四子棋实验报告

问题描述

编写一个重力四子棋的AI。本次作业中重力四子棋的基本规则如下：

1. 棋盘为宽为N，高为M的矩形棋盘，其中$M, N \in [9, 12]$；
2. 棋盘上有一个随机的禁止着点，这个点上不能落子；
3. 每次着子都必须落到最下方空的格点上；
4. 获胜条件为一方的棋子在横、竖、斜任意方向上能够连成四子，则该方获胜；如果棋盘上所有棋子落满时仍没有任意一方达到获胜条件则视为平局；

算法

算法的选择

主要的解决方案有α-β剪枝和蒙特卡洛方法。α-β剪枝算法简洁，但问题在于需要一个比较好的局面评估函数。四子棋的步数比较多，不能指望每次都一直探索到根节点，需要对搜索深度进行控制，就需要局面评估函数。而蒙特卡洛方法则几乎不需要任何四子棋的知识，它通

过模拟结果来评估各个节点的优劣。蒙特卡洛方法我认为是适用面更广的选择，因为不需要评估函数，我缺乏四子棋的知识，于是选择蒙特卡洛方法。信心上界树算法（UCT）是在蒙特卡洛方法的基础上对扩展节点的选择进行了改进。

棋局的表示

首先要回答一个问题：用什么来表示棋局的进行?

用节点来代表一个局面，它不会存储关于棋盘的任何信息，但两个不同的节点代表的棋局一定不相同（这里的棋局不是说棋盘的情况，而是包含着行棋的顺序，前者是有可能相同的）。它离根节点的距离暗含着是哪一方棋手的信息，比如说根节点代表此方将行棋，则它的子节点就表示对方将行棋。它每一个子节点是一个新的局面（由当前棋手落了一个子）。

算法的主要步骤

蒙特卡洛方法的主要步骤是：

1. 选择一个子节点：在所有可行的落子中，选择一个最有希望的节点，以供下一步进行扩展，这个最有希望节点的选择就是UCT的改进之处：

$$选择的子节点 = argmax(\frac{子节点赢次数}{子节点总次数} + c\sqrt{\frac{\log(父节点总次数)}{子节点次数}})$$

（但实际上有调整第一项）

c是一个可选参数，如果c大，表示我们更看重探索，比较冒险，小的话则倾向稳定。

2. 扩展这个子节点，即找一个从来没有以作为模拟开始的节点。
3. 对2得到的节点模拟。
4. 回溯更新：沿着模拟的起始节点，一直到根节点，更新评估。

   这里有一个易错点，回溯的时候，相邻两代节点代表的棋手是不一样的，如果子节点局面赢了，那么父节点就输了，所以要取相反数。

最后选择根节点最好的节点，也就选择了一个最好的走法。

实现

把整个算法封装为了UCT类，在getPoint中调用UCTroutine，`treePolicy`负责1、2，也就是找出一个可扩展的节点。`defaultPolicy`进行一次模拟，`backUp`回溯。

存在的问题

1. 如果有必胜着法，是否能保证下出这一步？以及如果有必输的走法，是否能保证下出这一步?

   原来的算法是不能保证这一点的，但是模拟次数越多，就越有可能下出。原因在于：原算法是从概率的角度来考虑的，它事实上无法完全区分一个必胜节点（对于必输节点也如此）和其它子节点，只是由概率估计哪一个节点更可能胜一些。一个简单的例子是（但是实际中未必容易出现）：除了必胜节点A，由于模拟次数少，另一个节点B的表现也不错，胜率也是1（模拟次数很少，偶然性大），那么算法事实上并不能区分必胜节点和此节点。

   同样，对必输节点也是如此。

2. 参数c如何选择?

   最重要的原则是，c的增大带来的是计算量的增加。如果时间更长，可以模拟的次数更多，那么c应更大。

   围棋和四子棋，c哪一个更大?

