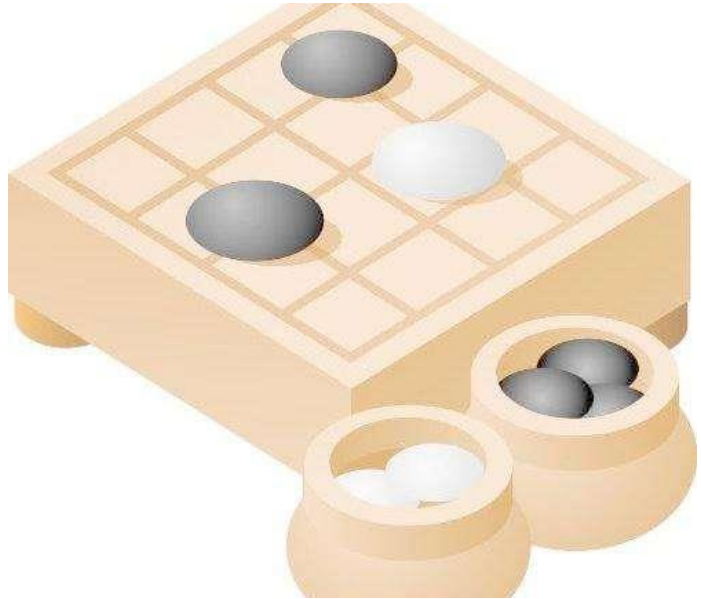
人工智能

博弈与搜索

主要内容

1. 博弈搜索
2. 极大极小算法
3. α-β剪枝算法
4. 当前研究进展



4.1 概述

- 自从有了计算机以后，人们开始就有用计算机下棋的想法，早在60年代就已经出现若干博弈程序，并达到较高的水平，目前计算机博弈程序已能够与人类博弈大师抗衡。
  - 1997年深蓝（deep blue）计算机与国际象棋大师卡斯帕罗夫对弈，三胜一负。
  - 2016年Alpha Go以4胜1负的战绩战胜韩国著名围棋棋手李世石。
  - 2017年Alpha Zero以100：0的成绩战胜了它的前代Alpha Go。

从AlphaGo到AlphaGo Zero及AlphaZero

- AlphaGo Zero: 不需要任何人类棋局

- AlphaZero 拥有更强大的泛化能力，经过不到 24 小时的训练后，Alpha Zero 可以在国际象棋和日本将棋上击败目前业内顶尖的 AI 程序（这些程序早已超越人类世界冠军水平），也可以轻松击败训练 3 天时间的 AlphaGo Zero

4.1 概述

计算机博弈的本质是人工智能的问题求解。

博弈是富有挑战性的智力游戏：“双人有限零和顺序游戏”

游戏：意味着我们在一种需要交互的情境中，交互通常会涉及一个或多个角色

有限：表明在任意时间点，角色之间存在的交互方式都是有限的

双人：游戏中只有两个角色

顺序：玩家依次交替进行他们的动作

零和：参与游戏的两方有完全相反的目标，即，游戏的任意结束状态双方的收益之和等于零

4 博弈与搜索

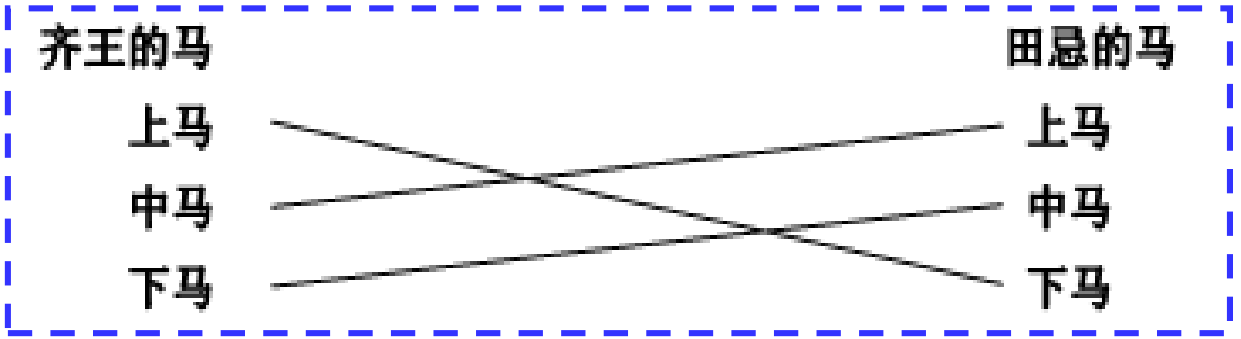
4.1 概述

参与博弈的各方，在严格竞争下，一方的收益必然意味着另一方的损失，博弈各方的收益和损失相加总和永远为“零”，双方不存在合作的可能。

博弈问题常与对策问题联系在一起。对策论（**Game Theory**）用数字方法研究对策问题。一般将对策问题分为**零和**对策和**非零和**对策。

最典型的**零和对策**问题：田忌赛马问题

- 齐王与田忌都有上、中、下三匹马
- 齐王的上马、中马、下马都比田忌相应的上马、中马、下马好
- 田忌的上马比齐王的中马好，田忌的中马比齐王的下马好
- 田忌如何取胜?



4 博弈与搜索

4.1 概述

非零和博弈是一种合作下的博弈，博弈中各方的收益或损失的总和不是零值，自己的所得并不与他人的损失的大小相等，即使伤害他人也可能“损人不利己”，博弈双方存在“双赢”的可能，进而达成合作。

非零和对策的例子：囚犯难题（The prisoner dilemma）

美国经济学家Nash将求解囚犯难题的方法用于解决经济模型中的问题，获得诺贝尔奖。

- 有两个嫌疑犯A和B，没有他们犯罪的确定的证据。
- 现对他们判刑的规则是：
- A、B如何决策?

判刑 A \ B	不承认	承认
不承认	各判 1 年	A 判 10 年 B 判 3 个月
承认	A 判 3 个月 B 判 10 年	各判 6 年

4.1 概述

什么是博弈搜索?

