博弈

首先考虑两人参与的游戏：MAX 和 MIN，马上就会讨论这样命名的原因。MAX 先行，两人轮流出招，直到游戏结束。游戏结束时给优胜者加分，给失败者罚分。游戏可以形式化成含有下列组成部分的一类搜索问题。

- S_0：**初始状态**，规范游戏开始时的情况。
- PLAYER(s)：定义此时该谁行动。
- ACTIONS(s)：返回此状态下的合法移动集合。
- RESULT(s,a)：**转移模型**，定义行动的结果。
- TERMINAL-TEST(s)：**终止测试**，游戏结束返回真，否则返回假。游戏结束的状态称为**终止状态**。
- UTILITY(s,p)：**效用函数**（也可称为目标函数或收益函数），定义游戏者 p 在终止状态 s 下的数值。在国际象棋中，结果是赢、输或平，分别赋予数值+1，0，或 1/2。有些游戏可能有更多的结果，例如双陆棋的结果是从 0 到+192。**零和博弈**是指在同样的棋局实例中所有棋手的总收益都一样的情况。国际象棋是零和博弈，棋局的收益是 0+1，1+0 或1/2 +1/2 。“常量和”可能是更好的术语，但称为零和更传统，可以将这看成是下棋前每个棋手都被收了 1/2 的入场费。

初始状态、ACTIONS 函数和 RESULT 函数定义了游戏的**博弈树**——其中结点是状态，边是移动。图 5.1 给出了井字棋的部分博弈树。在初始状态 MAX 有九种可能的棋招。游戏轮流进行，MAX 下 X，MIN 下 O，直到到达了树的终止状态即一位棋手的标志占领一行、一列、一对角线或所有方格都被填满。叶结点上的数字是该终止状态对于 MAX 来说的效用值；值越高对 MAX 越有利，而对 MIN 则越不利（这也是棋手命名的原因）。

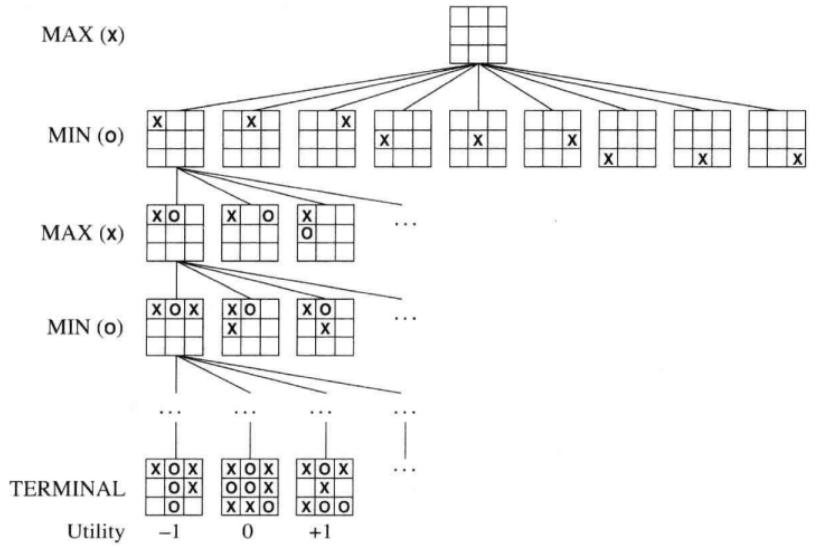


图 5.1　井字棋游戏的（部分）搜索树

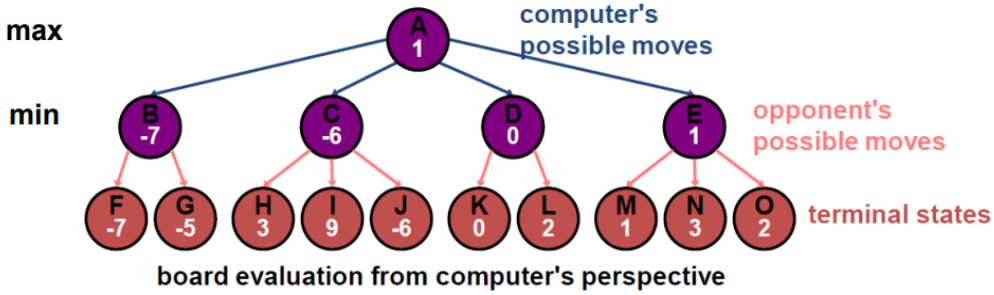
最上面的结点是初始状态，MAX 先走棋，放置一个 X 在空位上。图显示了搜索树的一部分，给出 MIN 和 MAX 的轮流走棋过程，直到到达终止状态，所有终止状态都按照游戏规则被赋予了效用值

