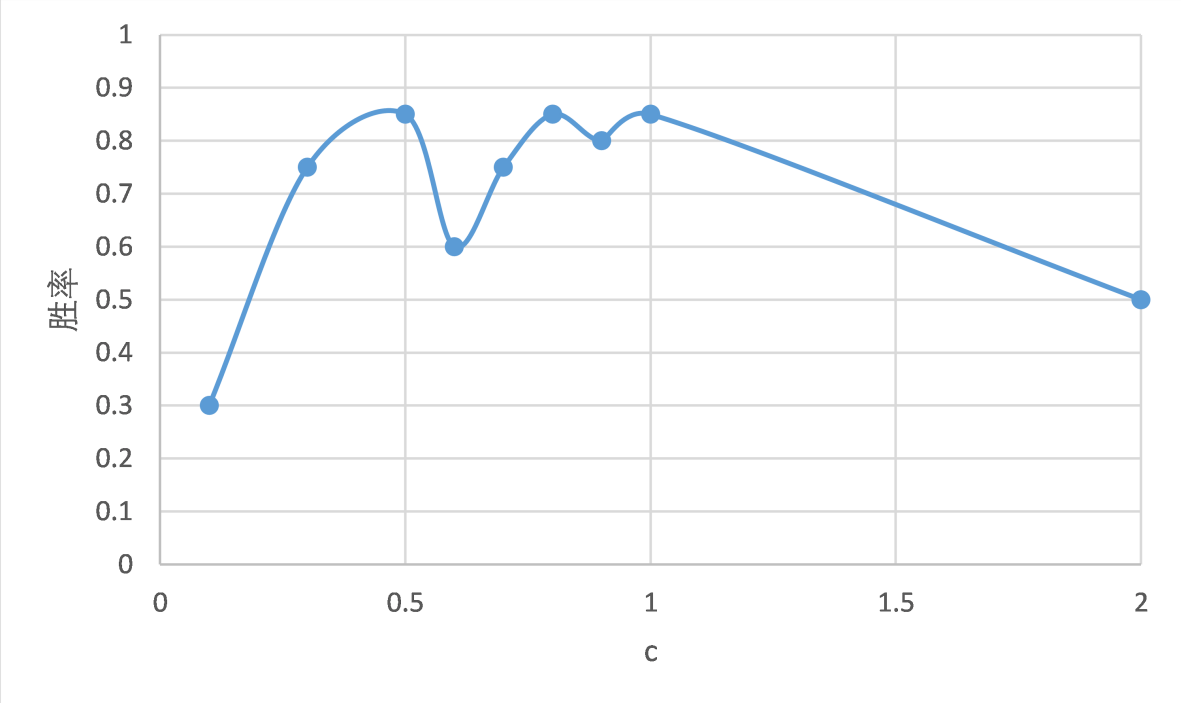
固定$c = 0.5$，最终落子选择**探索次数最多**的节点，与天梯最强AI94号AI对决20回合结果如下 (本地测试)：

```
Stat:
ratio of A wins : 0.75
ratio of B wins : 0.25
ratio of Tie : 0

ratio of (A wins + tie) : 0.75
ratio of (B wins + tie) : 0.25
```

固定$c = 0.5$，最终落子选择**胜率最大**的节点，与天梯最强AI94号AI对决20回合结果如下 (本地测试)：

```
Stat:
ratio of A wins : 0.85
ratio of B wins : 0.15
ratio of Tie : 0

ratio of (A wins + tie) : 0.85
ratio of (B wins + tie) : 0.15
```

固定最终落子选择**胜率最大**的节点，下面取不同的c值与天梯最强AI94号AI对决20回合结果如下 (本地测试)：



由如上参数调优结果，最后选择$c = 0.5$，**胜率最大**策略。

测试结果

在本机上进行测试，输出搜索次数，每次决策的平均搜索次数为**36w**次。 (除去对局开头和结尾的特殊情况)