每一个角色在做出决策时，不仅要考虑到自己的立场，还要预测对手可能的反应，由此构成博弈状态空间，在其中进行搜索。

博弈问题对人的深层次的知识研究提出了严峻的挑战:

1. 如何表示博弈问题的状态
2. 博弈过程和博弈取胜的知识

这是目前人类仍在探讨之中的问题。

4.1 概述

要提高博弈问题求解程序的效率，应作到如下两点：

- 改进生成过程，使之只生成好的走步，如按棋谱的方法生成下一步；
- 改进测试过程，使最好的步骤能够及时被确认。

要达到上述目的有效途径是使用启发式方法引导搜索过程，这样的博弈程序应具备：

- 一个好的（即只产生可能赢棋步骤的）生成过程。
- 一个好的静态估计函数。

4.2 极小极大搜索算法

（1）极小极大搜索的思想

- 极小极大搜索策略是考虑双方对弈若干步之后，从可能的步中选一步相对好的走步，即在有限的搜索深度范围内进行求解。
- 为此要定义一个静态估计函数f，以便对棋局的态势作出优劣估计。这个函数可根据棋局优劣态势的特征来定义。

4.2 极小极大搜索算法

博弈双方的目标量化：

- MAX方，程序方，我方目标是使每步得分最大
- MIN方，对手方，他方目标是最大化他的得分，相当于最小化我方得分；
- P 代表一个棋局（即一个状态）

f(P)由棋局态势的优劣来决定：

- 有利于MAX的态势，f(P)取正值
- 有利于MIN的态势，f(P)取负值
- 态势均衡，f(P)取零

4.2 极小极大搜索算法

使用静态函数进行估计必须以下述两个条件为前提：

1）双方都知道各自走到什么程度、下一步可能做什么。

2）不考虑偶然因素影响。

在这个前提下，博弈双方必须考虑：

1）如何产生一个最好的走步。

2）如何改进测试方法，能尽快搜索到最好的走步。

4.2 极小极大搜索算法

MINMAX的基本思想是：

1）当轮到MIN走步的结点时，MAX应考虑最坏的情况（因此，f(p)取极小值）。

2）当轮到MAX走步的结点时，MAX应考虑最好的情况（因此，f(p)取极大值）。

3）当评价往回倒推时，相应于两位棋手的对抗策略，不同层上交替的使用（1）、（2）两种方法向上传递倒推值。

4.2 极小极大搜索算法

(2) 极小极大搜索算法

1. T:=(*s*, MAX), OPEN:=(*s*), CLOSED:=( );
   {开始时树由初始结点构成，OPEN表只含有s.}
2. **LOOP1**: **IF** OPEN = ( ) **THEN** GO LOOP2;
3. *n*:=FIRST(OPEN),REMOVE(*n*, OPEN),
   ADD_TO_LAST(*n*, CLOSED); //约定加到尾部

4.2 极小极大搜索算法

4. **IF** *n*可直接判定为赢、输或平局 **THEN** f(*n*):= ∞∨-∞∨0, GO **LOOP1**

ELSE EXPAND(*n*)→n_i , ADD ({n_i}, T)

IF d({*n*}) < k **THEN** ADD_TO_LAST ({n_i}, OPEN), GO **LOOP1**

ELSE 计算f(n_i), GO **LOOP1**; { *n*达到深度k,计算各端结点f值}

5. **LOOP2**: **IF** CLOSED=NIL **THEN** GO LOOP3

ELSE n_p:= LAST(CLOSED);

//CLOSE表是后进先出的表，每次选CLOSE的尾巴节点

4.2 极小极大搜索算法

6. **IF** ($n_p \in MAX$)∧(f($n_{ci} \in MIN$)有值)　(其中n_{ci}为n_p的下一层节点)

THEN f(n_p):=MAX{ f(n_{ci})}, REMOVE(n_p,CLOSED);

{若MAX所有子节点均有值，则该MAX取其极大值。}

IF ($n_p \in MIN$)∧(f($n_{ci} \in MAX$)有值)

THEN f(n_p):=MIN{ f(n_{ci})}, REMOVE(n_p,CLOSED);

{若MIN所有子节点均有值，则该MIN取其极小值。}

4.2 极小极大搜索算法

7. GO **LOOP2**;

8. **LOOP3**: **IF** f(s)≠NIL **THEN** EXIT(END ∨ Mark(Move, T));

{s有值，或结束或标记走步}

其中ADD_TO_LAST约定加入节点到表的尾部，END表示失败或成功或平局退出，MARK标记一个走步。

4.2 极小极大搜索算法

该算法分三个阶段进行

- 第一阶段为步骤2 - 4，使用**宽度优先法生成规定深度的全部博弈树**，然后对其所有端节点**计算其静态估计函数值。**

- 第二阶段为步骤5 - 7，从底向上逐级求**非终结点的倒推估计值，直到求出初始节点的倒推值f(s)为止。**f(s)的值应为max min.... {f($n_{i1i2i3...ik}$)}，其中n_{ik}表示深度为k的端节点。

- 第三阶段，根据f(s)可选的相对好的走步，由Mark (Move, T)标记走步。

4.2 极小极大搜索算法

(3) 算法分析与举例

【例5-1】 在九宫格棋盘上两位选手轮流在棋盘上摆各自的棋子，每次一枚，谁先取得三子一线的结果就取胜。

- 设程序方MAX的棋子用X表示
- 对手方MIN的棋子用O表示

X_5	O_2	O_4
X_7	X_1	
X_3		O_6

4.2 极小极大搜索算法

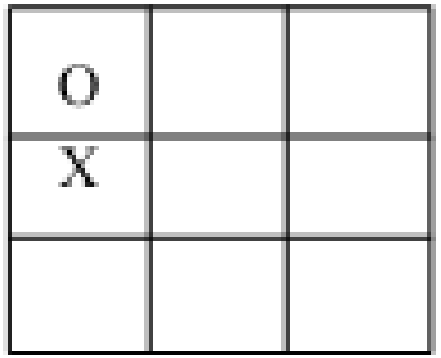
静态估计函数为:

$$f(p) = \begin{cases} +\infty & \text{当p为MAX赢} \\ -\infty & \text{当p为MIN赢} \end{cases}$$

(全部空格放X后三子成一线的总数 -全部空格放O后三子成一线的总数)