   直观感觉上是围棋更大。因为局面的不确定性更大，但实际上我认为并不是这样，这还是取决于计算资源的，只是潜意识中围棋有更多的计算资源。

   这是定性上的讨论，这对于此问题的解决并没有什么帮助。但由于3秒的时间限制，我们的c不会很大。

3. getPoint给出解法，以什么评价函数更合理？需不需要加上探索项$c_1\sqrt{\frac{\log(\text{父节点总次数})}{\text{子节点次数}}}$？

   不加上探索项实际上就是c为0的情况。这也是需要用实验回答的问题。但是定性来看，c_1会更小，因为需要更稳定的结果。但是否是c_1就应该为0，我认为选择扩展节点`selectChild`和`bestSolution`这两个函数的目的没有本质区别：都是为了找到概率最大的着法，`selectChild`找的是对方（并不是getPoint意义上的对方，而是相对于`cur_player`的对方）落子概率最大的点扩展，而`bestSolution`找的是己方概率最大的落子点。由于这样的分析，我认为蒙特卡洛方法基于的一个前提是，双方都是理智的，会下出最有利的结果（也就是胜率最大的结果）。

4. 如果3的分析是正确的，这个算法前提是两方棋手都是理智的，那么如果这个算法跟一个随机下可行解的算法（完全不具有任何智能）对弈，是否会影响这个算法的性能？是否有必胜的把握（而不是高胜率，毕竟随机下法胜率本就很低）？

   按照3的分析，应该是会影响算法性能的，我希望用足够多的实验来回答这个问题。

5. 每一次模拟，都需要先找出扩展节点，如果搜索次数已经够多，得到这样的一个待模拟节点也需要一定计算量，那么为了节省这个时间，是否有必要在模拟函数中多模拟几次，以期更可靠的结果?

   无论是寻找可扩展的节点该是模拟需要的计算量并不大，不会超过棋局剩余的空处个数。

   这样做的优点是，一次的`treePolicy`可以得到好几次的模拟结果，看上去节省了时间，但这样牺牲的就是更多扩展的机会，实际上这与c的选择问题有类似之处，也是一个需要用实验回答的问题。

改进

1. 在参考了网上的一些资料后，发现了一种剪枝策略，认为是一个非常好的做法，现对此分析。

   其做法就是：在寻找扩展子节点的函数中，如果发现一个子节点局面结束，就不再扩展其它走法。

   这样做的合理之处：据前面的分析，算法主要依据是概率，如果有必胜着法，概率最大(为1)，那么就应该这样行棋，对于其它处模拟得到概率就是多余的。

   选用这种方法能否保证下出必胜解以及避免必败解?

必胜解显然能下出，因为剪枝，`treePolicy`不会再扩展必胜子节点以外的节点，最后选择的一定是必胜解。对于必败解，不能保证百分之百不会下出（这是无法避免的，因为对于唯一一个不败的解，模拟也有可能会出现胜率为0的情况），概率是接近1的（这仍与唯一一个不败节点本来获胜的概率和模拟次数有关，在概率不是太小，模拟次数够多的情况下，几乎是100%），这是由评价子节点扩展的函数的性质决定的，如果是必败节点，在剪枝处理后，评价值增大的唯一可能性是第二项增大，但是由于log以及根号函数增速太慢，几乎没有什么影响，而由于下面一点的改进，减小的速度是很快的。

这样做的另一个优点是节省了一些时间，可用于更多更有价值的探索。

2. 和1相同的参考中，对于决定局面胜负的节点，返回的值不是0，而是一个更大的值，相应其父节点也会在回溯中减去更多的分数，相当于一个惩罚。认为这样也是有道理的，必败解/必胜解的父局面大概率也是比较坏/好的局面。

结果

每一个对局测试8局。对于除了100.dll，胜率都大于所给ddl，但对于100.dll，8局的对战中，一共赢了3局。多数有1的胜率，但对部分级别比较低的，比如22.dll，也有输的时候。

更多值得尝试的地方

1. 不同限时下参数c的选择。
2. 看有关选择函数合理性的文章。

参考

1. 本次与姚洋同学有过不少的讨论，比如问题5就是姚洋同学的想法。
2. 参考博客及思路：

   https://www.cnblogs.com/yifdu25/p/8303462.html

   https://towardsdatascience.com/monte-carlo-tree-search-158a917a8baa

   https://github.com/zhangchuheng123/Connect4