Value Function Guided Monte Carlo Tree Search for Connect Four Game

Zhengxiao Du[1] 2016011014

Department of Computer Science and Technology, Tsinghua University, China
duzx16@mails.tsinghua.edu.cn

Keywords: Artificial Intelligence, Monte Carlo methods

1 Problem Definition

对传统的重力四子棋 (Connect Four) 游戏的规则做出一定的修改，主要是棋盘的长宽在 [9,12] 之间随机选择，同时在棋盘上会随机生成一个不可落子点。针对这一游戏设计 AI。

2 Approach

2.1 Monte Carlo Tree Search

四子棋游戏是一个简单的、双方的、完全信息的博弈，所以可能的算法主要是 Alpha-Beta 剪枝和蒙特卡洛树搜索 (MCTS)。之所以选择 MCTS，是因为 Alpha-Beta 剪枝因为一般搜索的层数不足以到达游戏结束，所以需要一个估值函数 (Value Function) 来对到达的局面进行评估。而蒙特卡洛树搜索是依赖大量的随机模拟来对局面进行评估。所以蒙特卡洛树搜索不需要人类对于游戏的先验知识参与，更加符合我们对于“人工智能”的期待（其实说白了就是不想去设计复杂的估值函数）

2.2 Upper Confidence Bound applied to trees

在蒙特卡洛树搜索中，一个很重要的步骤就是从当前节点的子节点中选择 BESTCHILD，这决定了 TREEPOLICY 中会扩展什么样的节点和选择什么样的状态进行随机模拟。最简单的方法是选择平均胜率最高的子节点。但是对于那些没有被充分访问过的节点，当前的平均胜率是不能代表实际的

Algorithm 1 蒙特卡洛树搜索

Input: 当前的状态 s_0

Output: output 采取的行动 a

1: 以当前状态 s_0 创建根节点 v_0
2: **while** 尚未用完计算时长 **do**
3: $v_l \leftarrow \text{TREEPOLICY}(v_0)$
4: $\triangle \leftarrow \text{DEFAULTPOLICY}(s(v_l))$
5: $\text{BACKUP}(v_l, \triangle)$
6: **return** $a(\text{BESTCHILD}(v_0))$

其中, TREEPOLICY 表示从当前节点出发不断进入 BESTCHILD, 直到遇到叶子节点。如果这个节点可扩展，则扩展一个它的孩子并返回，否则直接返回该节点。DEFAULTPOLICY 表示从给定状态开始随机模拟并返回结果

局面优劣。如果有一个事实上的最优局面因为没有被充分访问，当前的平均胜率低于一个次优局面，那么它就永远也没有被访问的机会。

为了克服这一问题, 我们定义了一个节点的信心上界 (Upper Confidence Bound)

$$\text{UCB}(v_i) = \frac{\omega_i}{n_i} + c\sqrt{\frac{\log N}{n_i}} \tag{1}$$

其中 ω_i 表示的是 v_i 进行的随机模拟中获胜的次数，n_i 表示的是 v_i 已经进行的随机模拟次数，N 表示的是总共进行随机模拟的次数。

公式中的前面一项表示的是该节点的平均胜率，而后一项表示的是该节点胜率的不确定度，也就是该节点胜率的区间估计的上界。这样对于那些没有被充分访问的节点，由于后一项较大所以整体的信心上界也较大，就有可能被访问到。

2.3 Value Function Guided Policy

蒙特卡洛树搜索中另一个重要的步骤是从给定状态出发，进行随机模拟直到分出胜负。通常的随机模拟方法是双方轮流随机选择落子点。在模拟次数足够多的情况下，得到的胜率可以近似反映当前局面的好坏。但是这种随机模拟并不能真正代表双方可能的决策。尤其是和高水平的对手对弈的时候，对方很可能能准确找出对我方最不利的落子，而随机模拟所得到的大部分情况下较优的局面可能会被对方找到巨大的破绽。