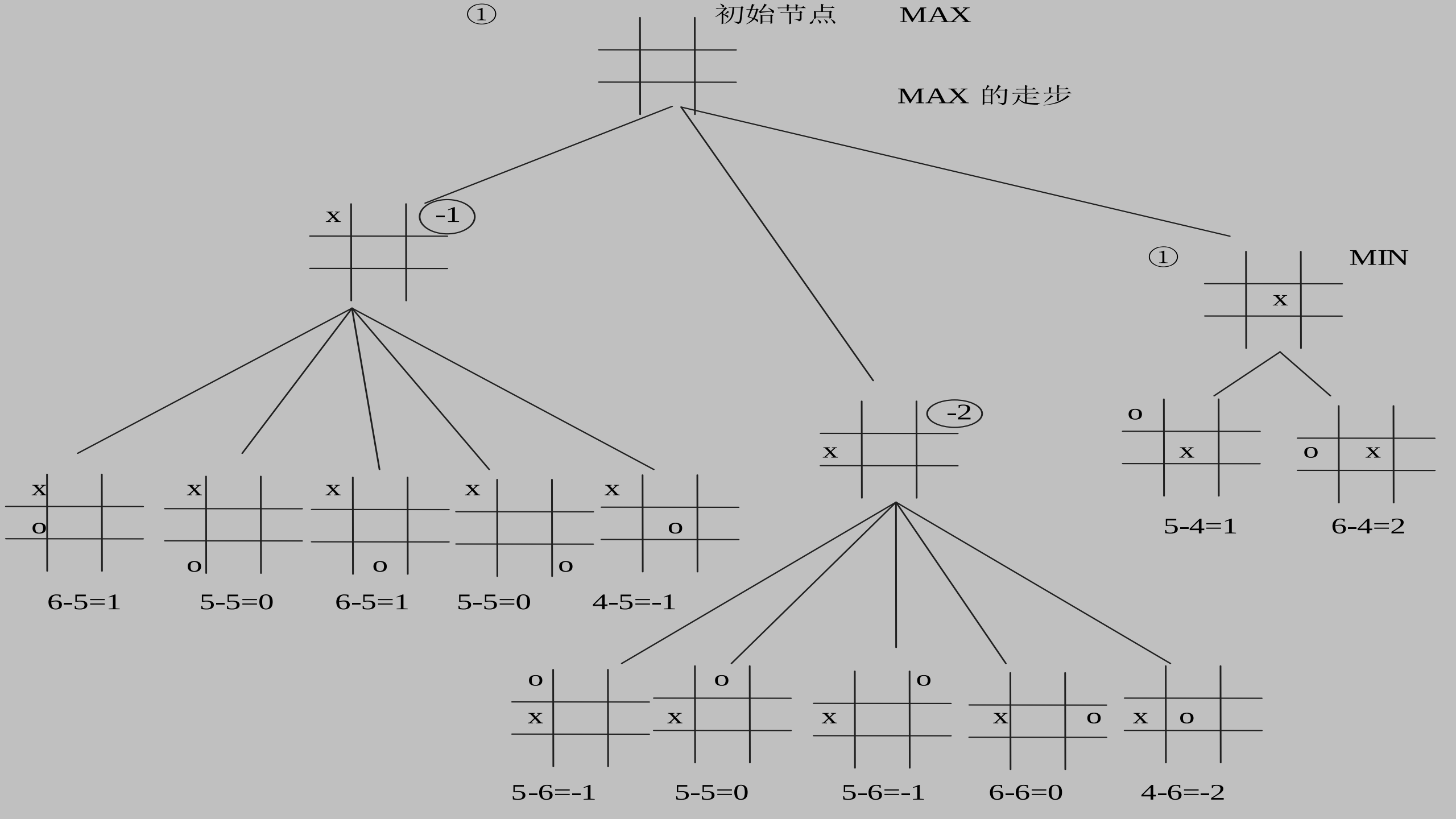
4.2 极小极大搜索算法

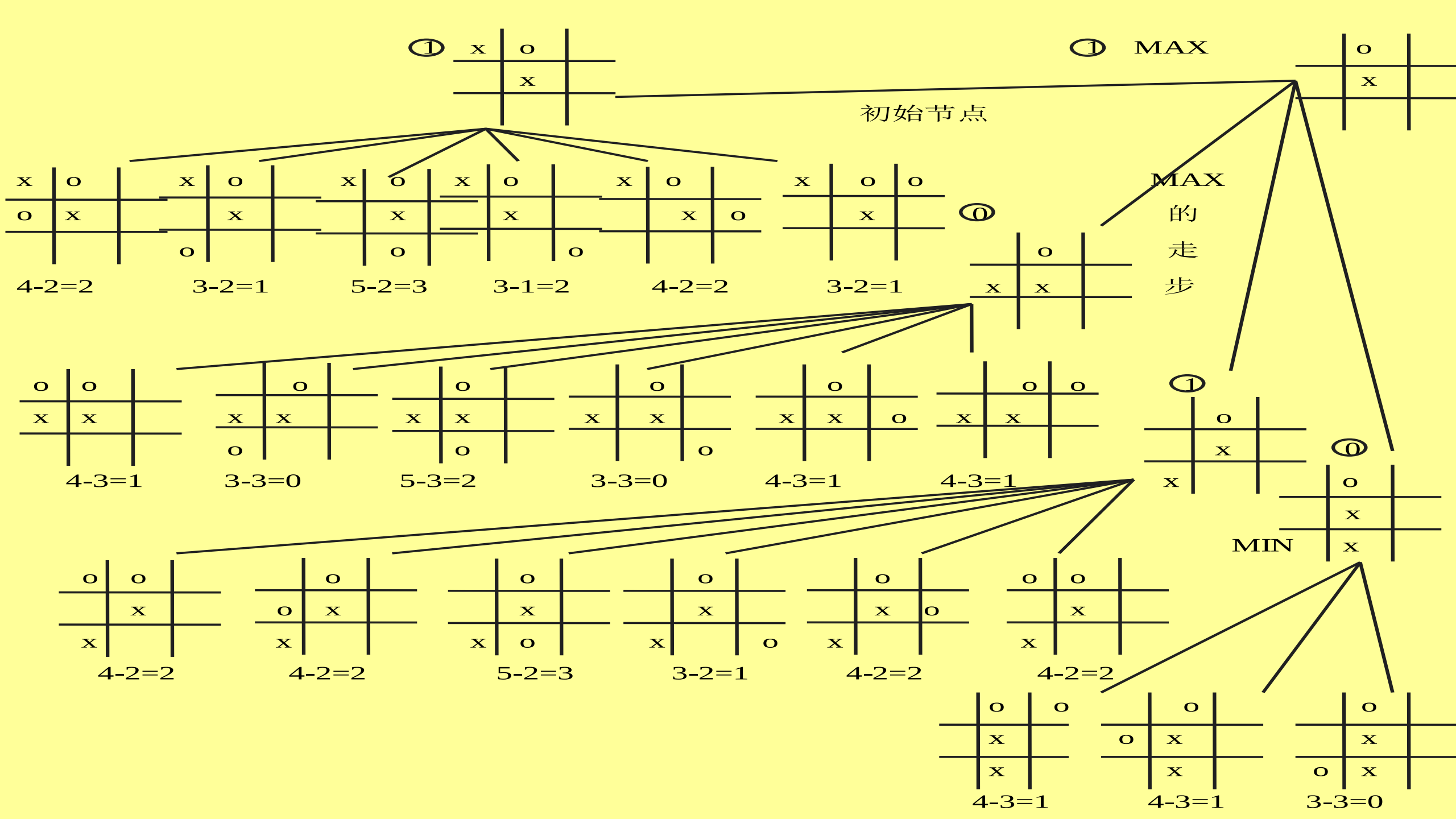
例如，P的格局为

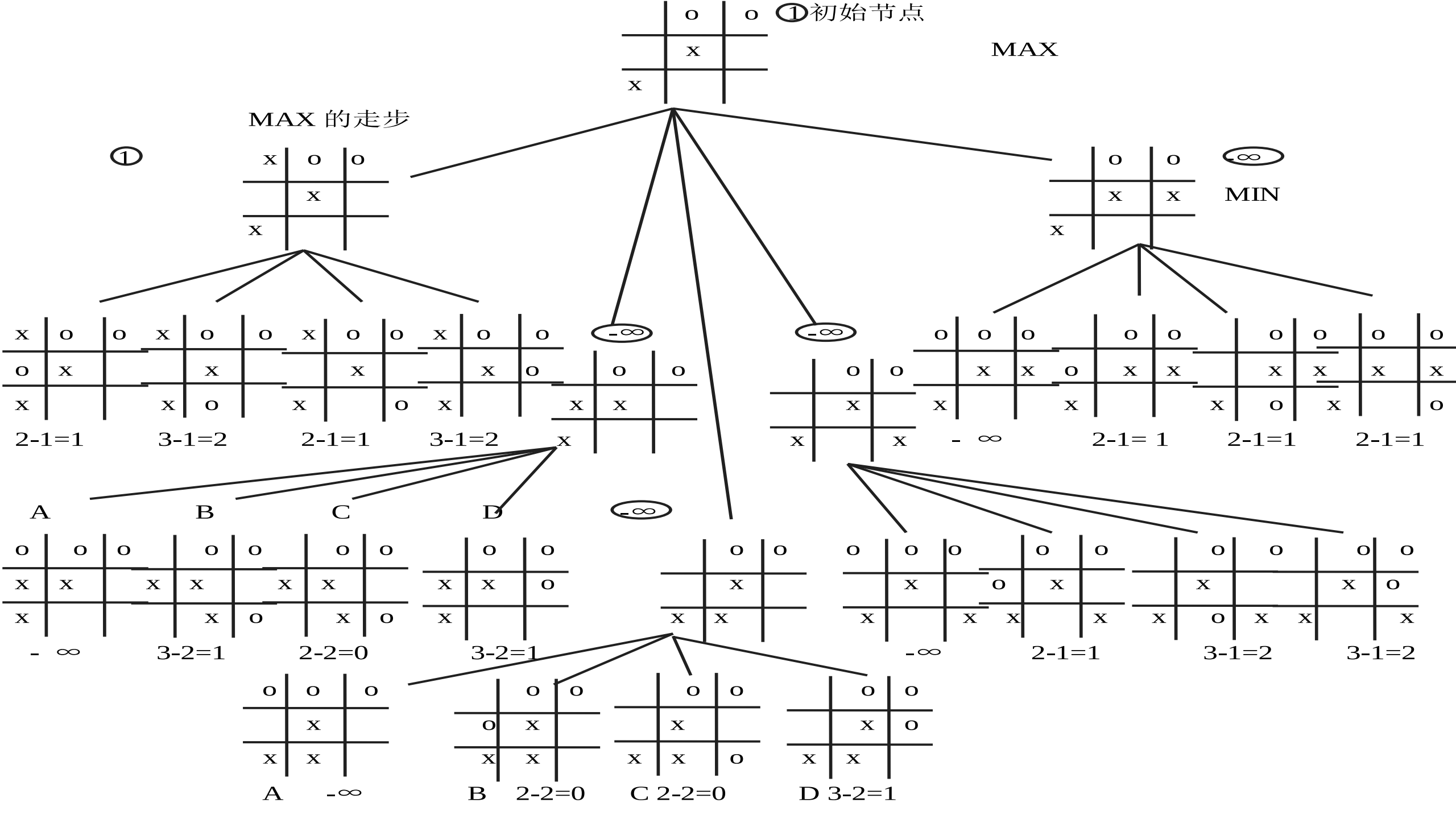


则可得f(p)=5 – 6 = -1。

现在考虑走两步的搜索过程，即算法中K=2。利用棋盘对称性条件，则MAX走第一步棋调用算法产生搜索树如图所示。


①
初始节点
MAX
MAX 的走步
x
-1
x
o
6-5=1
x
o
5-5=0
x
o
6-5=1
x
o
5-5=0
x
o
4-5=-1
-2
x
o
x
o
5-6=-1
o
x
5-5=0
o
x
5-6=-1
o
x
o
6-6=0
x
o
4-6=-2
①
MIN
x
o
x
5-4=1
o
x
6-4=2


①
初始节点
①
MAX
MAX
的
走
步
0
MIN
4-2=2
3-2=1
5-2=3
3-1=2
4-2=2
3-2=1
4-3=1
3-3=0
5-3=2
3-3=0
4-3=1
4-3=1
4-2=2
4-2=2
5-2=3
3-2=1
4-2=2
4-2=2
4-3=1
4-3=1
3-3=0


①初始节点
MAX
MAX 的走步
①
MIN
-∞
-∞
-∞
-∞
2-1=1
3-1=2
2-1=1
3-1=2
- ∞
2-1= 1
2-1=1
2-1=1
A
B
C
D
- ∞
3-2=1
2-2=0
3-2=1
-∞
2-1=1
3-1=2
3-1=2
A -∞
B 2-2=0
C 2-2=0
D 3-2=1