利用网站的批量测试平行测试五次，得到如下的统计结果

批量测试编号	胜数	负数	平数	胜率 (%)
#10835	98	2	0	98
#10686	98	2	0	98
#10650	99	1	0	99
#10909	98	2	0	98
#11090	97	3	0	97

改进MCTS的批量测试平均胜率为**98.0%**。经过改进，算法的平均搜索次数增加三倍，批量测试平均胜率也有明显改善。

启发式模拟MCTS

启发式模拟MCTS是基于改进MCTS模型进行优化的模型。

优化思路

在之前的模型中，模拟策略都是随机在一列进行落子，这显然与真实的棋手对弈情况不太相符。为了让模拟更加贴近真实的对局，有必要定义一种新的启发式方法进行模拟，取代之前的随机模拟。

首先定义启发函数，用于评价在某一列 (第x列)落子的效用：

$$f(x, player) = g(x, white) \times \Big(2^3 \times \sigma(player, black) + 2^4 \times \sigma(player, white)\Big) + g(x, black) \times \Big(2^3 \times \sigma(player, white) + 2^4 \times \sigma(player, black)\Big) + \eta$$

其中：

- $\sigma(player, color)$的定义：

$$\sigma(player, color) = \begin{cases} 1 & \text{if } player = color \\ 0 & \text{otherwise} \end{cases}$$

- $g(x, color)$是在第x列落颜色为$color$的子所得的效用。
- η是一个均匀分布的随机数。加这项的目的是给模拟加入一点随机性，不至于完全按照启发函数进行决策。

对于每一次模拟，总是倾向于在$f(x, player)$值较大的位置落子，其中$player$指当前落子的一方。

如何理解f函数的两项的呢？假设当前玩家是白方，对于每一列，我们会尝试在这个位置进行两次落子。

1. 首先落白子，如果这时的效用大，当然表明这个位置更有利。
2. 接着落黑子，如果这时黑子的效用大，也表明我们需要尽可能在这个位置落子。为什么呢？因为如果白方不下这个位置，就会被黑方抢占，进而为黑方带来效用。由于四子棋是零和博弈，因此白方也需要尽可能不让黑方获得效用。

f函数的两项分别对应1. 和2. 两点考虑。当然，我们赋给1. 较大的权重 (2^4，使用2的幂是为了便于位运算实现)，原因是下棋一方总是会优先考虑自己的效用。

以上是启发函数的大体框架，接下来我们详细看一下如何计算$g(x, color)$。

$$g(x, color) = \sum_{1}^{4} g_i(x, color)$$

$$g_i(x, color) = 2^{n+1} \times m$$

- $g_1(x, color)$表示第x列在落下颜色为$color$的子后顶端对应的行的效用值。依次列推，$g_2(x, color)$表示列，$g_3(x, color)$表示左上右下斜列，$g_4(x, color)$表示右上左下斜列。
- n表示这一行/列/左上右下斜列/右上左下斜列对应刚刚落下的颜色为$color$的子的位置有几个子连起来。m表示连起来的这一串子两头有几个位置是空的 (即没有对方的子占据且没有被棋盘边界限制)。

