四子棋实验报告

张驰 2022010754 zhang-ch22@mails.tsinghua.edu.cn

一、最终对抗结果

最终版本的AI在与所有双数AI的对抗中，取得了96胜，4负，0平局的战绩。如图：

网址：https://www.saiblo.net/batch/62683/

多次测试胜率基本都在95~96%浮动，而且每一次败绩的AI对手都不一样。这说明此AI并没有被某个AI对手专门“克制”，失败是由一些随机性带来的结果。

二、算法主体原理

我使用蒙特卡洛树搜索（MCTS）算法来实现AI。该算法主体逻辑如下：

1. 当前状态为树的根节点，经过下一步行动之后产生的状态为原状态的子节点。
2. 每次进行搜索时，找到一个“最优”的叶节点，在这个节点进行蒙特卡洛模拟。
3. 将模拟结果回传，更新路径上的节点胜率信息。
4. 不断重复2和3，直到达到指定搜索时间。随后选择根节点胜率最大的子节点行动。

更具体地，我们为了确定如何选择叶节点进行模拟，我们定义如下的“信心上界”：$\mathrm{UCB}(v) = \frac{Q(v)}{N(v)} + C\sqrt{\frac{2\ln N(v')}{N(v)}}$。其中，$v'$ 为 v 的父节点，$N(v)$ 表示 v 的访问次数，$Q(v)$ 表示 v 的胜利次数。C 是一个常数。

在第二步搜索时，我们逐层递归地进行搜索。对每一个节点，我们进行如下的分类讨论：

1. 该节点所对应的状态已经游戏结束：此时我们直接返回这个节点。

2. 该节点有未被模拟的子节点：此时我们选择一个未被模拟的子节点。
3. 该节点所有子节点已经被模拟：此时我们找到 UCB 值最大的子节点，并递归地继续搜索。

UCB 值的选择兼顾了选择该节点的预期胜率与该节点的访问次数，也即兼顾“exploitation”和“exploration”。这样的话，我们就是在鼓励模型多对有利的情形搜索（这些情形也更可能出现在实际游戏中），但也会照顾到其他节点，以保证搜索的全面性，不漏掉那些可能需要足够搜索深度才呈现出的特殊局面。

选择好节点之后，我们在该节点进行蒙特卡洛模拟，随机地让双方落子，直到有一方胜利或平局。之后，将这一局的输赢信息通过树上路径回传到根节点。路径上所有节点的总访问次数（$N(v)$）增加1，胜利次数（$Q(v)$）根据模拟胜负和节点对应局面的落子方视情况增加1或不变。

算法主体逻辑如下图：

Algorithm 2 The UCT algorithm.

function UCTSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD($v_0, 0$))

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return v

function EXPAND(v)
  choose $a \in$ untried actions from $A(s(v))$
  add a new child v' to v
    with $s(v') = f(s(v), a)$
    and $a(v') = a$
  return v'

function BESTCHILD(v, c)

$$\textbf{return} \quad \underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$$

function DEFAULTPOLICY(s)
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v

三、算法优化与测试

我一开始对算法进行了不少优化的尝试。

3.1. 决胜步优化

一开始我尝试让AI在能够一招制胜时直接落子取胜，同时在敌人能够一招制胜时及时封堵。另外，我在落子前进行一次判断，如果我的落子将助攻敌人制胜，那么就改变一处落子。起初，我的搜索写的并不正确，这种优化帮助我将胜率从约30%提升到了70%左右。但后来随着我正确改正了搜索算法，我发现去除这些优化对我的算法胜率影响不大。

3.2. 胜率计算

我们的游戏并非只有胜负两种情况，而是有胜负平三种可能。我的算法在计算胜率时只考虑了“胜利”，而将平和负视作同一种情况考虑。

我尝试将平局也考虑在内，即胜负平三种情况分别记 $1, 0, -1$，但经测试这样并没有什么好的效果。事实上，平局发生的可能性非常小。在公式 $\mathrm{UCB}(v) = \frac{Q(v)}{N(v)} + C\sqrt{\frac{2\ln N(v')}{N(v)}}$ 中，上述改动只会使第一项 $w = \frac{Q(v)}{N(v)}$ 发生改动 $w' = 2w - 1$。如果我们取 $C' = 2C$，则改动基本不会影响子节点的 UCB 大小关系。

3.3. 空间优化

在我自己的AI的代码实现中，我在每一个节点里保留了一整张棋盘。这其实是不明智的，因为我们其实可以在整个树上只保留一张棋盘，然后再选择的时候逐渐在棋盘上落子，模拟完毕后再恢复，从而减少大量内存耗费和拷贝数据的时间。而事实上，由于内存限制很宽松，且如果使用静态数组存储的话时间耗费相对较少，我的代码目前表现良好，故没有进行进一步优化。

3.4. 智能模拟

目前算法在进行蒙特卡洛模拟的时候是完全随机的。我尝试过在模拟过程中也加入一些智能，具体来说，即加入3.1.中提到的决胜步优化，使得模拟本身也计算的是带有优化的胜率。经过一次批量测试发现：对所有双数AI的胜率仍为95%，未有显著提高。我对此的解释是：在大多数情况下，模拟的结果并没有显著偏向性，基本是一个随机结果。但对于一些即将制胜/负的局面来说，模拟则有很大概率直接取胜/失败，对于已经胜利/失败的局面，模拟结果更是确定的。这些情况对蒙特卡洛树的搜索提供了主要信息来源。

3.5. 模拟次数

目前算法在每个状态只进行了一次蒙特卡洛模拟。我曾试图在每个节点进行多次模拟，以提升概率精度。我尝试在每个节点进行十次模拟，但批量测试胜率下降到了88%。对此，我认为相比在一个节点获得更多的胜率信息，不如多拓展它的子节点来增加树的深度。在一个局面模拟十次，信息不如在他的十个子节点各模拟一次更有效（这样还包括了子节点的落子信息）。