重力四子棋
实验报告

计 72 谢兴宇 2017011326

May 2019

目录

1 基本算法

我使用了蒙特卡洛树搜索与 UCT 相结合作为算法的主体框架。

1.1 Monte Carlo Tree Search

蒙特卡洛树搜索是一个在解空间中不断随机抽样的过程，来试图逼近最优解。蒙特卡洛树搜索的每一轮包括以下四步：

1. 选择（Selection）：从根节点 R 出发，不断地选择一个节点的一个子节点，直到到达一个节点 L。
2. 扩展（Expansion）：如果 L 的胜负状态尚不可判，为其创建一个儿子节点 C。
3. 模拟（Simulation）：在 C 的基础上，双方不断随机走子，直到一方获胜或和局。
4. 回退（Backpropagation）：用模拟的结果来更新自 C 到 R 路径上节点的信息（每一个节点的模拟次数和胜率）。

最终选择根节点模拟次数最多的儿子作为最终的答案。

1.2 Upper Confidence Bound 1 applied to trees

在选择子节点时，一个困难之处便在于平衡对于当前胜率较高的节点和模拟次数较少的节点的选择。第一个用于解决此问题的方法由 Levente Kocsis 和 Csaba Szepesvári 提出，称作 UCT(Upper Confidence Bound 1 applied to trees)。该算法选择以下列表达式最大的节点：

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}} \tag{1}$$

其中：

- w_i 代表当前节点模拟中获胜的次数（平局算 0.5 次）。
- n_i 代表当前节点的模拟次数。
- N_i 代表当前节点的父节点的模拟次数。

- c 是一个参数，理论上最优解为 $\sqrt{2}$，实际应用中往往依据经验来选择。

上式的前半部分是对于节点胜率的考量，胜率越大表达式结果也越大；后半部分是对于节点模拟次数的考量，模拟次数越少表达式结果越大。

2 算法改进

2.1 参数调整

经过实验，UCT 算法中的参数最终选择了 $c = \sqrt{2}/2$。

2.2 均匀选择

对于 UCT 算法而言，前期因为偶然而导致模拟时胜率较大的节点可能造成较大的干扰。为了解决这个问题，我采取了这样的策略：如果有子节点的模拟次数不足 Th 次，则优先在这部分节点中随机选择一个。

Th 是我设置的一个阈值，经试验选择了 $Th = 1$。

2.3 末端剪枝

为了加速搜索，在搜索时对于一些必胜/必败情况进行特判：

- 若再下一子便可直接取胜，便直接下这一子。
- 若我方不下，则对方再下一子便可直接取胜，便直接下这一子。
- 若无论我方怎么下之后，对方总可再下一子直接取胜，便将此状态记为必败态。

3 实验结果

与 80-100 的每一个测例 AI 对战一次，胜率 59%。

4 改进计划

- 对于 UCT 算法的**均匀选择**改进中，选取的阈值 5 或许过高了，适当降低这个阈值也许可以取得更好的效果。

- 事实上，博弈树上有的节点就其棋局状态而言是相同的，我们可以将这种节点合并，以更加精确地计算其胜率。不过其模拟次数是不宜合并的，还是需要分开来考虑，这样我们可以把模拟次数记在树的边上。
- 若不考虑空间问题，我们可以对 MCTS 算法作缩减，将 Select-Expansion-Simulation 合并成一步。这样可以更好地利用 Simulation 得到的信息。
- 关于 Final Answer 的选取，除了最终模拟次数这一个指标之外，也需要考虑胜率，或许我们可以借鉴 UCT 的思想，使用根节点 $w_i/n_i+c\sqrt{\frac{\ln N_i}{n_i}}$ 最大的儿子。
- 在模拟时，纯随机模拟想要逼近一个真实的胜率的话，需要的时间代价是非常巨大的，尤其是在解空间巨大的时候。我们可以使用一个估价函数来代替纯随机模拟。