同化棋: 一种基于蒙特卡罗树搜索的解决方案

梁博强　元培学院

2022 年 1 月 6 日

摘要

在本学期计算概论课程的同化棋大作业中，笔者采用了一种基于蒙特卡罗树搜索（MCTS）的算法来实现并优化机器决策，并针对同化棋进行改进，最终取得了优于朴素贪心算法的成绩。本文将简要介绍这次大作业的设计思路、实现过程、决策优化、相关工作和后续进一步工作的方向。

通过本次大作业的实践，我对各种主流的博弈算法有了初步了解。此外笔者将 C++ 实现的决策算法封装在动态链接库中，并搭建了与 Python 编写的图形界面框架的连接，这加深了我对程序之间交互的理解。而在实现程序具体功能、提高程序实用性的过程中，笔者对文件读写、面向对象编程、树数据结构、异常处理等有了进一步认识。

0　背景

0.1　同化棋

同化棋（Ataxx），是 Dave Crummack 和 Craig Galley 在 1988 年发明，1990 年出品于电视游戏而流行的两人棋类，可说是黑白棋的衍生。其最主要的特点是落子后会将邻近八格的所有敌方棋子颜色翻转，并且原棋子有可能是被复制的，也有可能是被移动的。

同化棋的上述特点为机器决策带来了难度: 相比于相同规模的其他常见棋类，从当前局面状态到下一局面状态的**决策分支**可能更多；当前状态与下一状态间的**局面估值**（value）的差可能较大，这就要求我们在蒙特卡罗树搜索算法的基础上加以改进。

0.2　蒙特卡罗树搜索

1　不足与后续工作

必须承认，笔者完成这份作业较为仓促，因此仍有很多地方有待改进。

1.1 MCTS 探索节点数较少

尽管笔者尝试采用限制展开环节的深度，以及用叶节点的局面的估值（棋子数之差），但在规定时间内所能探索的节点数仍然较少，仅达到 1000 个左右。而同化棋中，每个状态节点对应的子节点约有 10-100 个，MCTS 要求我们完成展开的次数应明显高于每个节点的子节点数，才能实现较好的决策。

反映在结果中，我们可以看到本算法在对局开始（前 15 回合）和结尾（后 15 回合）时的决策明显优于中期的决策。原因可能是在对局前期，决策算法会倾向于先复制并占领更多的空白，或是试图在搜索深度限制内吃掉对方所有棋子。在对局末期，MCTS 几乎能枚举到所有最可能被打出的决策，从而给出精确的最优决策。但在对局中期，MCTS 很难模拟展开到终局，使得节点的估值很不精确。

探索节点数较少的原因可能在于: 首先，同化棋的决策分支数相对较多。更重要的是，采用贪心算法，而非随机决策进行展开模拟虽然可以获得比较好的模拟结果，但是对于每一个局面我们需要枚举所有可能的落子决策，这极大增加了算法的时间复杂度。

后续我们可以进一步优化不同回合时 MCTS 展开的深度限制，使得我们在“探索更多节点”与“模拟到更多终局”间取得一个平衡。另外，我们可以优化展开时的贪心算法，从所有移动方法为“复制”的落子决策中取得最优解；抑或是用叶节点的局面估值代替模拟展开结果，这样都能减少模拟展开时的时间复杂度。

1.2 搜索树未被长时保留

在 Botzone 平台上运行以及线下提交的版本中，笔者采用的都是常见的短时运行方式，即每一次决策结束后退出搜索函数，同时清空搜索树。这样做的稳定性较高，但每一次决策时都要重新构建搜索树。有趣的是，MCTS 最终选择的子节点是我们探索次数最多的节点，往往有最多已探索的后代节点，重新探索、模拟、展开这些后代节点是不必要的。

保留搜索树、采用长时运行模式并不难，但关键是决策后需要清理其他的分支节点，避免造成内存浪费。否则，经过数轮决策，搜索树的大小会显著超过 Botzone 上的内存限制。也有必要优化数据结构，减小单个节点占用的空间。