四子棋 实验报告

Colin

May 24, 2021

1 背景与原理

四子棋对弈是满足双人、一人一步、双方信息完备、零和四个条件的博弈问题。该问题可以用基于模拟的搜索解决。

下面首先按照强化学习中相关内容的思路厘清求解方法的由来。

1.1 普通的基于模拟的搜索

从某一给定的状态（比如轮到己方的棋盘状态）开始，以该节点为根，不断枚举所有可能的动作，从而建立一棵搜索树。对该搜索树应用Markov决策算法求解，即可得到一条最优决策路径，从而得知在当前状态应该做何决策。

1.2 简单Monte Carlo搜索

显然，上小节的方法对于空间较大的问题（比如大多数棋类博弈）很难求解，因为计算代价不可估量。

因此，人们引入Monte Carlo方法，即统计模拟方法。该方法的原型为通过随机采样来进行数值计算，比如定积分。

简单Monte Carlo搜索可以被描述为：对于当前状态S_t和下一步所有可能的策略$a_t \in A_t$（A是策略集合），对于每个策略a_t进行K轮采样。用$G_{t,k}$表示每轮采样的回报。于是，选择每种策略的价值可以被表述为：

$$Q(S_t, a_t) = \frac{1}{K}\Sigma_{k=1}^{K} G_{t,k} \tag{1}$$

由此，最佳策略得以产生：

$$a_t = \arg\max_{a \in A_t} Q(S_t, a) \tag{2}$$

1.3 Monte Carlo树搜索

简单Monte Carlo搜索仍然要枚举每一种决策。此外，它没有保存模拟过程中的中间信息，而只得到了模拟的结果。因此，MCTS从这两方面进行了改进。

1.3.1 状态评估

MCTS直接对当前状态进行K次采样。第k次采样的状态序列可以被描述为：$\{S_t, S_{t+1}^k, S_{t+2}^k, \ldots, S_T^k\}$。

根据以上采样结果，建立一棵包含了已经访问过的状态及其决策信息的搜索树。然后，对每一种状态和策略的组合，都可以用下式来评估其价值：

$$Q(s,a) = \frac{1}{N(s,a)}\Sigma_{k=1}^{K}\Sigma_{u=t}^{T} 1((S_u, A_u) = (s,a)) G_{u,k} \tag{3}$$

其中“1”函数表示其内部等式成立时值为1，否则为0。该式的意义其实是从所有采样的结果中统计出某一种状态和策略组合的价值。

1.3.2 模拟搜索

MCTS算法除了状态评估之外，还包含模拟搜索部分。

模拟策略需要进行动作（决策）的选择，包含两部分：一是对于搜索树的内部节点（即已经访问过的节点），采取树内策略；该策略基于前述的状态评估方法。二是默认策略，即对于未访问的节点采取随机的动作选择策略（具体到后述的实现即为Rollout过程采取随机快速走子策略）。

在不断迭代的过程中，树内策略会不断被改善，最后使得搜索树收敛到最佳的搜索树。

树内策略不仅仅是选择评估值最大的状态。人们首先发明了$\epsilon-$贪心策略，即选取某一固定的值$\epsilon \in (0,1)$。面对策略集合A以及对应的$Q(s,a)$，每次有ϵ的概率选择状态评估值$Q(s,a)$最大的状态，而有$(1-\epsilon)$的概率选择剩下的其它状态。这样做是为了平衡搜索树中已经访问过并且有回报值的节点与未访问过（或者访问次数少）还需要进一步采样的节点。

而不同于选一固定值ϵ，本次实验中采用的UCT算法给出了一个动态调节的UCB公式：

$$UCB(v) = \frac{w}{n} + c\sqrt{\frac{\ln N}{n}} \tag{4}$$

其中v为一个搜索树中代表某种状态的节点，w为该节点上的回报值，n为该节点的访问次数，N为节点v的父节点的访问次数（即在父节点上进行决策的总次数），而c是一个需要调节的常数。不难看出，该式的第一项反映了该节点的历史回报水平，而后一项为根据历史中该节点被选次数来评估是否还需要对其采样的评估值。

2 实现细节

总地来说，本次实验采用的是MCTS算法。在实现中有以下细节。

2.1 反向传播时的参数更新

令Rollout过程中，己方赢的回报为1，对手赢的回报为-1，平局回报为0。

该回报值反向地沿当轮迭代的路径从被Rollout节点向根节点传播。具体来说，如果一个节点是极大节点（说明其父节点为极小节点），则该节点的值减去回报值；如果一个节点是极小节点，则该节点的值加上回报值。

选择上述反向传播的更新方法时，UCB评估算式对于所有节点都是一样的，不必区分极大与极小节点。（相反，若在反向传播过程中没有区分极大与极小节点，则UCB算式需要区分。）

