重力四子棋 实验报告

2018011365 张鹤潇 731931282@qq.com

April 29, 2020

目录

1 基本算法

1.1 MCTS

重力四子棋是典型的完全信息博弈，相应 AI 算法框架主要有 $\alpha\beta$ 剪枝和蒙特卡洛树搜索 (MCTS)。我选用的方法是 MCTS。相比于 $\alpha\beta$ 剪枝，它能充分利用计算资源，不受限于人的先验知识，容易达到更高的竞技水平。

Algorithm 1: MCTS

输入: 初始状态 s_0

输出: 采取的行动 a

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);
  return a(BESTCHILD(v_0));

MCTS 算法首先根据搜索树策略 (Tree Policy) 在搜索树上选择一个节点用于扩展，再以该节点为根进行随机模拟 (Default Policy)，根据模拟结果更新历代祖先节点的胜率估计值 (Back Up)；最后，根据胜率估计值返回最优行动 a.

1.2 UCT

UCT 算法在 MCTS 的基础上，对 Tree Policy 做了一些改进。选择待扩展节点时，以估计胜率为依据并不可靠，因为随机模拟的轮数有限，当前胜率未必能代表真实胜率，最优决策可能因为没有被充分探索，胜率低于某个次优决策，这样，算法就不能选出真正的最优子节点。

为了解决这个问题，UCT 算法定义了节点的信心上界。对于节点 v_i，

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

其中 Q_i 表示 v_i 的胜率，n_i 表示其模拟次数，N 表示总的模拟次数，c 是可调参数。UCT 算法以子节点的信心上界替代胜率，作为选取最优子节点的依据。

对这种改进的直观解释是，如果某个子节点被探索的次数很少，那么 (1) 式中后一项就会很大，算法就能在该节点上进行更多次探索，也就更有可能选出最优解。

2 成果展示

开发环境见 readme 文档。

采用 2.5s 时限，在本机上与 2.dll-100.dll 进行模拟测试，结果胜 95 局、负 5 局；在 Saiblo 在线平台上进行模拟测试，结果获得全胜；在天梯上与同学们的 AI 对战，排名前十。

3 优化细节

3.1 状态压缩

用位运算进行状态压缩，实现常数优化。具体地说，以四个一维数组分别表示棋盘 X 轴、Y 轴、左对角线、右对角线四个方向上的状态。举例如下：

```
# 假设棋盘第一行某时刻的状态是:
. . A . . X B A .
# 将其编码为二进制数(即X[M-1])
00 00 01 00 00 00 10 01 00
```

要判断某一行，或某一列，某一斜对角线上有没有形成四连珠，只需要做如下位运算：

```
inline bool Board::inLine(const int& a) const{
    return (a & (a >> 2) & (a >> 4) & (a >> 6));
}
```

这样判断是否取胜要比 judge.h 中反复访问二维数组快得多。

3.2 Value Policy

UCT 算法中，在叶节点上进行模拟 (即 Default Policy) 的方法是随机选择落子点。在模拟次数足够多的情况下，模拟的平均结果可以近似反映局面的好坏。但显然，这种随机模拟的效率很低。

借鉴 AlphaGo 的思路，我手工设计了一个估值函数来指导随机模拟。具体地说，决定局面 s 下一步的落子时，不是随机选取，而是从概率分布

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

中抽样。其中 $v(a|s)$ 为局面 s 下决策 a 的价值估计。

接下来要做的就是设计合适的 v，它既不能过于复杂，以致拖慢模拟的速度，又要能起到估值的效果。我尝试了很多思路，最后选定的估值方法是统计落子点周围 $A.A/AA.A/\ AA.AA$ 等棋形出现的次数，计算其加权和。

Algorithm 2: Value Policy

```
Function VALUEPOLICY(s):
    while s 不是终止状态 do
        for a_i ∈ A do
            计算 v(a_i|s);
        根据概率分布 P(a|s) 对 a 抽样;
        s := f(s, a);
    return 状态 s 的收益;
```

表 1: Value Policy 优化前后对比

算法	Default Policy	Value Policy
模拟测试胜率	80%	95%

可见，估值模拟的优化效果十分显著。

3.3 内存管理

MCTS 需要大量的空间存储树节点。为了避免动态分配内存消耗时间，我设置了一个节点内存池，在程序启动时开辟全局静态内存，供搜索树使用。

另外，重力四子棋连续两步之间的联系很大，如果能利用上一回合的模拟结果，就能增加模拟次数，提高程序性能。为了实现这个想法，我在内存池中设置了一个维护可用节点的队列。当新一回合开始，需要移动树根时，就将原树根压入队列中；每从队列头取出一个节点，就将其所有子节点都压入队列。这就实现了节点的重复利用。

3.4 搜索剪枝

注意到，如果我方存在制胜点，即我方在此落子后就能取胜，那就只需在此落子；而如果我方不存在制胜点，对方存在，则我方必须在对方的制胜点落子以避免落败。

基于这个思路，对扩展子节点的策略做剪枝。

Algorithm 3: Expand + 剪枝

Function *EXPAND(v)*:
  if *A(state(v))* 中存在我方制胜点 a_{me} **then**
    $a := a_{me}$;
  else if *A(state(v))* 中存在对方制胜点 a_{op} **then**
    $a := a_{op}$;
  else
    任意选择 $a \in A(state(\mathrm{v}))$;
  向节点 v 添加子节点 v', 使 $s(v')=f(s(v),a)$, $a(v')=a$;
  return v';

4 总结反思

本次大作业花费了我很多心血，好在结果还算满意，虽然和时间投入不成正比。

估值模拟对提高 AI 表现很有帮助，但它的局限性也很明显：一是设计合适的估值函数极为困难，只能靠不停尝试，不合理的估值也会限制算法的上限；二是估值和抽样会不可避免地拖慢蒙特卡洛模拟的速度。我也曾仿照 ϵ 贪心的思路尝试将随机模拟和估值模拟结合起来，但是效果不佳。

本地评测环境的不稳定，测例性能与其编号的不相关严重影响了我的开发体验。优化和迭代机器学习系统需要评估算法的性能，而在本地进行一次模拟测试就要花近三个小时，还经常出现错误。

希望老师和助教能重视这些问题，提高大作业的质量！

相比之下，线上测试平台就要好的多。非常感谢 Saiblo.net 的相关开发人员！