人工智能导论:作业二实验报告

薛正海（181220063 xuezh@smail.nju.edu.cn）

(南京大学 人工智能学院, 南京 210093)

1 最小最大搜索

样例代码与伪代码之间最大的区别在于样例代码将最大层和最小层的搜索相统一。这反映出软件工程中“尽量避免 copy-paste”的原则，这种行为使得 bug 的出现难以跟踪。其统一的方式是引入 flag 变量指示当前是否为最小/最大。

在递归进行最小最大搜索的函数 miniMaxRecursor 中，代码首先利用 computedStates 对已计算过的 state 进行识别并直接返回，这是用空间换时间的典型做法，其次其判断是否达到搜索的最大深度，若已达到即利用 herustic function 返回启发函数值作为当前节点的 value 值。若需继续搜索，则继续递归过程并交换最大最小。事实上，代码中实际未将已搜索过的节点放入一个哈希表中以备检索，可能是因为重复的状态太少，需要维护的哈希表过大，同时加速搜索的效果不明显。另外，若有若干个相同的最好动作，则代码随机选择一个动作执行。

2 加入 alpha-beta 剪枝的最小最大搜索

我的加入 alpha-beta 剪枝的代码同样体现了“避免 copy-paste”的原则：我没有将原 miniMaxRecursor 复制一遍，加入改进代码变为 miniMaxAlphaBeta 函数，而是利用 use_pruing 参数直接在原 miniMaxRecursor 中进行修改。根据伪代码，Alpha-beta 剪枝的实现较为容易，但需要理解 flag 控制最小最大层的方法并加以利用。具体的代码如下：

```
if (use_pruning)
{
        if (flag * value >= flag * prune_condition)
        {
                return finalize(state, value);
        }
        if (flag * value > flag * ab_to_change)
        {
                ab_to_change = value;
        }
}
```

代码中我还加入 assert(value==value_pruned)，保证剪枝不会影响 value 的计算。具体的调用过程如下：

```
State newState = action.applyTo(state);
float newValue = this.miniMaxRecursor(newState, 1, !this.maximize, false, Float.NEGATIVE_INFINITY, Float.POSITIVE_INFINITY);
float newValue_pruned = this.miniMaxRecursor(newState, 1, !this.maximize, true, Float.NEGATIVE_INFINITY, Float.POSITIVE_INFINITY);
assert (newValue == newValue_pruned);
```

是否加入 alpha-beta 剪枝在不同深度下消耗时间的对比如下表所示。可见 alpha-beta 剪枝的引入可显著缩短搜索所需的时间。

深度/算法	Vanilla miniMax	miniMax with alpha-beta pruing
2	1.74ms	0.92ms
3	7.09ms	5.43ms
4	31.44ms	22.03ms
5	173.29ms	108.82ms

(PS:随着搜索加深，电脑变得越来越强，人类完全不是对手）

3 理解 heuristic function

框架代码中的 heuristic function 是五个函数值的加权平均。其中 winconstant 描述了比赛结果，获胜则正，否则为负。pieceDifferential 描述了对战双方当前拥有的棋子数，函数值与比对方多的棋子数成正比（可能为负）。moveDifferential 描述了当前双方可选的落子方式，函数值与比对方多的选择方式数成正比。cornerDifferencial 描述双方占据棋盘四个角的情况，占据越多函数值越高。这是因为角上的棋子不可能被翻转，角上的棋子越多，当前选手优势越大。stabilityDifferential 是广义的 cornerDifferential，描述了所有不可能被翻转的棋子，函数值与比对方多的棋子数成正比。这五个函数值的权重被硬编码在代码中，应该是经过多次试验尝试出的较优值。

在熟悉黑白棋游戏的过程中，我起初觉得电脑实力非常强劲，自己完全不是对手。但后来我了解到挂角的重要性后，有意识地占据四个角的位置，便经常能战胜电脑。因此，对于非初学者，可以加大占角的 heuristic function 的权重值。更好的方法可能是对对手的行为建模，以针对性调整自身的启发函数权重。

4 理解 MTDDicider 类

MTDDecider 采用 MTD(f)算法。其与 vanilla alpha-beta 剪枝在剪枝的条件判断上基本相同，但有两个主要区别。第一个区别是搜索深度的选择。在 Alpha-beta 剪枝中，程序直接搜索到指定深度，而 MTD(f)算法迭代地加深搜索，直到达到指定深度。第二个区别在于 MTD(f)算法限制了 alpha 与 beta 的距离（通常为 1）。这样小的距离造成更多的剪枝，速度更快，但是获得的信息更少：相比于 alpha-beta 剪枝能获取精确的 value 值，MTD(f)算法只能获得关于 miniMax value 的一个 bound，因此需要一轮轮迭代，逐步缩小 bound 获得精确值。另外，框架代码中显式指出需存储 state-value pair，以备在到达相同状态时直接寻值。MTD(f)中的 f 代表对 alpha 的第一次猜测。由于 alpha-beta 的窗口大小只有 1，好的猜测能极大加速程序的执行过程。

以上就是我此次作业报告的全部内容，感谢您的阅读！