2.2 信息存储

为了加快程序运行速度，提高迭代次数，在存储信息上使用一下方法：

1. 每轮迭代之前创建一个初始棋盘（自行实现的Board类，存储处于某种状态的棋盘），每轮迭代开始时复制一个初始棋盘。由于MCTS算法在搜索树中的路径是一直向下的，因此可以根据节点（Node类）存储的其相对于父节点多走的一步棋的信息，边向下搜索边更新棋盘。这样不用每个节点存储一个棋盘，占用大量空间。

2. 对于搜索树上的节点，如果采用C++中new的方式每次逐个创建，则会由于new操作的常数以及访问不连续的地址浪费较多时间。因此，在经过实验发现本次实验中节点个数不会超过200万后，在一开始以数组的形式一次性申请足够的空间并供所有轮迭代使用（每轮迭代之后计数器归零，从头复用）。经尝试，该优化能将性能提高到原来的约1.5倍。

3 实验与优化

3.1 最终策略选择

根节点的每个子节点对应了一种可能的走子策略。有研究[1]指出，此处的选择方式分为4种。除了大部分人使用的方式，即选择被访问次数最多的节点作为最终选择之外，还有一种为选择价值最大的节点作为最终选择。

然而，在测试时发现，仅仅将最后一步选择由选择被访问次数最多的节点改为选择价值最大的节点后，在Saiblo网站上使用50个样例AI进行评测正确率由90%以上下降到了50%左右。鉴于实现策略存在明显错误

的AI在Saiblo网站上的正确率仍为40%-50%（此现象为通过一些尝试发现），可知后一种选择方式是极其糟糕的。最终选择还是应该选择被访问次数最多的节点。无论是价值最大，还是UCB评估值最大，都是错误的。

3.2 随机模拟与策略

MCTS利用大量模拟（Rollout操作）支撑其算法的正确性。然而，为了使模拟中的“有效部分”更高，需要加入策略。

在Saiblo网站上的对比评测结果如下：

类型	第一回合模拟次数	胜局数	败局数	平局数	胜率
无策略	40~50万	88	12	0	88%
有策略	8~12万	99	1	0	99%

其中，无策略指在Rollout过程中的每一步，采用完全随机的方式走子。

而有策略包含的策略为：

1. 遍历所有可能的走子方式。若当前角色能在某处落子后获胜，则本次模拟立即以这种方式结束，并返回相应的结果。
2. 若当前角色没有能一步获胜的方法，则再次遍历所有可能的走子方式。若当前角色的对手能在某处落子后获胜，则该模拟回合当前角色立即抢占该位置，然后直接进入下一回合的模拟。

这两条最简单的策略其实带来了性能上的隐忧。每一模拟回合中的两轮遍历比随机走子多了一次棋盘宽度的遍历（随机走子也要先遍历一次得到所有可落子点），并且每次遍历中由于要尝试某处落子后某方是否能立即胜利，还要花费更多的时间去判断是否获胜。而上表中第一回合模拟次数的成倍减少正反映了这一问题。（之所以使用第一回合模拟次数，是因为随着棋局的发展，搜索空间会逐渐变小，单次迭代时长简短，迭代次数增加。第一回合的迭代是搜索空间最大而迭代次数最小的，更能反映迭代算法中的性能。）

然而，实验结果却“出乎意料”——迭代次数降为原来的1/5后的有策略组，胜率反而大幅提高，这似乎与MCTS算法由统计数据驱动正确性的原理不符。不过，经过分析可知，后者的优势其实是因为模拟的有效性大幅提高。假想宽度为10的棋盘，某一处落子后某方直接胜利。如果完全随机模拟，假设10个位置均可落子，则只有1/10的概率可以落到一步结束之处，即实际情况中任何理性决策体都会做出的决策；而其余90%的情况，在实际理性博弈中都不会出现，成为了无效模拟。而完全随机中的无效模拟是可以多步发生的，因此最终有效模拟的占比甚至不到10%！

由此可知，策略的存在牺牲了部分迭代次数，但提高了大量模拟中有效模拟（即贴近现实理性情况的模拟）次数的占比。因此，给定一定的时间资源，无法进行超大量模拟的前提下，把握好策略的复杂度，平衡迭代次数、模拟次数与有效率的trade-off，是优化MCTS算法的关键。

本次实验的最终程序仅使用了上述两条最简单的策略。

此外，上述策略仅仅加在了Rollout模拟过程中，而未加入到其它决策过程中。虽然在迭代开始之前就做上述两条判断可以免于大量迭代而极快地给出当时回合的落子点，但是这是不必要的。因为MCTS的正确性保证了其本身一定能给出应对这种特殊情况的正确答案。

3.3 其它技巧

网站https://www.xqbase.com/computer.htm上介绍了位棋盘的数据结构，可以用于加速胜负的判断，从而进一步提高迭代次数。不过由于时间精力所限，本次实验中没有实现。

4 小结

本次四子棋实验中，我学习并实现了MCTS算法，并在关于“随机模拟与策略”问题的实验和分析中加深了对该算法的理解，同时也练习了寻找问题并优化矛盾的思维方式。

References

[1] Guillaume Chaslot, Mark Winands, H. Herik, Jos Uiterwijk, and Bruno Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 04:343–357, 11 2008.