  例如：

  ```
  b
  b
  ```

  对于上述局面，$g_2(0, black) = 2^{3+1} \times 1$。

  ```
  b w
  b w w
  b w w b
  ```

  对于上述局面，$g_3(0, white) = 2^{3+1} \times 0$。

参数优化

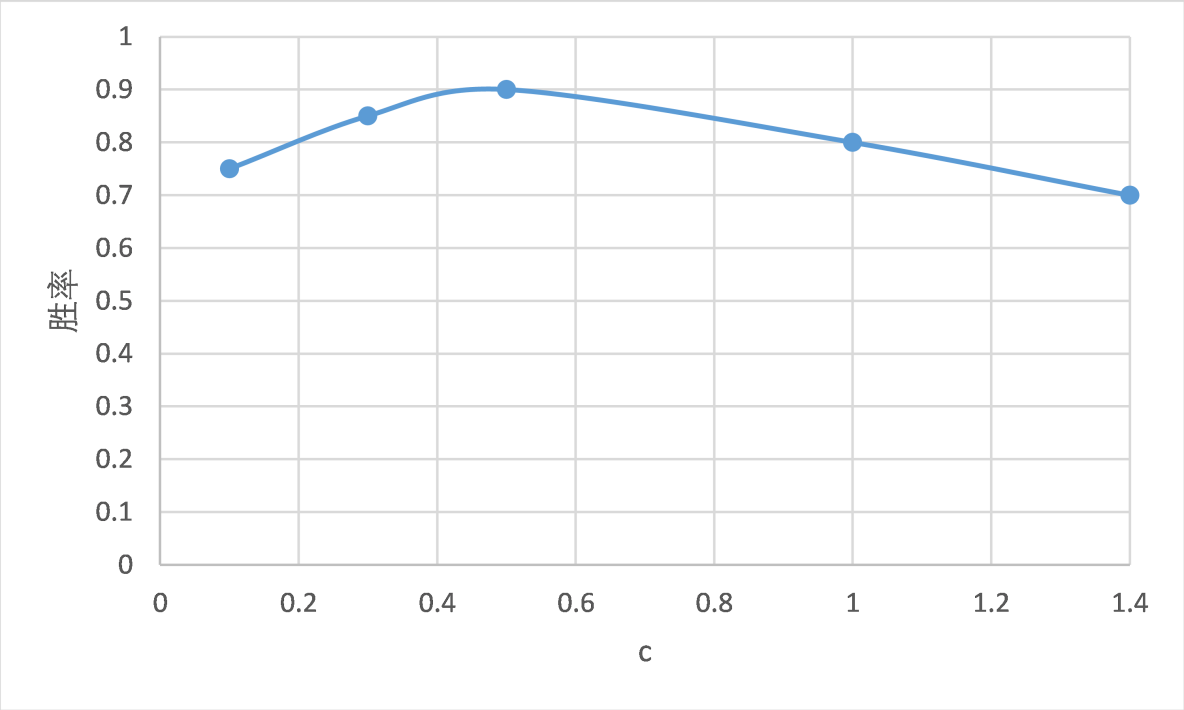
固定$c = 0.5$，最终落子选择**胜率最大**的节点，与天梯最强AI94号AI对决20回合结果如下 (本地测试)：

```
Stat:
ratio of A wins : 0.9
ratio of B wins : 0.1
ratio of Tie : 0

ratio of (A wins + tie) : 0.9
ratio of (B wins + tie) : 0.1
```

固定$c = 0.5$，最终落子选择**探索次数最多**的节点，与天梯最强AI94号AI对决20回合结果如下 (本地测试)：

```
Stat:
ratio of A wins : 0.75
ratio of B wins : 0.25
ratio of Tie : 0

ratio of (A wins + tie) : 0.75
ratio of (B wins + tie) : 0.25
```

固定最终落子选择**胜率最大**的节点，下面取不同的c值与天梯最强AI94号AI对决20回合结果如下 (本地测试)：



由如上参数调优结果，最后选择$c = 0.5$，**胜率最大**策略。

测试结果

在本机上进行测试，输出搜索次数，每次决策的平均搜索次数为**23w**次。(除去对局开头和结尾的特殊情况)

利用网站的批量测试平行测试五次，得到如下的统计结果

批量测试编号	胜数	负数	平数	胜率 (%)
#11307	93	7	0	93
#13057	90	10	0	90
#13058	85	15	0	85
#13059	95	5	0	95
#13247	91	9	0	91

启发式模拟MCTS的批量测试平均胜率为**90.8%**。

可以发现，由于需要计算启发函数，平均搜索次数有所下降。另外，启发函数给的启发信息可能不太准，导致平均胜率相比改进MCTS下降较为明显。

常数优化MCTS

通过启发式模拟MCTS的测试结果发现，启发式模拟可能不适合该问题，或者该问题需要定义另一种形式的启发函数。遂尝试从常数优化的角度去优化MCTS模型。

常数优化MCTS是基于改进MCTS模型进行优化的模型。

优化思路

优化MCTS有两个思路，一个是增加模拟的准确度，即在随机模拟的阶段加入更多的启发信息。另一个是增加模拟的次数，也就是通过常数优化加速搜索的过程。常数优化MCTS就是通过第二种途径优化改进MCTS模型。

