16 级计科 7 班: 人工智能 # 博弈树搜索

Due on Tuesday, December 27, 2018

饶洋辉 周三 *3-4* 节

颜彬

16337269

Content

Page

1 算法简介

1.0.1 Minimax 搜索

Minimax 算法即最大最小值算法，它是一种树型的搜索算法，被广泛运用在博弈类问题上。

加深玩家与 AI 进行博弈（AI 先手）。AI 可能有多种行动方法，对每种行动玩家也有对应的多种行动方法。Minimax 搜索解决的问题是，如何为 AI 选择一种最适宜于当前局势的行动方法。

Minimax 算法的思路是，

- 每当 AI 行动时，它总是会选择让 AI 胜率最高的方法行动
- 每当玩家进行行动时，他总是会选择让 AI 胜率最低的方法行动

当搜索树的某层是轮到 AI 行动时，该层的结点的评分数会等于所有子结点评分的最大值，对玩家亦然。运用这个算法搜遍整个博弈树（或搜索完某个特定深度），AI 即可选出最佳的行动方法。

这就是算法名字的来历，他会交替地使用最大值和最小值来确定当前结点的估值。如果当前结点的值由子节点的最大值来确定，那么当前结点称为最大结点，反之依然。

理论上，当整棵搜索树可以被完全遍历时，可以找到一个全局最优的下法。但搜索树往往很大，无法被完全遍历。故一般搜索时会设定一个最大深度，当搜索达到深度上限时，使用估值函数近似当前棋盘的局势。

1.0.2 Alpha-beta 剪枝

Alpha-beta 剪枝是一种 minimax 算法的剪枝方法。它的思路是，当搜索到的某个结点的子结点评分太高或太低时，该结点很可能对整棵博弈树的评分没有影响（因为父结点很可能根本不会犯傻走这个结点），于是进行剪枝，避免进行不必要的运算。

具体地说，在整个 minimax 的搜索中，维护两个变量 α（初始化为 $-\infty$）和 β（初始化为 ∞)。区间 $[\alpha, \beta]$（初始为 $[-\infty, \infty]$) 代表着一个 ‘‘可接受的区间’’。

随着搜索逐步地进行，α 和 β 都会逐步地更新。

- 在 ‘‘最大值’’ 层的搜索中，仅会更新 α。如果最大值层的某个子结点的值大于 α，则将 α 的值更新为子结点的值。
- 在 ‘‘最小值’’ 层的搜索中，仅会更新 β。如果最小值层的某个子结点的值小于 β，则将 β 的值更新为子结点的值。

在搜索的任何时刻，如果出现 $\alpha >= \beta$，则可以直接进行剪枝，没有必要进一步地搜索。

这是由于，$[\alpha, \beta]$ 标记一个 ‘‘可接受的区间’’。换言之，即 AI 已找到一种决策，获得 α 以上的估值；玩家已找到一种策略，获得 β 以下的估值。如果发生了 $\alpha >= \beta$，要么是因为在最大值结点更新了 α, 要么是因为在最小值结点更新了 β。但无论是那种情况，例如是 α 被更新了，那么玩家根本不会选择走向这个结点，因为玩家可以选择分值更低的 β 对应的结点，反之亦然。

以图 1为例子介绍一下 alpha-beta 剪枝。

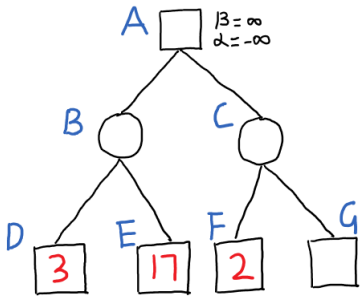

图 1: alpha-beta 剪枝示意图

如图 1所示，方块结点是取最大值的结点，圆形结点是取最小值的结点。在搜索到 D 时，更新 $\alpha_D = 3$。在搜索到 E 时，更新 $\alpha_E = 17$。回溯到结点 B，它是取最小值结点，B 的取值为 3, 更新 $\beta_B = 3$。回溯到 A 结点，其为最大值结点，更新 A 的值为 3，更新 $\alpha_A = 3$ 。

在结点 A，$\alpha_A = 3$，意味着可接受区间为 $[3, \infty]$。即 AI 可以确保，局面至少会获得 $\alpha_A = 3$ 分。

在搜索到结点 F 时，更新 $\alpha_F = 2$。回溯到结点 C 时，更新 $\beta_C = 2$。由于 C 继承了 A 结点的 α 值，有 $\alpha_C = \alpha_A = 3$。在结点 C 出现了 $\alpha_C > \beta_C$，满足剪枝条件，不再对 C 作进一步的搜索，直接回溯到上层结点 A。