再理解

MINMAX的基本思想是：

1）当轮到MIN走步的结点时，MAX应考虑最坏的情况（因此，f(p)取极小值）。

2）当轮到MAX走步的结点时，MAX应考虑最好的情况（因此，f(p)取极大值）。

3）当评价往回倒推时，相应于两位棋手的对抗策略，不同层上交替的使用（1）、（2）两种方法向上传递倒推值。

再理解

MINMAX的基本过程：

- 第一阶段为步骤2－4，使用**宽度优先法生成规定深度的全部博弈树**，然后对其所有端节点**计算其静态估计函数值。**

- 第二阶段为步骤5－7，从底向上逐级求**非终结点的倒推估计值，直到求出初始节点的倒推值f(s)为止。**

- 第三阶段，根据f(s)可选的相对好的走步，标记走步。

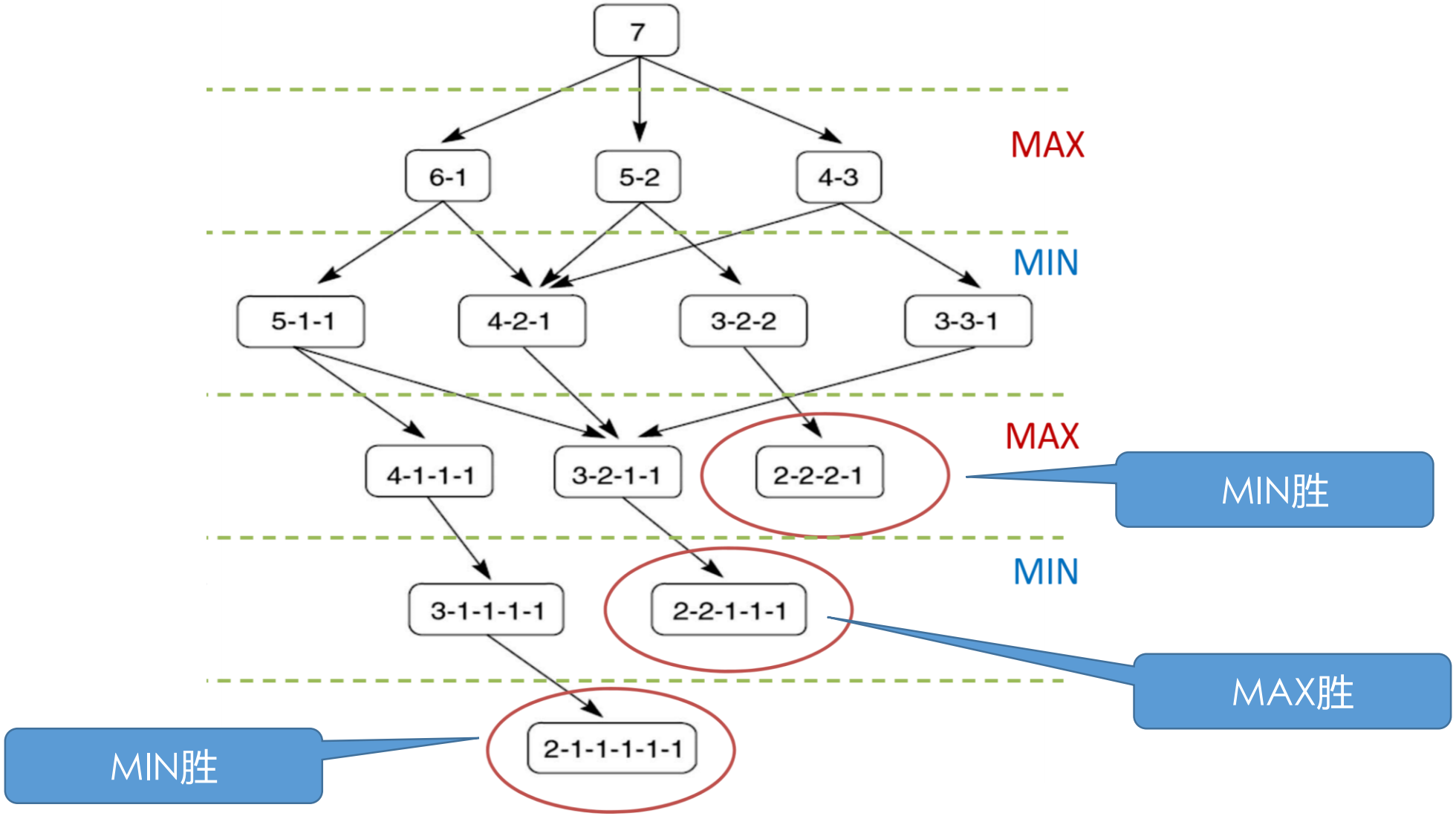
4.2 极小极大搜索算法

例：“余一棋”游戏。两个对弈者参与这个游戏：
桌子上放着一堆牌，双方交替行棋。
每一步，行棋方必须把牌中的某一摞分成两摞不等数量的牌，如一摞6张牌可以被分为1，5或者2，4，但不能分为3，3。直至某一方无法继续下去，为败者。

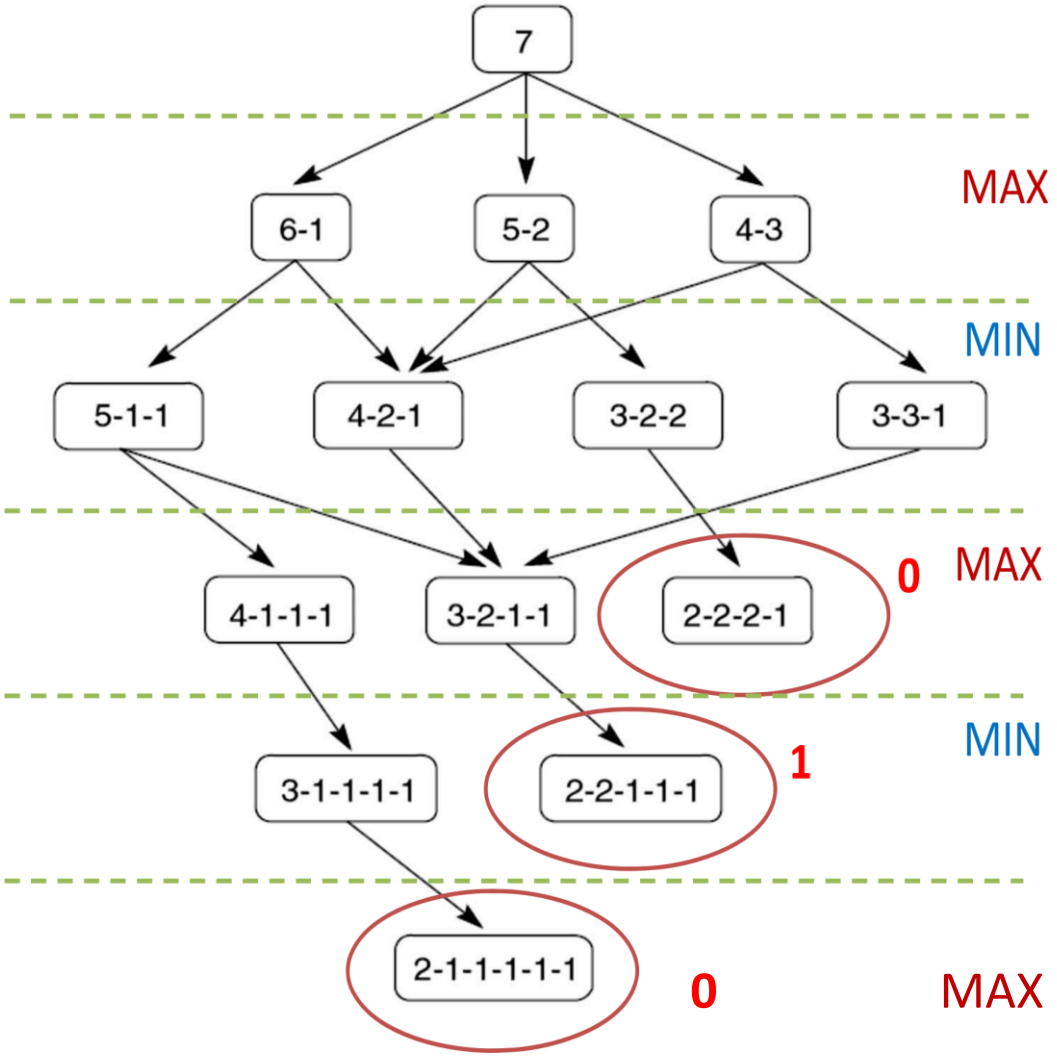
假设 开始一共有1摞共7张牌，由MIN先行。
问：MAX如何行棋能胜利?

