重力四子棋 实验报告

计 82 郑凯文 20181011314

2020 年 4 月 26 日

1 问题陈述

重力四子棋（Connect Four）是一种双人对弈、信息完全的回合制棋类游戏，具有 M 行 N 列的方形棋盘。游戏双方轮流选择一列，将棋子下在列的顶部，直到某一方首先具有四颗棋子落在同一行、同一列或同一斜线。

传统重力四子棋的棋盘是 6 行 7 列的固定大小，此种规则下已有先手必胜策略。本实验中增加游戏的不确定性，棋盘的行数和列数在 $[9, 12]$ 中随机选择，且在棋盘上随机生成一个不可落子点。针对这一修改版本设计 AI。

2 算法选用与优化策略

2.1 蒙特卡洛树搜索

蒙特卡洛树搜索（MCTS）和 Alpha-Beta 剪枝是此类对弈游戏的两种主要算法。由于时限 3s，Alpha-Beta 剪枝虽搜索较为全面，但不足以达到足够的层数，且需要人为设计启发函数，引入一些先验。这里采取 MCTS，在一定规则指导下进行大量随机模拟，从而对局面进行评估，可以充分利用算力。

2.2 信心上限树

MCTS 中一个重要步骤是从根节点出发不断选择 Bestchild，这决定了将要拓展什么结点。一种最简单的策略是选择截止当前胜率最高的结点，但这样将会导致算法对初始模拟敏感，在开始阶段胜率高的分支将会被不断深入，从而缺失了对更多可能性的探索。信心上限树（UCT）是对广度和深度的一种权衡，它定义了一个节点的信心上界

$$UCB(i) = \frac{score(i)}{num(i)} + c\sqrt{\frac{\log num(father(i))}{num(i)}} \tag{1}$$

进而选择信心上界最高的结点作为 Bestchild。式中前一项为胜率，后一项保证访问较少的结点有更高几率被访问到。其中 $score(i)$ 代表结点 i 的收益（胜为 1，平为 0，负为-1），$num(i)$ 代表访问次数。完整的 UCT 算法如下图。

算法 3：信心上限树算法（UCT）

function UCTSEARCH(s_0)
  以状态s_0创建根节点v_0;
  while 尚未用完计算时长 **do**:
    v_l ←TREEPOLICY(v_0);
    Δ←DEFAULTPOLICY(s(v_l));
    BACKUP(v_l, Δ);
  end while
  return a(BESTCHILD(v_0, 0));

function TREEPOLICY(v)
  while 节点v不是终止节点 **do**:
    if 节点v是可扩展的 **then**:
      return EXPAND(v)
    else:
      $v \leftarrow$ BESTCHILD(v, c)
  return v

function EXPAND(v)
  选择行动$a \in A(state(v))$中尚未选择过的行动
  向节点v添加子节点v'，使得$s(v')= f(s(v), a), a(v') = a$
  return v'

function BESTCHILD(v, c)
  return $argmax_{v' \in children\ of\ v}\left(\frac{Q(v')}{N(v')} + c\sqrt{\frac{2ln(N(v))}{N(v')}}\right)$

function DEFAULTPOLICY(s)
  while s不是终止状态 **do**:
    以等概率选择行动$a \in A(s)$
    $s \leftarrow f(s, a)$
  return 状态s的收益

function BACKUP(v, Δ)
  while $v \neq NULL$ **do**:
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta$
    $\Delta \leftarrow 1 - \Delta$
    $v \leftarrow v$的父节点

图 1: UCT 算法

2.3 MCTS 的改进手段

对于经典 MCTS 在 Treepolicy、Defaultpolicy、Bestchild 各个阶段的改进，已有大量研究成果，如在 Bestchild 选择时引入先验的 Progressive Bias，以及最著名的 Rapid Action Value

Estimation（Rave）。这些是较为普适的技巧，同时引入了更多的超参数。我对这些方法进行了尝试，发现收效甚微，且使得进一步改进的工作完全演变为了黑箱的调参工作。既然计算资源不足以支持 AlphaGo 的估值网络与策略网络这样的大改，与其寄希望于这些玄学调参，不如尝试针对性的、基于规则的优化方法。

2.4 针对性的优化与剪枝

2.4.1 概念定义

为方便表述，定义如下几个概念：

必胜手 当某方下在此位置时，将立即形成四连 $XXXX$，取得胜利。

次胜手 当某方下在此位置时，若对方不能马上胜利，则本方将取得胜利。这样的情况为“活三”$\_XXX\_$。

迫手 当某方下在此位置时，逼迫对方马上应对，否则本方胜利。大致分为两种情况：“冲三”$OXXX\_$、XX_X，以及此位置被占据后，其上的位置是必胜手。

2.4.2 理性拓展与模拟

在 Expand 与 Defaultpolicy 中，采取不基于规则、不辨是非的选择与模拟方式，虽可快速产生大量模拟次数，但其中的很多模拟充斥着迫手不应对、有必胜手不下等荒谬情况，因此是质量不高的。大量低质量模拟局给算法的评估引入了过多不确定性。我采用的“理性模拟”利用规则对一些简单的、决定性的情况进行处理：