极小极大值

给定一棵博弈树，最优策略可以通过检查每个结点的**极小极大值**来决定，记为 MINIMAX(n)。假设两个游戏者始终按照最优策略行棋，那么结点的极小极大值就是对应状态的效用值（对于 MAX 而言）。显然地，终止状态的极小极大值就是它的效用值自身。更进一步，对于给定的选择，MAX 喜欢移动到有极大值的状态，而 MIN 喜欢移动到有极小值的状态。所以得到如下公式：

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & s \text{ 为终止状态} \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & s \text{ 为 MAX 结点} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & s \text{ 为 MIN 结点} \end{cases}$$

- 假设两个玩家都以最大化自身利用进行博弈
- 举例：
  - 计算机假设在它移动后，对手会选择最小化的行动
  - 计算机在考虑自己的行动和对手的最佳行动后选择最佳行动



- 算法实现

```
function MINIMAX-DECISION(state) returns an action
   inputs: state, current state in game
   return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
   return v

function MIN-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← ∞
   for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
   return v
```

- 性能：

  极小极大算法对博弈树执行完整的深度优先探索。如果树的最大深度是 m，在每个结点合法的行棋有 b 个，那么极小极大算法的时间复杂度是 $O(b^m)$。一次性生成所有的后继的算法，空间复杂度是 $O(bm)$，而每次生成一个后继的算法（参见原书第 87 页），空间复杂度是 $O(m)$。当然对于真实的游戏，这样的时间开销完全不实用，不过此算法仍然可以作为对博弈进行数学分析和设计实用算法的基础。

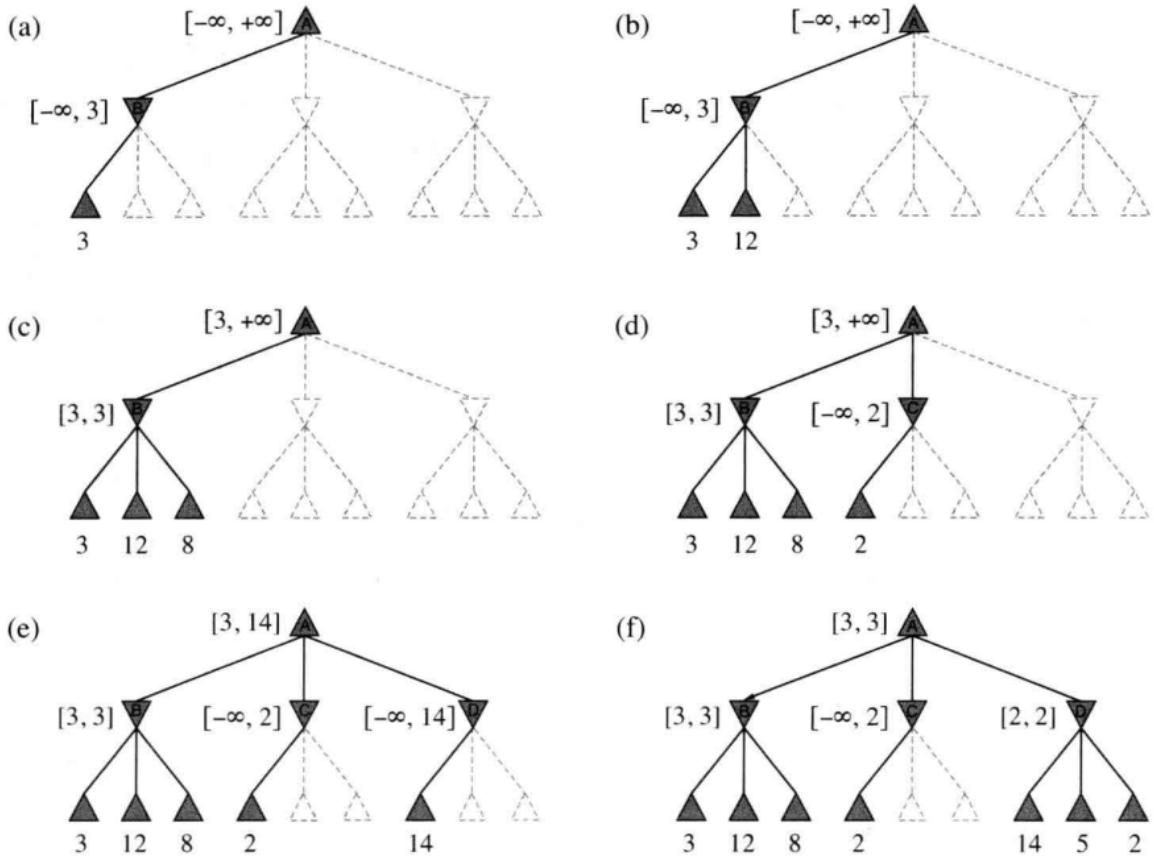
  对于国际象棋，b≈35，m≈100时，要得到精确解完全不可行。

  完备性(能得到问题的解)：可以，如果树是有限的

  最优解：可以，相对另一个对手而言

α-β剪枝

剪枝可以用来忽略一些分支
以下图为例：



在扩展完第一棵子树后，根结点能确定他能得到的收益最小为3，此时第二层为了选择一个最小值，但第一层会选择第二层所有值中的最大值，所以在d图中扩展了得到2已经小于3，所以剩下的 不用扩展，在e中，因为先扩展了14，大于3为了不让第一层得到14所以继续扩展，最后也变成2，这样第一层最多得到3