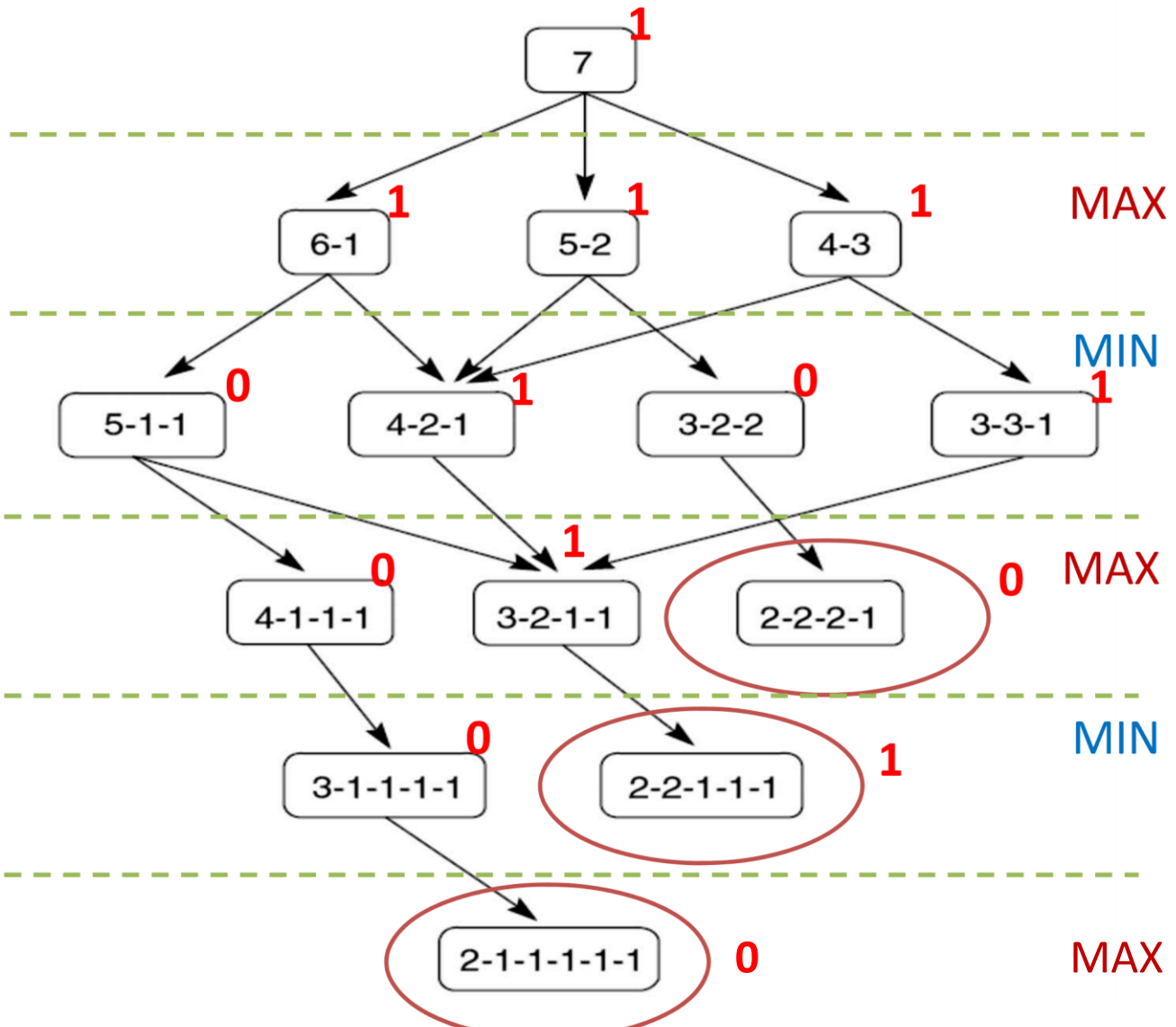
MOOC 第5.5博弈搜索

4 博弈与搜索


7
MAX
6-1
5-2
4-3
MIN
5-1-1
4-2-1
3-2-2
3-3-1
MAX
4-1-1-1
3-2-1-1
2-2-2-1
MIN胜
MIN
3-1-1-1-1
2-2-1-1-1
MAX胜
2-1-1-1-1-1
MIN胜