- 本方具有必胜手，或本方具有次胜手且对方上一步不是迫手时，必然下在必胜位置
- 不送死，即必然应对对方的迫手

这样的简单判断使得算法具有了“向前看一步”的能力，避免了过多无用的分支，因此相当于对蒙特卡洛树的剪枝。

2.4.3 必胜与必败的标注与向上传播

在对某个局面进行拓展时，若下一方的某一手必胜，或无论下在哪里均必败，将对此局面进行标注，并将这种标注向上传播：若子节点代表本方，且此子节点为本方必胜，则将其父节点标注为本方必胜；若子节点代表本方，且所有子节点均为本方必败，则将其父节点标注为本方必败。被标注过的子节点便没有探索的必要，而当向上传播至根节点时，反映的便是当前局面的必胜与必败情况。

2.4.4 估值函数

在 Defaultpolicy 的模拟过程中，虽然对一些简单情形进行判断，但不免出现对对方有利、对己方不利的下法。在随机模拟次数足够多的情况下，得到的胜率可近似反映当前局面，但这种量变到质变的过程需要的模拟时间是很长的，3 秒的模拟和 1 秒的模拟差别不大，均不能准确对局面进行评估。

AlphaGo 采取的是估值网络和策略网络，通过策略网络缩小搜索范围和指导随机模拟，通过估值网络更准确地计算信心上限。但这基于大量棋谱、长时间的训练，以及规模庞大的神经网络。这里对其进行了简化，人为设计一个估值函数，每个候选点的得分代表着随机模拟时选择此点的概率，得分越高被选择的概率越大。估值函数的设计基于显而易见的直觉，如优先选择连接丰富的点，优先选择双方争夺的关键位置。这样，模拟时便不是完全随机的，而是在一定指导下的随机。

2.4.5 优先算杀

在对对局的分析中我发现，算法常常在己方优势很大、可经过连续多步迫手完成必胜时选择保守的下法，在对方即将形成连续迫手杀时不及时阻挡。这主要是由于随机性，算法常常不能使得激烈攻击（或对攻）的分支得到很多访问。因此在 Treepolicy 进行选择时，我采取了如下策略：在未访问拓展的结点中，优先选择迫手。这样，即使采用的是随机算法，也可以达到多数棋类中“算杀”效果。通过对输出的分析，最多可以提前 10 步左右确定必胜或必败。

