四子棋大作业.实验报告

2018011324

计82

尤艺霖

1.实验综述

根据提供的接口，选择合适的算法，实现一个四子棋对战AI。

2.算法选择与设计

算法的选择主要包括alpha-beta剪枝和蒙特卡洛树。在阅读了讲述小棋盘必胜策略的论文[1]后，我认为不依靠神经网络我无法自己实现高效的估价函数，故选择蒙特卡洛模拟，辅之以信心上限树。思路转向为写一个具有一定通用性的AI，期望从其中收获可以用于四子棋之外的知识。而在不针对四子棋进行算法层面优化的情况下，为了保证AI的棋力，选择的优化路线是优化每次计算的时间，访问尽可能多的状态，从而做出更合理的决策。

3.实现细节

由于我基本的思路是压缩每一轮的时间，获取更大的搜索空间，所以实现的细节就很重要。

3.1.整体流程

如果需求大量的状态数，那么就需要注意在线平台的内存限制，不能给每个状态都存储对应的棋盘。考虑在蒙特卡洛树上，每次运行都只会选取一条从根到叶子的路径，然后对叶子进行蒙特卡洛模拟，所以可以通过即时计算棋盘状态来省下内存：维护根节点的棋盘状态和当前节点的棋盘状态即可完成。

同时这样的写法使得可以在后一次运行中利用前一次运行的结果，因为下一次运算实际上直接对应了当前根节点的儿子的儿子，其他地方的节点都可以删除。为了简化删除的过程，我用了一个循环队列，预先开好两百万的状态，存储编号在队列里，新增节点就从队列里取出编号，删除节点就将编号加入队列。

3.2.胜负判断

重写了对于胜负的判断，利用以前做题的经验写出了如下代码。

```
while ((curboard[cx+c1+1][cy]&id))c1++;
while ((curboard[cx-c2-1][cy]&id))c2++;
while ((curboard[cx][cy+c3+1]&id))c3++;
while ((curboard[cx][cy-c4-1]&id))c4++;
while ((curboard[cx+c5+1][cy+c5+1]&id))c5++;
while ((curboard[cx-c6-1][cy-c6-1]&id))c6++;
while ((curboard[cx+c7+1][cy-c7-1]&id))c7++;
while ((curboard[cx-c8-1][cy+c8+1]&id))c8++;
if(c1+c2>2)return true;
if(c3+c4>2)return true;
if(c5+c6>2)return true;
if(c7+c8>2)return true;
```

八个循环分别统计八个方向上的最长的1，然后判断四种获胜的情况。

在做其他题目的时候，类似的写法可以提速很多。据说这样能更有效的并行代码，从而加速。

3.3.蒙特卡洛模拟细节

这实际上是一个向棋盘里放棋子的问题，为了能够高效的放入棋子，需要维护棋盘状态对应的**top**数组，这样可以进行O(1)的放棋子。同时为了O(1)地选择放棋子的位置，需要维护一个存储着尚有空位的列的数组，这个数组可以暴力维护，不影响均摊复杂度。由此可以做到高效地进行蒙特卡洛模拟。

3.4.信心值计算

信心值计算公式采用著名的**UCB1**，公式如下：

$$\frac{Q(V')}{N(V')}+c\sqrt{\frac{2ln(N(v))}{N(v')}}$$

在程序中，c取值为$\frac{1}{\sqrt{2}}$，这样可以抵消掉根号内的2，从而降低计算复杂度。

论文**Finite-time Analysis of the Multiarmed Bandit Problem**[2]中提出的c为1，但我认为它只针对单层，对于多层的蒙特卡洛树，应当更强调探索而非扩展，即强调前一项，更多地探索同一子树下的更多可能性。

同时这篇论文它也提供了一个**UCB1-tuned**的写法，据称表现优于**UCB1**，但考虑到其计算公式包含了更多的数学函数计算，同时还需要维护现有的方差，会降低效率，综合考虑未必更优，就没有采用。