- 设计得分：MAX胜的节点得分为1；MIN胜的节点得分为0
- 则下面可以根据规则，倒推所有节点的得分：

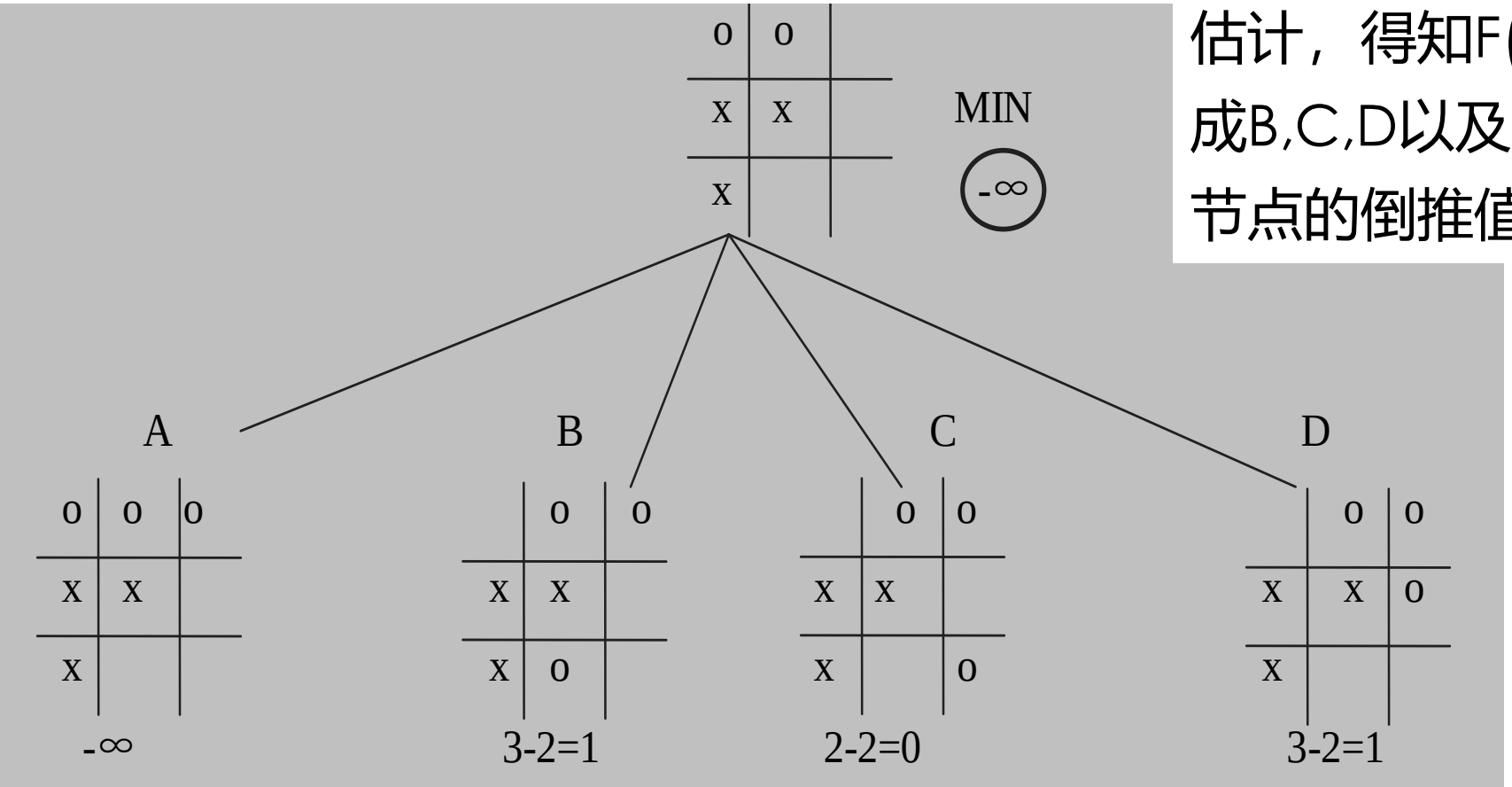




极小极大算法（缺点）

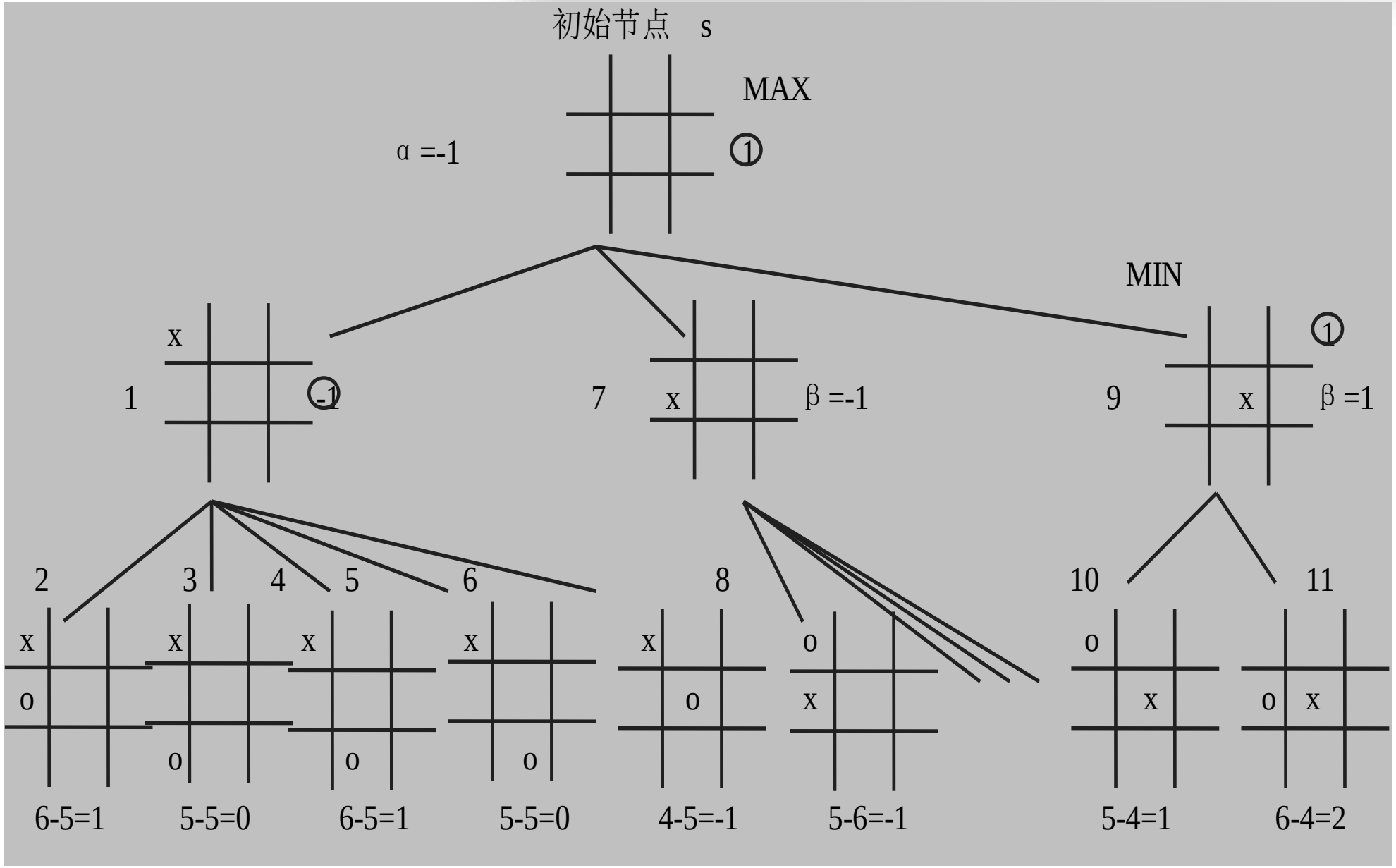
- 一个MIN节点要生成A,B,C,D四个节点
- 然后逐个计算其静态估计值
- 最后求得倒推值-∞，把它赋给这个结点。

