Minimax 算法在中国象棋的应用

武自厚 20336014 保密管理

2022 年 4 月 20 日

1 原理分析

1.1 Minimax 算法

Minimax 算法, 是一种应对博弈游戏 (尤其是零和博弈) 时考虑步骤所需要的博弈树搜索算法. 其基本原理为: 每一个叶子结点都具有一个值, 而奇数层 Max(代表一名玩家的决策) 结点的值是其子结点的最大值; 偶数层 Mini(代表另一名玩家的决策) 结点的值是其子结点的最小值.

算法步骤可以用自然语言归纳为如下几步.

1. 获取当前状态的行动列表.
2. 遍历行动列表, 并获取下一步的权值 (递归执行).
3. 选取最大 (小) 的权值返回.

如此得出当前状态下一步状态的全部权值, 选择其中具有最大权值的行动执行即可.

1.2 $\alpha - \beta$ 剪枝

Minimax 算法虽然可行, 但是它过大的时空复杂度限制了算法解决问题的规模. 对此, 一个有效的优化就是引入 $\alpha - \beta$ 剪枝.

此时, 一个结点拥有的权值不再是一个, 而是两个, 它们称为 α 和 β, 代表了这个结点的递归算法中出现过的最小值和最大值. 如果某一时刻一个结点出现了 $\alpha \geq \beta$ 的情况, 这就说明它最后采纳的子结点已经被遍历, 不再需要考虑之后的子结点 (即 “剪枝”). 容易知道,Max 结点的 α 值就是其权值, 而 Mini 结点的 β 值就是其权值.

1.3 深度受限与评估函数

对于中国象棋而言, 一昧向下搜索企图寻找到终止状态 (即一方获胜) 所造成的时空代价会非常高, 所以对于搜索的深度进行限制很有必要. 但是到达最大深度时并不能直接获取状态的权值, 这时就需要评估函数来对最大深度结点进行启发式评估以获取权值.

目前已经提出了一些根据棋子位置、棋子个数、棋子灵活性和特殊局面进行棋局评估的函数[1], 对此本文不再赘述.

1.4 历史表启发 (创新点)

$\alpha - \beta$ 剪枝在一定程度上可以降低计算次数, 但是其效果取决于行动的排序, 如果被采纳的行动是最后被遍历的, 那么 $\alpha - \beta$ 剪枝将不起作用. Jonathan 在 1989 年提出一种称为 "历史表启发" 的优化方式会启发式地调整算法遍历顺序[2], 让 $\alpha - \beta$ 剪枝发挥最大的作用.

历史表记录了一个行动最优性的权值, 它的操作分为访问和更新两种. 访问操作用于获取行动的最优性权值, 而更新操作用于增加权值. 在获取行动列表时, 算法将会根据各个行动在历史表中的权值进行排序. 在返回时, 算法将会对造成最大 (小) 值的行动进行历史表更新.

2 伪代码实现

2.1 $\alpha - \beta$ 剪枝的 Minimax 算法实现

Algorithm 1: `alphabeta`$(s, \alpha, \beta, \texttt{depth})$

输入: 当前状态 s, 继承权值 α, β

输出: 最大 α 值

```
if depth ≤ 0 then
    return evaluate(s)
S := s 的后继状态
for s' ∈ S do
    v := alphabeta(s', −β, −α, depth − 1)
    if v ≥ α then
        α := v
    if α ≥ β then
        break
return α
```

2.2 具有历史表启发以及 $\alpha-\beta$ 剪枝的 Minimax 算法实现

Algorithm 2: alphabeta$(s, \alpha, \beta, \text{depth})$

输入: 当前状态 s, 继承权值 α, β
输出: 最大 α 值
全局: 历史启发表 history

```
if depth ≤ 0 then
    return evaluate(s)
S := s 的后继状态
S.sort() ;                                  // 按照 s ↦ history[s] 排序
for s' ∈ S do
    v := alphabeta(s', −β, −α, depth − 1)
    if v ≥ α then
        α := v
        s* := s' ;                          // 记录最优后继状态
    if α ≥ β then
        break
history[s*] := history[s*] + 2^depth ;      // 更新历史表记录
return α
```

