重力四子棋作业报告

计 81 包涵 2018011289

2020 年 4 月 26 日

1 算法简介

在该实验中，我使用了信心上限树的方法，构建 UCT 树，随机模拟对战过程，计算节点收益，选择最佳落子点。

1.1 算法描述

算法主干与课堂教授的 UCT 算法一致，逻辑如下：

- 开始搜索时，以当前输入状态创建根节点 v_0.
- while 计算时长还未达到:
  - 从根节点出发，若当前节点可以扩展新的子节点，那么选择新扩展的子节点作为 v_i；若当前节点不可以扩展新的子节点，那么选择该节点的子节点中收益最高的节点作为 v_i. 其中，收益使用如下公式计算：

$$\frac{Q(v)}{N(v)} + c\sqrt{\frac{2 * \ln N(u)}{N(v)}} \tag{1}$$

    u 为当前节点，v 为子节点，$N(v)$ 为以 v 为根的子树总的模拟次数，$Q(v)$ 为 $N(v)$ 次模拟所获得的收益。
  - 从 v_i 所表示的状态开始，随机模拟对战过程直到棋局结束，并将结果回溯，更新 v_i 和祖先节点。
- 从 v_0 的子节点中选择收益最高的节点进行作为落子点返回。

1.2 优化调整

经过一定的实验发现，如果能够尽量提高模拟的次数，也就是扩展 UCT 树的深度，可能比提高模拟的质量（加入更多对棋局的判断）要更有效。因此，我舍去了本来更复杂的模拟方法，尽量提高模拟次数。

- 在随机模拟对战的过程中，加入了一点对棋型的检测，提高模拟的效率和置信程度。
  - 如果发现对手已经形成 3 个连续的棋子而且两边都可以落子，那么不在扩展当前节点，并且判负。
  - 如果发现对手形成了 3 个连续的棋子，而且只有某一边可以落子，那么在那个位置落子。
  - 否则在可落子的位置上随机选择
- 在检测棋局胜负的时候，把棋盘用 bitset 存储，并使用位运算加速判断，提高模拟的效率。
- 在内存管理方面，保留已经分配的节点地址，重复使用。避免多次分配内存带来的时间开销。

2 实验效果

网络平台测试结果如下：

胜	负	平	总局数
98	2	0	100

分别输给了 86 和 92 号测例。网站账号为 baocvcv, 批量测试 id 为 1499.

3 总结

本次作业加深了我对信心上限和蒙特卡洛搜索方法的理解，通过动手实现算法和调试，加强了编码的能力。收获良多。如果有更多时间，或许可以从以下方面提高算法的表现：

- 增加更多快速有效的棋型检测，现有的棋型检测存在不合理的地方，会限制算法在一些情况下的表现。

- 对于子节点的评估尝试结合 heuristic，提高初期搜索的效率。
- 进一步优化数据结构，提升模拟速度。