在常数优化前，首先分析一些程序各个函数的开销，集中火力优化那些开销大的函数往往能事半功倍。利用`valgrind`分析程序各个函数的开销 (执行指令数)如下：

函数名	Instruction Read (IR) event counts
MyBoard::isEnd(int)	158,538,527
__memset_avx2_unaligned_erms	148,000,017
MyBoard::remove(int, int)	101,053,016
MyBoard::initBoard(MyBoard const*)	53,648,168
MCTS::checkWinPlaceExplore(int const&, int const&)	49,911,225
MyBoard::isFull(int)	7,320,456
MCTS::run(int const&, int const&)	2,815,505
...	...

下面是常数优化用到的方法：

- 对于一些经常使用的变量，可以加上`register`使之放入寄存器。加速对这些变量的访问。

  此外，对于经常需要调用的函数，也可以对其参数使用`register`，加速传参。

  ```
  int MyBoard::isEnd(const int x) {
      ...
  }
  ```

  可以改为

  ```
  int MyBoard::isEnd(const int register x) {
      ...
  }
  ```

- 对于经常调用的函数，且没有复杂的递归和循环，可以将其改为内联函数，减小调用的开销。

  ```
  int MyBoard::isEnd(const int register x) {
      ...
  }
  ```

  可以改为

  ```
  inline int MyBoard::isEnd(const int register x) {
      ...
  }
  ```

- 可以将递归函数写成非递归的形式，降低调用函数建栈的开销。不过经测试这一优化效果并不明显。
- 将比较简短的函数写成宏，减小调用的开销。

```
double MCTS::getUCT(const int &cur_situ, const int &x) {
    return (double)situs[cur_situ].num_win[x] / situs[cur_situ].num_expl[x]
+ sqrt(2 *     log(situs[cur_situ].tot_num_expl) /
situs[cur_situ].num_expl[x]);
}
```

可以改为

```
#define getUCT(cur_situ, x) \
    ((double)situs[cur_situ].num_win[x] / situs[cur_situ].num_expl[x] + \
        C * sqrt(2 * log(situs[cur_situ].tot_num_expl) / \
            situs[cur_situ].num_expl[x]))
```

- 将 `bool` 等速度比较慢的类型统一换成 `int` 类型。
- 改写数据结构，提高内存访问的局部性，便于最大化缓存的作用。

```
int next[MAX_SIZE][MAX_SITU];
int num_expl[MAX_SIZE][MAX_SITU];
int num_win[MAX_SIZE][MAX_SITU];
int tot_num_expl[MAX_SITU];
```

可以改为

```
struct Situation
{
    // next situations after moving
    // content: index in the global
    int next[MAX_SIZE];
    // exploring times of next situs
    int num_expl[MAX_SIZE];
    // winning times of next situs
    int num_win[MAX_SIZE];
    // exploring times of the situ
    int tot_num_expl;
};

Situation g_situs[MAX_SITU];
```

由于使得关系比较紧密的几个数组地址相邻，故在访问时能更好地命中缓存。

参数优化

由于本模型是直接在改进版MCTS基础进行常数优化的模型，没有进行任何的算法层面上的改动，故不再进行参数方面的测试，直接沿用其参数：$c = 0.5$，**胜率最大**策略。

测试结果

在本机上进行测试，输出搜索次数，每次决策的平均搜索次数为**61w**次。 (除去对局开头和结尾的特殊情况)

与天梯最强AI94号AI对决60回合，**平均胜率达90%。**

利用网站的批量测试平行测试五次，得到如下的统计结果

批量测试编号	胜数	负数	平数	胜率 (%)
#11892	100	0	0	100
#11897	100	0	0	100
#12071	98	2	0	98
#12077	100	0	0	100
#12473	100	0	0	100