AlphaGo 解决这一问题的主要方法是通过训练估值网络和策略网络。一方面通过策略网络来缩小搜索的范围，一方面通过估值网络来改善信心上界的计算。同时通过策略网络来指导随机模拟。这一系列方法完全是依赖了神经网络强大的表示能力和较高的运行效率（比起基于随机模拟和搜索的方法）。

但是对于这个问题来说神经网络显然是没法应用的。AlphaGo 是基于人类棋谱训练的。AlphaGo Zero 虽然可以从 0 开始训练，但是 Google 在 TPU 上都训练了 40 多天。

好在通过估值函数来指导随机模拟这个想法仍然是可以应用的。具体来说，就是我们在随机选择下一步的落子时，不是完全随机抽取，而是从一个概率分布 P 中抽取，P 为

$$P(a_0|s) = \frac{f(a_0|s)}{\sum_{a_0 \in A} f(a|s)} \tag{2}$$

其中 $f(a_0|s)$ 为在局面 s 下，落子点 a_0 的估值。

在实现的时候，只需要采用类似于遗传算法决定哪些个体进入下一代的“模拟轮盘赌”算法就可以。

这里的估值函数主要考虑的是简洁性，不能过多地拖慢模拟的速度。而且太过复杂的估值函数设计也有违选择蒙特卡洛树搜索法的初衷，具体的估值函数可以参见代码。

2.4 Other Optimization

此外需要考虑的就是如何让程序在限定的时间内尽可能多的进行模拟。为了安全起见，这里将整个搜索的时间限定在了 2.5 秒。

内存分配 MCTS 从某种程度上是一个用空间换时间的算法，需要用到大量的空间来存储树的节点。这种动态内存分配很容易占据大量的时间。所以单独设置了一个节点池，每次只需要从中划出一个节点来。而且这个节点池可以作为全局变量，只需要在程序开始的时候分配，在结束的时候释放。

计算优化 在信心上界的公式中我们可以发现 $\log(N)$ 这一部分在一次搜索中，对于所有的节点来说是相同的，所以可以把其数值作为全局变量存储起来，只在一次搜索结束 BACKUP 的时候更新。

搜索剪枝 虽然 MCTS 强调的是随机模拟，但是实际上我们依然可以通过一定的剪枝来减少搜索空间，从而实现对那些有价值的节点进行更多的更多的随机模拟。为了避免剪掉可能的最优解，这里采用了最保守的剪枝策略，对于一个节点的可能的后续落子点，如果我方下该落子点必胜的话，那么不会考虑其它落子点。如果没有这样的落子点，而又有某个落子点对方下的话必胜的话，那么也不会考虑其他落子点（就是先考虑必胜，再考虑必败）。这样可以大大减少某些分支的深度，将时间花在更需要探索的分支。

决策标准 观察一局对战中不同落子点的信心上界可以发现，很多时候不同落子点的信心上界差距是很小的。在最终决定最佳落子点的时候，为了防止出现“抖动”，将选择标准从进行搜索时的信心上界改成了模拟次数。这和老师上课讲的 AlphaGo 的策略是一致的，实测发现不同落子点的模拟次数差距还是相当大的，最小的时候也差了几万次模拟，可以避免因为最后几局的结果而影响最终的决策。

3 Experiment

3.1 How UCT can help

虽然 UCT 有非常完善的理论框架，但是在实际对战中 UCT 比起普通的 MCTS 真的能有胜率的提升吗，如果有的话又有多大的提升，为了回答这一问题，我们用 UCT 和 MCTS 各自和 100.dylib 进行了 100 局对战，然后统计了结果。可以看到提升还是非常非常显著的。单纯的 MCTS 算法一旦有某一个子节点的第一次模拟对局是负的话，而恰好又有另一个子节点的某一局模拟对局是胜，那么前一个子节点永远都不会被访问到。在这种情况下 MCTS 无法有好的表现也是可以预料的。