还可以把这个过程看作是对 MINIMAX 公式的简化。假设图　　中的 C 结点的两个没有计算的子结点的值是 x 和 y。根结点的值计算如下：

$$\begin{aligned}\text{MINIMAX}(root) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) \\ &= \max(3, z, 2) \qquad \text{其中 } z = \min(2, x, y) \leqslant 2 \\ &= 3\end{aligned}$$

记住极小极大搜索是深度优先的，所以任何时候只需考虑树中某条单一路径上的结点。α-β 剪枝的名称取自描述这条路径上的回传值的两个参数：

α = 到目前为止路径上发现的MAX的最佳（即极大值）选择

β = 到目前为止路径上发现的 MIN 的最佳（即极小值）选择

α-β搜索中不断更新α 和β 的值，并且当某个结点的值分别比目前的 MAX 的α 或者 MIN 的β 值更差的时候剪裁此结点剩下的分支(即终

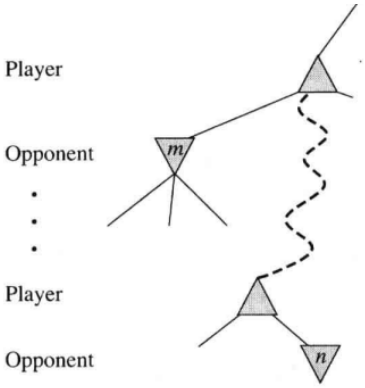


图　　α-β剪枝的一般情况
如果对选手而言 m 比 n 好，那么行棋就不会走到 n

- 算法实现：

```
function ALPHA-BETA-DECISION(state) returns an action
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  same as MAX-VALUE but with roles of α, β reversed
```

α-β剪枝的效率很大程度上依赖于检查后继状态的顺序。例如在图 5.5（e）和（f）中，根本不能剪掉 D 的任何后继（从 MIN 的角度），因为首先生成的是最差的后继。如果 D 的第三个后继先生成，就能够剪掉其他两个。这意味着应首先检查可能最好的后继。

如果能够这样做[1]，那么α-β算法只需检查 $O(b^{m/2})$个结点来做出决策（m 是对的最大深度），而不是极小极大算法的 $O(b^m)$。这意味着有效分支因子不是 b 而是 $\sqrt{b}$ ——对于国际象棋而言不是 35 而是 6。换种说法，在同样的时间里α-β算法比极小极大算法向前预测大约两倍的步数。如果后继状态采用随机顺序而不是最佳优先的顺序，那么要检查的总结点数大约是 $O(b^{3m/4})$。对于国际象棋，有一些相当简单的排序函数（如吃子优先，然后是威胁、前进、后退）可以使得检查的总结点数为 $O(b^{m/2})$的两倍。

截断测试

极小极大算法生成整个博弈的搜索空间，而α-β算法允许我们剪裁掉其中的一大部分。然而，α-β算法仍然要搜索部分空间直到终止状态。这样的搜索深度也是不现实的，因为要在合理的时间内确定行棋——典型地最多只有几分钟的时间来决策。Claude Shannon 发表论文《设计计算机国际象棋程序》（1950），文中提出应该尽早截断搜索，应将启发式**评估函数**用于搜索中的状态，有效地把非终止结点转变为终止结点。换言之，建议按两种方式对极小极大算法或α-β算法进行修改：用估计棋局效用值的启发式评估函数 EVAL 取代效用函数，用决策什么时候运用 EVAL 的**截断测试**取代终止测试。因此得到如下的启发式极小极大值，s 为状态，d 为最大深度：

$$\text{H-MINIMAX}(s,d)=\begin{cases}\text{EVAL}(s) & \text{如果 CUTOFF-TEST}(s,d)\text{为真}\\ \max_{a\in Actions(s)}\text{H-MINIMAX}(\text{RESULT}(s,a),d+1) & s\text{ 为 MAX 结点}\\ \min_{a\in Actions(s)}\text{H-MINIMAX}(\text{RESULT}(s,a),d+1) & s\text{ 为 MIN 结点}\end{cases}$$

- **用可以估计棋局效用值的启发式评价函数EVAL取代效用函数**