3 关键代码

3.1 棋局评估函数

```python
class Evaluate(object):
    # 棋子棋力得分
    single_chess_point = {
        'c': 989,   # 车
        'm': 439,   # 马
        'p': 442,   # 炮
        's': 226,   # 士
        'x': 210,   # 象
        'z': 55,    # 卒
        'j': 65536  # 将
```

```
    }
    # 红兵（卒）位置得分
    red_bin_pos_point = [
        [1, 3, 9, 10, 12, 10, 9, 3, 1],
        [18, 36, 56, 95, 118, 95, 56, 36, 18],
        [15, 28, 42, 73, 80, 73, 42, 28, 15],
        [13, 22, 30, 42, 52, 42, 30, 22, 13],
        [8, 17, 18, 21, 26, 21, 18, 17, 8],
        [3, 0, 7, 0, 8, 0, 7, 0, 3],
        [-1, 0, -3, 0, 3, 0, -3, 0, -1],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
    ]
    # 红车位置得分
    red_che_pos_point = [
        [185, 195, 190, 210, 220, 210, 190, 195, 185],
        [185, 203, 198, 230, 245, 230, 198, 203, 185],
        [180, 198, 190, 215, 225, 215, 190, 198, 180],
        [180, 200, 195, 220, 230, 220, 195, 200, 180],
        [180, 190, 180, 205, 225, 205, 180, 190, 180],
        [155, 185, 172, 215, 215, 215, 172, 185, 155],
        [110, 148, 135, 185, 190, 185, 135, 148, 110],
        [100, 115, 105, 140, 135, 140, 105, 115, 110],
        [115, 95, 100, 155, 115, 155, 100, 95, 115],
        [20, 120, 105, 140, 115, 150, 105, 120, 20]
    ]
    # 红马位置得分
    red_ma_pos_point = [
        [80, 105, 135, 120, 80, 120, 135, 105, 80],
        [80, 115, 200, 135, 105, 135, 200, 115, 80],
        [120, 125, 135, 150, 145, 150, 135, 125, 120],
        [105, 175, 145, 175, 150, 175, 145, 175, 105],
        [90, 135, 125, 145, 135, 145, 125, 135, 90],
        [80, 120, 135, 125, 120, 125, 135, 120, 80],
        [45, 90, 105, 190, 110, 90, 105, 90, 45],
```

```
        [80, 45, 105, 105, 80, 105, 105, 45, 80],
        [20, 45, 80, 80, -10, 80, 80, 45, 20],
        [20, -20, 20, 20, 20, 20, 20, -20, 20]
    ]
    # 红炮位置得分
    red_pao_pos_point = [
        [190, 180, 190, 70, 10, 70, 190, 180, 190],
        [70, 120, 100, 90, 150, 90, 100, 120, 70],
        [70, 90, 80, 90, 200, 90, 80, 90, 70],
        [60, 80, 60, 50, 210, 50, 60, 80, 60],
        [90, 50, 90, 70, 220, 70, 90, 50, 90],
        [120, 70, 100, 60, 230, 60, 100, 70, 120],
        [10, 30, 10, 30, 120, 30, 10, 30, 10],
        [30, -20, 30, 20, 200, 20, 30, -20, 30],
        [30, 10, 30, 30, -10, 30, 30, 10, 30],
        [20, 20, 20, 20, -10, 20, 20, 20, 20]
    ]
    # 红将位置得分
    red_jiang_pos_point = [
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 9750, 9800, 9750, 0, 0, 0],
        [0, 0, 0, 9900, 9900, 9900, 0, 0, 0],
        [0, 0, 0, 10000, 10000, 10000, 0, 0, 0],
    ]
    # 红相或士位置得分
    red_xiang_shi_pos_point = [
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 60, 0, 0, 0, 60, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [80, 0, 0, 80, 90, 80, 0, 0, 80],
        [0, 0, 0, 0, 0, 120, 0, 0, 0],
        [0, 0, 70, 100, 0, 100, 70, 0, 0],
    ]

    red_pos_point = {
        'z': red_bin_pos_point,
        'm': red_ma_pos_point,
        'c': red_che_pos_point,
        'j': red_jiang_pos_point,
        'p': red_pao_pos_point,
        'x': red_xiang_shi_pos_point,
        's': red_xiang_shi_pos_point
    }

    def __init__(self, team):
        self.team = team

    def get_single_chess_point(self, chessmap, src):
        chess = chessmap[src[0]][src[1]]
        if chess[0] == self.team:
            return self.single_chess_point[chess[1]]
        else:
            return -1 * self.single_chess_point[chess[1]]

    def get_chess_pos_point(self, chessmap, src):
        row, col = src
        chess = chessmap[row][col]
        red_pos_point_table = self.red_pos_point[chess[1]]
        if chess[0] == 'r':
            pos_point = red_pos_point_table[row][col]
        else:
            pos_point = red_pos_point_table[9 - row][col]
```

```
        if chess[0] != self.team:
            pos_point *= -1
        return pos_point

    def evaluate(self, chessmap):
        point = 0
        for chess in fetch_any_chess(chessmap):
            point += self.get_single_chess_point(chessmap, chess)
            point += self.get_chess_pos_point(chessmap, chess)
        return point
```

3.2 具有历史表启发以及 $\alpha-\beta$ 剪枝的 Minimax 函数

```
history_map = {}
# 此处初始化history
def history_get(x):
    if not history_map[x]:
        history_map[x] = 0
    return history_map[x]

def history_renew(x, depth):
    history_map[x] += 2 << depth

def alpha_beta(state, alpha, beta, depth):
    if depth == 0:
        return evaluate(state)
    camp = 'r' if depth % 2 == 0 else 'b' # 深度决定行动方
    steps = generate_possbile_steps(state, camp)
    steps.sort(key=history_get)
    next_states = [make_nxt_state(state, src, dst) for src, dst in steps]
    target = None
    for i, next_state in enumerate(next_states):
        temp = alpha_beta(next_state, -beta, -alpha, depth - 1)
        if temp > alpha:
            alpha = temp
            target = steps[i]
```

```
24            if alpha >= beta:
25                break
26
27        if target is not None:
28            history_renew(target, depth)
29        return alpha
```

4 结果分析

同目录的 `result.mp4` 文件展示了一个模拟棋局的效果. 可以发现该算法具有一定的智能, 至少对于一个象棋菜鸟来说绰绰有余.

5 总结

参考文献

[1] 徐心和, 王骄. 中国象棋计算机博弈关键技术分析[J]. 小型微型计算机系统, 2006, 27(6): 961-969.

[2] SCHAEFFER J. The history heuristic and alpha-beta search enhancements in practice [J]. IEEE transactions on pattern analysis and machine intelligence, 1989, 11(11): 1203-1212.