常数优化MCTS的**批量测试平均胜率为99.6%。**

可以发现，经过常数优化后，平均搜索次数几乎翻了一番，批量测试的平均胜率也有了明显的提升。

利用网站的测试数据，统计该模型与不同段位的样例AI (如序号为(0,10]的AI)对决的平均胜率 (对决65回合)如下：

样例AI区间	平均胜率 (%)
(0,10]	100
(10,20]	100
(20,30]	100
(30,40]	100
(40,50]	100
(50,60]	100
(60,70]	98.5
(70,80]	100
(80,90]	95.4
(90,100]	81.5

从中可以发现：

- 该模型对决80以下的样例AI几乎都是全胜，说明模型具有比较好的鲁棒性，不容易在实力较弱的对手中爆冷。
- 对决(80,90]中的AI胜率达到95%，说明模型面对实力相对强劲的对手也能很好地发挥优势。
- 对决(90,100]中的AI胜率达到80%，说明模型在面对实例最强的AI时仍能保持较高的胜率。观察这些输掉的对局：大多数都是对方先手赢的对局，侧面说明四子棋先手仍然有比较大的优势。

模型性能比较

将上述各个模型经过调参后的最优结果总结为下表：

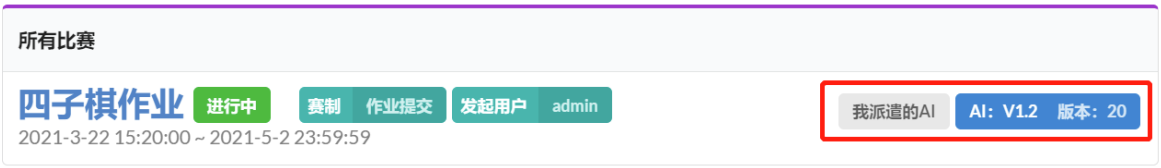
模型	批量测试平均胜率 (%)
朴素MCTS	**96.6**
改进MCTS	**98.0**
启发式模拟MCTS	**90.8**
常数优化MCTS	**99.6**

从上表可以看出目前最优秀的模型是**常数优化MCTS**。这也是最终提交的模型版本。

该模型对应版本`v20`

我的代码

目前已经提交到小组`作业提交`中。



改进方向

- 设计更好的启发函数

  本次实验中的启发式模拟MCTS模型效果并不好。猜想很有可能是启发函数定义不够好导致。但设置启发方法优化随机模拟过程的思路是可以延续的。可能的改进方向是阅读必胜策略的论文，尝试将必胜策略中的原则，如奇偶迫手等引入到启发方法中，优化随机模拟的准确率。

- 将神经网络与MCTS相结合。

  除了启发方法外，还可以仿照AlphaGo训练一个"快速走子网络"。具体来说，可以利用已有的模型直接互相pk生成局面，将局面编码成序列输入网络，以局面对应的落子位置作为标签训练网络。将这个用在模拟阶段，虽然可能会降低模拟次数，但会让模拟的结果更加准确。

- 进一步常数优化。

  从常数优化MCTS模型的测试结果中可以看出常数优化效果比较明显。未来可以尝试在缓存的层面做更多的优化，进一步加速MCTS搜索过程。

实验收获

通过这次实验，我实践了人工智能课上学到MCTS方法。通过自己编写程序，我对MCTS方法有了非常深刻的理解与感受。通过自己优化模型，我也更能体会到MCTS的优势、劣势在哪里，影响MCTS性能的关键是什么。

此外我通过自己的实践，包括**模型设计**，**编程实现**，**代码调试**，**参数优化**，**模型测试评估**，**模型改进**等，切身感受了设计一个四子棋AI的不容易。尤其是代码调试，非常考验工具选择和调试技巧。这些工作让我的实践能力，理论能力和创新能力都有了一定的提高。

参考文献

1. https://en.wikipedia.org/wiki/Monte_Carlo_tree_search