实验二 黑白棋游戏 实验报告

银琦（141220132、 141220132@smail.nju.edu.cn）

(南京大学 计算机科学与技术系, 南京 210093)

摘 要: 黑白棋，又称反棋（Reversi）、奥赛罗棋（Othello）等，游戏使用围棋的棋盘棋子，在 8*8 的棋盘上，黑白双方分别落棋，翻动对方的棋子。本文就黑白棋游戏对 Minimax 算法、AlphaBeta 剪枝算法和衍生的 MTDDecider 算法谈一谈自己的理解和想法。

关键词: MiniMax 算法 AlphaBeta 剪枝算法 MTDDecider 算法

1 MiniMax 搜索

1.1 要求

阅读源代码 MiniMaxDecider.java，理解并介绍 MiniMax 搜索的实现（与课本上分别实现 Max 和 Min 函数不同，程序中使用一个开关将两者合并为一个函数）

1.2 理解

假设电脑为 Max。在 MinMaxDecider.java 文件中，主要有两个函数：decide 和 miniMaxRecursor。

其中，decide 函数的主要作用是判断下一步如何走。函数中将当前的节点值与父节点的值进行比较，flag 为一个开关，若当前为 Max，flag 取 1，所以判断若当前节点的值比父节点的值大，则将父节点的值替换为当前节点的值；若当前为 Min，flag 取-1，所以当前节点与父节点取负后选大的，即在当前节点和父节点中选取小值，这样使得总是选取最有利的状态。若父节点经过了更新，则将原来的旧状态删去，并把新的状态加入最佳操作的候选队伍中。

另一个函数为 miniMaxRecursor，这个函数在 decide 中被调用，递归并利用 heuristic 函数求出叶节点的值，然后回溯，比较其与父节点的值，过程与 decide 函数类似。即可选择出最有利的下一步。

当用户选择的下一位置不合法时，会利用 try，catch 机制提示用户选择错误。

2 AlphaBeta 剪枝算法

2.1 要求

请修改 MiniMaxDecider 类，加入 AlphaBeta 剪枝，并且比较引入剪枝带来的速度变化（尝试不同的搜索最大深度）。

2.2 修改

根据 AlphaBeta 剪枝的定义，如果当前节点比父节点小，则不需要继续搜索当前节点的其他分支了，利用原来代码中的开关“flag”，可以实现当前节点与父节点的大小。所以在函数 miniMaxRecursor 中增加了变量 lastValue，用来记录父节点的值，如果子节点的值小于等于父节点的值，那么跳出循环，直接返回该状态的下该节点的值。添加两行代码如下：

```
float lastValue = value;
for (Action action : test) {
    // Check it. Is it better? If so, keep it.
    try {
        State childState = action.applyTo(state);
        float newValue = this.miniMaxRecursor(childState, depth + 1, !maximize);
        //Record the best value
                        if (flag * newValue > flag * value) {
                            lastValue = value;        //剪枝后
                            value = newValue;
                        }
                        if(flag * value <= flag * lastValue) break;        //剪枝后
    } catch (InvalidActionException e) {
                        //Should not go here
        throw new RuntimeException("Invalid action!");
    }
}
```

2.3 比较

在 Othello.java 文件的 computerMove 函数中，在"Starting Computer Move"与"Finished with computer move"之后获取了当前时间，并作差获得了电脑移动一步所需时间。输出结果可看到电脑每次移动所需时间，我测试了深度为 4、5、6。若电脑为先手，剪枝前电脑下第一步的时间总是比剪枝后下第一步的时间短许多，但是多下几步之后并不一定总是剪枝后优于剪枝前。我觉得主要原因是电脑的搜索顺序是随机的，若剪枝后的搜索碰到了最坏情况就不会优于剪枝前。我认为对剪枝前后的效率的测试应该让电脑与电脑对战，多次测试后才可得出结论，仅看几步的时间难以得出结论。

3 Heuristic 函数

3.1 要求

请理解 othello.OthelloState 类中的 heuristic 函数，并尝试改进。

3.2 理解

Heuristic 函数是一个启发式函数，用来估算每个状态下的收益，搜索的深度可以进行限制，源代码中 heuristic 函数的返回值由四部分组成，该启发式函数大致如下：heuristic() = pieceDifferential() + 8*moveDifferential() + 300*cornerDifferential() + 1*stabilityDifferential() + winconstant

其中 pieceDifferential 是棋盘上双方棋子数量之差，moveDifferential 是双方可以下的位置数之差，cornerDifferential 是双方在四个角落占据的数量之差，stabilityDifferential 是指不会被对方翻动的稳定棋的数量之差，winconstant 是一个定值，Max 方为+5000，Min 方为-5000。每个函数前面所乘的数字是它们所占的权值。

3.3 改进

我觉得第一个可以改进的地方是可以考虑在边上但是不在四个角上的棋子数量。但是这个的权值不应该比角上的权值大，所以可以在原函数的基础上增加一项（共有 24 个格子）：100*sideDifferential()。

第二个可以改进的地方是：要增加不能占位的点的判断，对于八行八列的棋盘，第二行第二列，第二行第七列，第七行第二列，第七行第七列不能落子，否则如果让对方占据了四个角落，会有很大损失，所以可以在原函数的基础上增加一项：(-200)*punishDifferential()。

综上，启发式函数可改为：heuristic() = pieceDifferential() + 8*moveDifferential() + 300*cornerDifferential() + 1*stabilityDifferential() + 100*sideDifferential() + (-200)*punishDifferential() + winconstant

4 MTDDecider 算法

4.1 要求

请阅读并尽量理解 MTDDecider 类，介绍它与 MiniMaxDecider 类的异同（在 Othello.java 中，将 294 行注释掉，将 295 行去掉注释，即可换用 MTDDecider 算法）。

4.2 理解

MTDDecider 选择迭代加深来优化算法。输入参数中有一个是 firstguess，此后一直根据搜索的结果寻找当时最优的选择来进行 AlphaBeta 剪枝，因为该算法将每次搜索过的最优情况都记录下来了，最终将收敛为一个确定的值。MiniMax 算法中没有这么一个优化环节，所以 AlphaBeta 剪枝如果在最坏情况下，即搜索顺序很糟糕，效率的提升效果就不会很明显。MTDDecider 在下一层搜索开始前有限选择以搜索的最优记录，这就能让 AlphaBeta 的效率尽可能提高。

MTDDecider 通过使用置换表技术，把已经搜索过的节点的 value 保存在内存中，迭代回溯的过程中如果重复可以直接返回结果，减少了搜索时间。

References:

[1] http://lamda.nju.edu.cn/yuy/(X(1)S(bg3zda45zj2no0ffklpjihzg))/course_ai17_hw2.ashx

[2] http://www.soongsky.com/strategy/rule.php

[3] http://www.cnblogs.com/pangxiaodong/archive/2011/05/26/2058864.html

[4] http://jingyan.baidu.com/article/f71d60375866441ab741d15a.html

[5] http://blog.csdn.net/u012501320/article/details/25098401