表 1. Winning rate of MCTS and UCT

Method	Rate(%)
MCTS	25.0
UCT-Value	**98.0**

3.2 How value function can help

为了衡量估值函数对于 AI 的影响，我们用采用估值函数策略的 UCT 和采用随机策略的 UCT 分别与 100.dylib 进行了 100 局对战，然后统计了结果。

表 2. Winning rate of MCTS and UCT

Method	Rate(%)
UCT-Random	91.0
UCT-Value	**98.0**

我认为估值函数对于 UCT 的帮助主要是两方面的。一是因为估值函数指导下的策略会更加倾向于走那些已经有比较多连子的区域，所以可以更快结束对局从而进行更多的模拟对局（当然因为估值函数和模拟轮盘赌的运行时间，这种提升并不明显）。二是估值函数指导下的策略更接近人以及有一定智能的 AI 的走法，所以这种模拟要更有意义。

3.3 Simulation Test

为了得到最终 AI 的真实水平，这里模仿要求中给出的测评方法（即和 50 个 AI 各对战 2 局，交换先后手）的方法进行了多次模拟测评，结果如下：

表 3. Result of Simulation Test

Num	Win	Tie	Lose	Rate
100	99	0	1	99%
100	100	0	0	100%
100	98	0	2	98%

可以看出此时的 AI 已经有比较好的性能，按照测评方法几乎是不败的。

3.4 Case Analysis

通过分析我的 AI 与样例 AI 对局时候的胜负可以发现，我的 AI 失败的局面都是很后期的局面（就是有一半的列都已经被填满的局面）。

我猜测样例 AI 应该是基于 alpha-beta 剪枝的（因为样例 AI 出解很快，不大可能是 MCTS）。分析背后的原因可能是基于蒙特卡洛的方法因为靠的是绝对的胜负作为选取落子点的标准，因此非常擅长找出在当前局面下如何走出必胜或者接近必胜的策略。在大量的对局中，样例 AI 在棋盘落子数目小于 50% 的情况下游戏结束的话几乎必定会输掉。而 alpha-beta 剪枝靠的是估值函数来选取落子点，如果给它足够长的落子数目的话，产生的局面必然是有很多“三连子”的攻击点，这种情况下后期不管采取什么策略都是很难获胜的。

当然并不是说 UCT-Value 就不擅长处理后期局面。如果不是局面非常有利于对方，因为后期局面模拟对局结束得都很快，所以 UCT-value 能进行大量的模拟对局从而找出最有利的策略。在后期局面中 UCT-Value 的胜率还是占优的，只是不能打出超过 90% 的胜率而已。

4 Conclusion

在一开始写这次大作业的时候我以为 MCTS 是要优于 alpha-beta 剪枝的，因为它和 UCT 结合起来，理论框架非常完整，听起来也非常能发挥计算机的优势。但是实际写下来，以及和样例 AI 对局之后，我发现 alpha-beta 剪枝还是有很多优势的。前面说的后期优势是一个。另一个是 alpha-beta 剪枝的出解很快，几乎肯定不会超时（不过在我们这个规则下这又添加了一个劣势：不能像 MCTS 那样最大化利用时间而且保证不会超时），而且内存占用也没有 MCTS 那么大。只不过 alpha-beta 剪枝依赖于人在特定领域的经验来设计估值函数，我连五子棋都没怎么下过，想来也设计不出什么好的估值函数。另外随着计算机计算能力的提升，MCTS 相对于 alpha-beta 肯定是优势越来越明显的（感觉有点像在机器学习领域，因为算力的提升，深度学习就明显超过了基于手工设计特征 + 简单分类器的方法）。

另外，MCTS 的一个关键是如何进行更多的随机模拟。我想的主要还是从算法角度去优化，而没有考虑过实现层面。比如看到有同学在群里说可以用 OpenMP 写多线程，确实是有“这么好的想法我怎么想不到”的感觉。不过想到助教的电脑未必是多核的，多线程未必能真正的加速，而且还容易出 bug，那就没法挽回了，所以没有去做。