3 实验与分析

3.1 参数选择

实验中需要调节的超参数只有 UCT 中的 c 值，这里选取 $c = 0.8$。

3.2 评测环境

本实验的测试环节完全在 Saiblo（www.saiblo.com）上进行，编译时采用了-O2 优化。我在 Saiblo 上的用户名为 zhengkw。

3.3 实验结果

这里采用了要求中的测试方法（对战 50AI，正反手各一轮），进行了一次测试。如图 2所示，胜率为 97%。

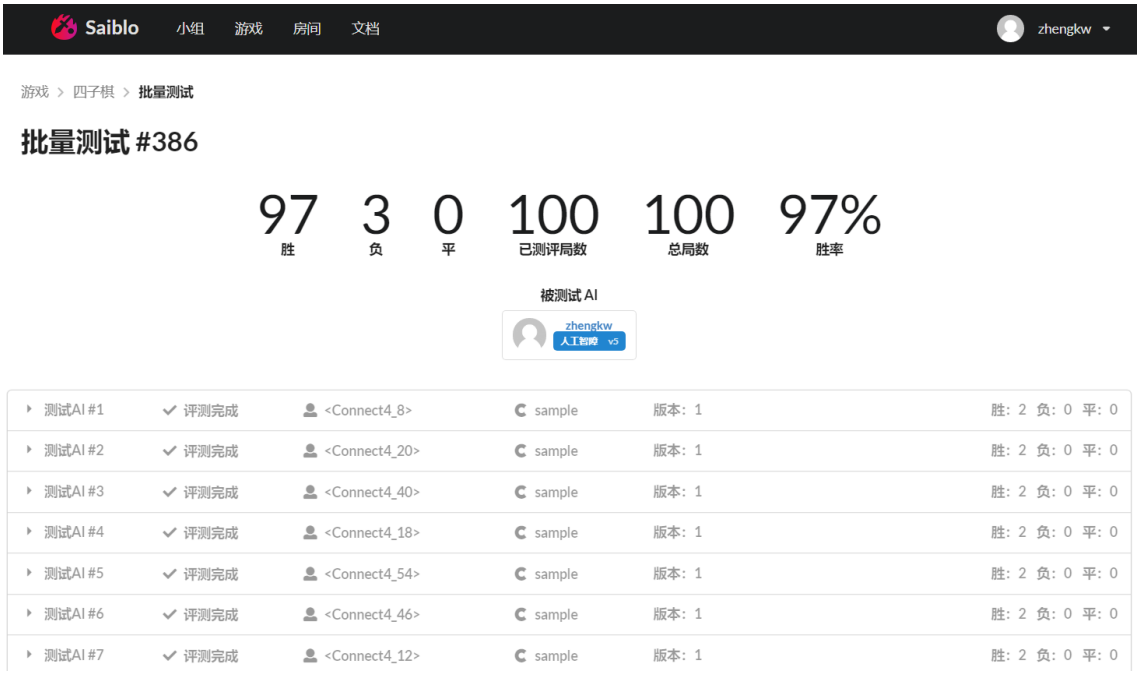

图 2: 对战 50AI 的结果

同时，我还将我的 AI 加入了天梯排名系统，在 2020.4.26 13:10 时排名 2/240，如图 3所示。

3.4 样例分析

在与 50AI 对战时，对大部分的 AI 呈碾压之势，且对局结束很快，可以“速杀”。而个别的对局可能拖到大后期，此时大部分的列已经被占满，我的算法已经无力掌控。

由于是随机算法，尽管对一些弱的 AI 胜率很高，但稳定性不能有很强的保证，“翻车”现象也经常发生。一旦没有快速解决战斗，被拖入大后期，输掉的可能性很大。经过观测，输掉的局均为 50 手以后，即我的算法不会被轻易“速杀”，但打后期的能力较弱。

4 结论

通过对各种资料的查阅参考以及我自己的尝试，我得出了在计算资源和数据量有限时，人为设计指导规则会最大限度提升胜率的结论。在增加了针对性的规则后，总的模拟次数从几十万次下降到了几万次，但模拟对局的质量大大提高。而其余的改进方法，诸如 RAVE、位运算

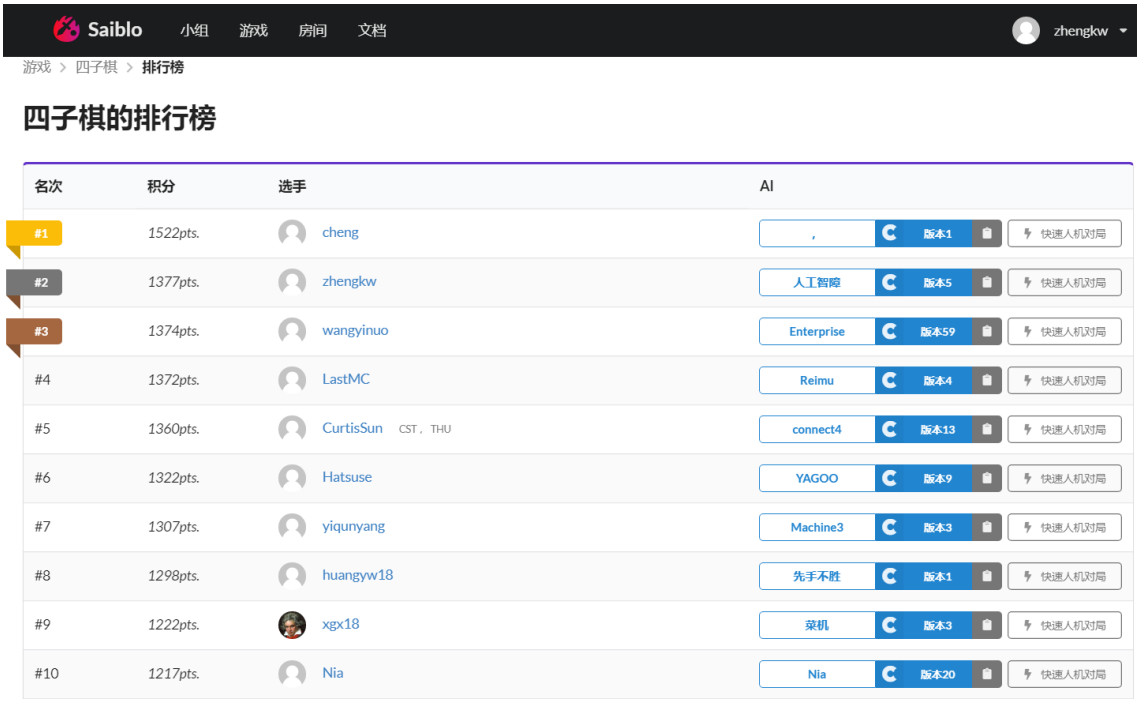

图 3: 天梯排行

优化等，将会使问题变为玄学调参，或只能在常数级别增加模拟次数，总的来说改进不大。

与此同时，我认为我在算法中引入的指导规则仍不够强大，算法不够“稳”，缺乏鲁棒性，虽然经过长时间的起伏可以到达天梯第二的排名，但随便一次的胜率测试并不能显示出相较于普通 UCT 的优势。经过对后期局面的分析，我发现很多情况是，一些列几乎为空，但一旦我方下了这些列，其上的位置便是对方的必胜手，对方可以立即胜利。而当其余列被占满时，又不得不下这些列。对于这种“卡位”胜利法，我的想法是在我的算法中提升其优先级，使其倾向于“优先卡位”。我会在 ddl 之后对其进行尝试。