结点 C 发生的剪枝可以这样理解。AI 已经能确保能获得估值为 3 的状态。由于结点 C 搜索到了估值为 2 的 F 结点，如果 AI 走向 C 结点，玩家必然会使局势走向 F 结点（或比 F 更差的结点）。故 C 结点没有进一步探索的必要了。

2 算法伪代码

2.1 Minimax 算法

minimax 算法的伪代码如算法 1所示。其中对 Minimax 函数的调用方法见第21行。数字 8 表示调用最大深度。

2.2 Alpha-beta 剪枝算法

Alpha-beta 算法的伪代码见算法 2。其中调用方法见 29行。

与算法 1相比，多出来的地方是，在取最大值的结点更新 α, 在取最小值的结点更新 β。在 $\alpha > \beta$ 时剪枝。

3 尝试的估值函数

3.1 数子法

数子法的思路很简单，棋盘的估值等于己方棋子数减去对方棋子数。

Algorithm 1 Minimax 算法伪代码

```
function MINIMAX(node, depth, isMax)
    if depth == 0 then
        return evaluation                                  ▷ 返回结点的估值
    end if
    if isMax then                                          ▷ 如果是取最大值的结点
        v ← −∞
        for Each child of node do
            val ← MINIMAX(child, depth - 1, ! isMax)
            v ← MAX(v, val)
        end for
    else                                                   ▷ 如果是取最小值的结点
        v ← ∞
        for Each child of node do
            val ← MINIMAX(child, depth - 1, ! isMax)
            v ← MIN(v, val)
        end for
    end if
    return v
end function

MINIMAX(root, 8, True)
```

3.2 不动点估值

不动点指的是不可能再被对方翻转的棋子。不动点相当于棋子的根基，可以认为是绝对的地盘。

对不动点完整的计算比较耗费时间。这里采用近似估计的方式实现。考虑到不动点一般出现在边缘和角落的位置，故可以在估值时给予边缘和角落更大的权重。

采用不动点估值时，AI 会更倾向于占领四周和四角，以企图获得更多的不动点。

3.3 行动力估值

行动力指的是，能够落子的位置的个数。行动力约低，意味着落子的可能选择越少，局势越容易受到对方的牵制。行动力约高，意味着改变局势的可能性越大。

进行行动力估值的一种方式是，用己方的行动力减去对方的行动力。另一种方式是，用己方的行动力减去对方行动力的两倍。后者的 AI 更倾向于压制对方的行动力，往往会在中后期控制整个局面。

4 关键部分代码

alpha-beta 剪枝的 C++ 实现见代码 1。其基本上按照伪代码的思路完成。

在代码中，role 是一个 char 类型的变量，取值有 BLACK('X') 或 WHITE('O') 两种。ChessBox 是棋盘类型，有两个成员函数。Drop 用来下子并翻转。dropables 返回一个可迭代对象，表示每个可落子的

Algorithm 2 Alpha-beta 算法伪代码

```
function ALPHABETA(node, alpha, beta, depth, isMax)
    if depth == 0 then
        return evaluation                                  ▷ 返回结点的估值
    end if
    if isMax then                                          ▷ 如果是取最大值的结点
        v ← −∞
        for Each child of node do
            val ← MINIMAX(child, alpha, beta, depth - 1, ! isMax)
            v ← MAX(v, val)
            α ← MAX(α, val)                                ▷ 更新 α
            if α > β then
                break
            end if
        end for
    else                                                   ▷ 如果是取最小值的结点
        v ← ∞
        for Each child of node do
            val ← MINIMAX(child, alpha, beta, depth - 1, ! isMax)
            v ← MIN(v, val)
            β ← MIN(β, val)                                ▷ 更新 β
            if α > β then
                break
            end if
        end for
    end if
    return v
end function

ALPHABETA(root, −∞, ∞, 8, True)
```

点。

4.1 alpha-beta 剪枝代码

4.2 一些估值函数的实现

棋盘旗子估值和限制行动力估值相结合，成为本项目主要运用的估值方法。如代码 2所示。

代码的第一个循环用来进行棋面估值。其中每个棋子占 1 分，每个边上的棋子占 size 分。size 为棋盘的大小。

行动力的部分，在上述的估值完成后，加上自己的行动力，减去对方的行动力。

另外一种表现不错的估值方法是行动力估值。如代码 3所示。

事实上，在跟随机 AI（永远随机选择可落子点）的对战中，行动力估值法的表现比上述的棋盘估值和行动力限制法效果要更好。但这不一定意味着行动力估值法的棋力真的更强。

原因可能是，棋盘估值法过于 ‘‘聪明’’, 会避免走很多 ‘‘容易被对手将死’’ 的路线，但随机 AI 由于过于笨，很可能走不出 ‘‘将死对方’’ 的下法。这就导致棋盘估值 AI 太过 ‘‘谨慎’’，放弃了很多高风险但高收益的下法。所以获胜得很慢。