3.5.其他优化

经过一定测试发现，在计算信心值的过程中，两个数学函数（**std::log**和**std::sqrt**）会消耗大量时间，故选择对其进行针对优化。对于自然对数这一部分，由于该函数参数取值永远在一个范围内，故选择在初始化一局棋的时候预先计算好所有需要的自然对数的数值，在需要的时候直接从数组调用；对于求平方根这部分，由于其参数为浮点数，没有特别好的优化方式，参考了网络上基于**magic number**的快速开根号算法，结果不理想：只用一次迭代，则精度不足，使用两次迭代，则时间效率优势不明显，故选择继续使用**std::sqrt**。优化完这一部分后，能计算的状态数提升了大约**30%**，是很有效的提升。

在这些测试中还发现了AI对于残局的处理很糟糕，偶尔会有主动让出胜利的情况，为了杜绝这一现象，在扩展过程中如果扩展到了胜利节点，就将他的父节点和他自己的Q值分别设为极小和极大值即可。

3.6.优化结果

对于内存的高效利用使得每一局中使用内存不会超过**200MB**；对时间的优化也使得它能够搜索大量的状态，在本地测试时每一步可以搜索到**120万**以上的状态，在线平台上测试，也能搜索到**40万**以上的状态。

3.Ex.对上交的文件的说明

随压缩包上交的文件和提交到测评网站上的文件是一致的，除了去除了一个**readme.md**，是在线平台的测试规则，上交到在线平台时是随着**sdk**文件夹一起复制进去的，现为了精简将其删除。同时由于是在线平台上测试的，没有加入c++生成dll对应的两段代码。

同时也随压缩包上传了对应文件在加入那两段代码之后编译出的dll。如果源文件无法编译，可以去测试平台上取下我最终版本的代码，也可以使用该dll进行测试。

4.测试结果

主要采取在线平台**(saiblo)**作为测试手段，进行了多次批量测试。

在线平台用户名为**youyl**，完善了个人资料，可以通过查看个人资料确认。

4.1.和所有样例的批量测试

总共进行了四次面对所有样例的测试，编号分别为：**392**，**812**，**1233**，**1663**。胜率如下：

编号	**392**	**812**	**1233**	**1663**
胜率	94%	96%	99%	93%

这样的结果可以说明，在对战样例AI的时候，胜率是足够的。

4.2.和较强样例的批量测试

总共进行了三次针对性测试，编号分别为：**401**，**827**，**1236**。

401号批量测试对战的是**90**、**86**、**92**、**94**、**100**五个样例，结果如下：

编号	**胜利**	**失败**	**胜率**
90	18	2	90%
86	18	2	90%
92	15	5	75%
94	11	9	55%
100	13	7	65%

827号批量测试与**100**号样例对战了100局，取得了61胜39负的成绩，胜率**61%**

1236号批量测试对战的是编号最大的五个样例，结果如下：

编号	**胜利**	**失败**	**胜率**
92	18	2	90%
94	16	4	80%
96	17	3	85%
98	16	4	80%
100	11	9	55%

综合多次测试的结果来看，本AI对战编号**100**以外的样例都具有压倒性优势，对战**100**号样例也可以有六成胜算。

4.3.和天梯第一名对战

使用了一次批量测试和天梯第一名的AI对战了十六局，获得了其中四局的胜利，说明在面对极强的对手时它也并非毫无还手之力，仍然能保有25%的胜率。批量测试编号为**1293**。

5.总结

本次实验我使用信心上限树结合蒙特卡洛模拟，写出了一个能够正常运行且具有一定强度的四子棋AI，并对其进行了高效的优化，内存和时间开销均为上乘水准，最终测试也得到了比较理想的结果。这一AI没有使用特别的估价函数，也没有使用任何基于人类经验的知识，具有一定的通用性。但也正是由于没有针对四子棋的估价函数设计，导致相比于使用神经网络等高水平估价函数的AI，他的棋力不足。但即使在和排行榜第一的AI对战时，也并非毫无还手之力。从这次实验中，我也对信心上限树和蒙特卡洛模拟有了全面的理解，并收获了不少调试的经验。

6.参考资料

[1]Victor A. A Knowledge-based Approach of Connect Four

[2]Peter A, Nicolo C, Paul F. Finite-time Analysis of the Multiarmed Bandit Problem