但对于纯行动力 AI 来说，由于在开局时，棋盘空位多，能限制对方行动力的走法很少，故纯行动力 AI 在刚开局时接近于随机 AI，与随机 AI 打成平手。在棋局发展，棋子越来越多时，纯行动力 AI 开始发挥作用，让随机 AI 的下法越来越少。最终随机 AI 的下法会少到只剩下 1 到 2 种下法，甚至出现连续无法下子的情况，最终纯行动力 AI 让局面翻天覆地地变化。

5 实验结果

在本节展示的实验结果图片中，’X’ 均代表玩家的子，’O’ 均代表 AI 的子。其中玩家先手。’+’ 代表可落子的位置。在每一个棋盘后都会附带一个数字，这个数字代表场上的估值（数字越高对 AI 越有利）。

5.1 纯行动力 AI

纯行动力 AI 即估值的时候，仅仅以双方行动力来估值，而不管其他条件（例如场上棋子数和棋子分布等）。

事实上，纯行动力 AI 在和人的对战中获得了比较好的结果。在棋局刚开始时，双方还没有展示出优势。如图 2所示。但很快，纯行动力 AI 对玩家的行动力进行了极其大的压制，玩家出现了较长时间的 ‘‘下子可选位置只有 2 或 3 个’’ 的窘境。例如 图 3所示。图中，玩家连续两轮只有 3 个可落子点，但 AI 的可落子点分别为 13 个和 12 个。

在整个对战过程中，出现了 6 轮玩家无法下子，3 轮玩家只有 1 个落子点，5 轮只有 2 个落子点，5 轮仅有 3 个落子点。整个对战中，甚至玩家出现过连续 3 轮无法落子。最终 AI 大比分赢了玩家。完整的棋谱见代码压缩文件包的/log/match1.txt。

5.2 棋盘局面和行动力限制估值 AI

这个 AI 会综合估计场上的棋子位置进行估值，并同时参考双方的行动力进行限制。如图 5和**??**所示。

这个 AI 表现得稳扎稳打。即它会与玩家来回拉锯，逐渐占据自己的有利地位。它会同时估量自己场上的棋子数和行动力，以一定的权重考虑最优解。

完整的棋谱见代码压缩包中的/log/match2.txt

代码 1: alpha-beta 剪枝代码

```
double MiniMaxSolution::__solve(const ChessBox& cb,
    const Position& position,
    char role,
    double alpha,
    double beta,
    int depth) const {

    char otherRole = role == BLACK ? WHITE : BLACK;
    // recursive base
    if (depth >= __depth) {
        return evaluation(cb);
    }

    ChessBox new_chess_box = cb;
    new_chess_box.Drop(position.first, position.second, role);

    double pivot;
    if (depth % 2 == 0) {
        // min node
        pivot = 1e8;
    } else {
        pivot = -1e8;
    }

    vector<Position> dropables = new_chess_box.Dropable(otherRole);
    if (dropables.size() == 0) {
        return evaluation(cb);
    }
    for (const auto& p : dropables) {
        double val = __solve(new_chess_box,
            p, otherRole,
            alpha, beta,
            depth + 1);
        if (depth % 2 == 0) {
            // min node, update beta
            beta = std::min(beta, val);
            pivot = std::min(pivot, val);
            if (alpha >= beta) {
                break;
            }
        } else {
            // max node, update alpha
            alpha = std::max(alpha, val);
            pivot = std::max(pivot, val);
            if (alpha >= beta) {
                break;
            }
        }
    }
    return pivot;
}
```

代码 2: 限制行动力和不动点估值算法

```
double ActionPressEval::evaluate(const ChessBox& chessbox, char role) const {
    char otherRole = role == BLACK ? WHITE : BLACK;
    int size = chessbox.size();
    int eval = 0;
    // 棋盘估值和不动点估值
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            int factor;
            if (chessbox.val(i, j) == role) {
                factor = 1;
            } else if (chessbox.val(i, j) == otherRole) {
                factor = -1;
            } else {
                factor = 0;
            }
            eval += factor;
            if (i == 0 || j == 0) {
                eval += size * factor;
            }
            if (i == size-1 || j == size-1) {
                eval += size * factor;
            }
        }
    }
    // 限制行动力估值
    eval += chessbox.Dropable(role).size() ;
    eval -= chessbox.Dropable(otherRole).size() ;
    return eval;
}
```

代码 3: 纯行动力估值算法

```
double OnlyDroppableEval::evaluate(const ChessBox& chessbox, char role) const {
    char otherRole = role == BLACK ? WHITE : BLACK;
    int size = chessbox.size();
    int eval = 0;
    eval += chessbox.Dropable(role).size() ;
    eval -= chessbox.Dropable(otherRole).size() ;
    return eval;
}
```

```
  | 0 1 2 3 4 5 6
0
1
2         X
3   +   X X +
4     0 0 0 0
5   + + X + + +
6

0
>>>   0 1 2 3 4 5 6
0
1         +
2     + + X +
3       X X
4   + X 0 0 0
5   X   X
6     + +

-3
  | 0 1 2 3 4 5 6
0
1       +
2     + 0 X
3       0 0 +
4     X 0 0 0 +
5   X   X +
6

2
>>>   0 1 2 3 4 5 6
0
1         + +
2       0 X +
3       0 0
4     X X X X X
5   X + X + + +
6 +     +
```

图 2: 纯行动力 AI 刚开局

```
  | 0 1 2 3 4 5 6
0
1           +
2 X X X X X 0 +
3     X X 0 X
4   X X 0 X X X
5   X 0 0 X
6   0 0 0 + X

-8
>>>   0 1 2 3 4 5 6
0
1 +   + + +
2 X X X X X 0 +
3 + + X X 0 X +
4 + X X 0 X X X
5 + X 0 X X + +
6   0 0 0 X X +

-18
  | 0 1 2 3 4 5 6
0
1           +
2 X X X X X 0 +
3     X X 0 X
4   X X 0 X X X
5   X 0 X X
6   0 0 0 0 0 0

22
>>>   0 1 2 3 4 5 6
0
1 +   + + +   +
2 X X X X X X X
3 + + X X 0 X +
4 + X X 0 X X X
5 + X 0 X X + +
6   0 0 0 0 0 0
```

图 3: 纯行动力 AI 中盘

```
100     | 0 1 2 3 4 5 6
101   0       + 0 +
102   1   + + + 0 + 0
103   2     0 0 0 0
104   3     X X X X X
105   4     + 0 X
106   5     + + +
107   6
108
109   9
110   >>>   0 1 2 3 4 5 6
111   0       + 0
112   1     + X 0   0
113   2   + 0 X X 0
114   3     X X X X X
115   4   + + 0 X +
116   5         + +
117   6
118
119   4
120     | 0 1 2 3 4 5 6
121   0     + 0 0 +
122   1   + + 0 0 + 0
123   2   + 0 0 X 0 +
124   3     X 0 X X X
125   4     + 0 X
126   5     +   +
127   6
128
129   18
130   >>>   0 1 2 3 4 5 6
131   0       0 0
132   1   X   0 0   0
133   2   + X 0 X 0
134   3   + X X X X X
135   4   +   0 X + +
136   5         +
137   6
```

图 4: 棋盘估值和行动力限制 AI 初始局面

```
320     | 0 1 2 3 4 5 6
321   0 0 0 0 0 0 0 0
322   1 0 0 0 X 0 0 0
323   2 0 0 X 0 0 0 +
324   3   X X X 0 0 X
325   4     X X X 0 +
326   5   X X X X
327   6     0   X
328
329   78
330   >>>   0 1 2 3 4 5 6
331   0 0 0 0 0 0 0 0
332   1 0 0 0 X 0 0 0
333   2 0 0 X X X X X
334   3 + X X X 0 X X
335   4 + + X X X 0 +
336   5   X X X X +
337   6 + + 0 + X +
338
339   62
340     | 0 1 2 3 4 5 6
341   0 0 0 0 0 0 0 0
342   1 0 0 0 X 0 0 0
343   2 0 0 X X X X X
344   3   X X X 0 X X
345   4     X 0 X 0 +
346   5   X 0 X X + +
347   6   0 0   X
348
349   74
350   >>>   0 1 2 3 4 5 6
351   0 0 0 0 0 0 0 0
352   1 0 0 0 X 0 0 0
353   2 0 0 X X X X X
354   3   X X X X X X
355   4 + + X 0 X X +
356   5 + X 0 X X + X
357   6 + 0 0 + X +
```

图 5: 棋盘估值和行动力限制 AI 后期局面