考虑：生成节点A后，如果马上进行静态估计，得知F(A)= -∞之后，就可以断定生成B,C,D以及进行估计是多余的，该MIN节点的倒推值一定是-∞。



4.3 α-β剪枝算法

- α-β剪枝法就是把生成后继和倒推值估计结合起来，及时剪掉一些无用分枝，以此来提高算法的效率。
- α-β剪枝法，采用有界深度优先策略进行搜索，当生成节点达到规定的深度时，就立即进行静态估计，而一旦某个非端节点有条件确定倒推值时，就立即赋值。


初始节点 s
MAX
α =-1
①
MIN
x
1
-1
7
x
β =-1
①
9
x
β =1
2
3
4
5
6
8
10
11
o
6-5=1
5-5=0
6-5=1
5-5=0
4-5=-1
5-6=-1
5-4=1
6-4=2

4.3 α-β剪枝算法

- 当生成到节点6后，节点1的倒推值可确定为-1。
- 对于初始节点S来说，虽然其他子节点尚未生成，但由于S属于极大层，所以可以推断它的倒推值不会小于-1。
- 定义：极大层的这个**下界**值为α。因此S的α= - 1。
  - S的α值为-1，说明的S倒推值不会比-1更小，但会不会比-1更大，还取决于其他后继节点倒推值。我们继续生成搜索树。

4.3 α-β剪枝算法

- 当第8个节点生成后，其估计值为-1，就可以断定节点7的倒推值不可能大于-1。
- 定义：极小层的这个**上界**值为β 。因此现在可以确定节点7的β= - 1。
- 容易发现α≥β时，节点7的其他子节点不必生成，因为S的极大值不可能比这个β值小，再生成无疑是多余的，因此可以进行剪枝。

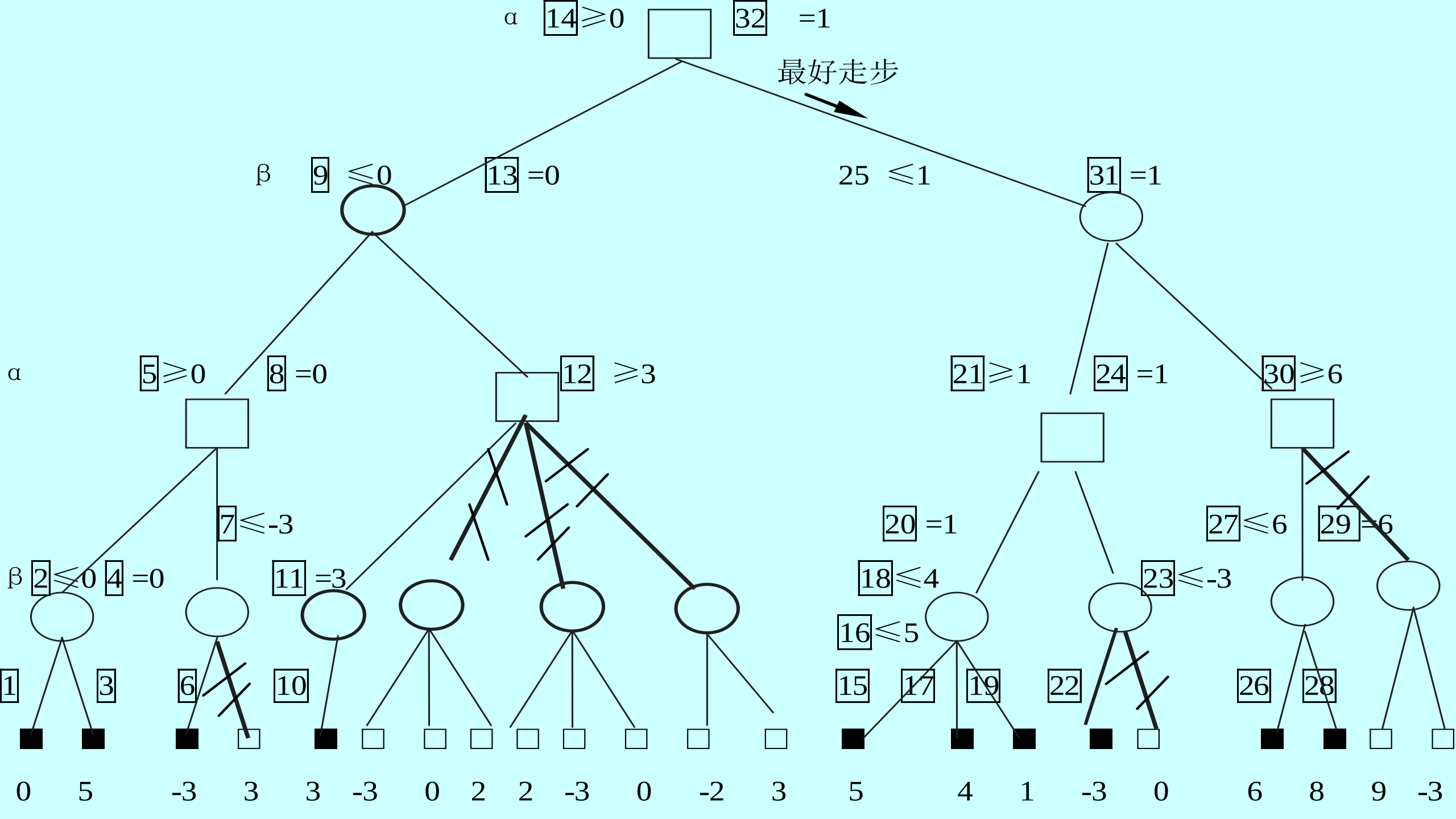
4.3 α-β剪枝算法

- 只要在搜索过程中记住倒推值的上下界并进行比较，当α≥β时就可以实现修剪操作。
- α,β值还可以随时修正，但
  - 极大层的α倒推值下界永不下降，因为实际的倒推值取后继节点最终确定的倒推值中的最大者。 α↑
  - 极小层的倒推值上界β永不上升，因为实际倒推值取后继节点最终确定的倒推值中的最小者。 β↓

α-β过程的剪枝规则

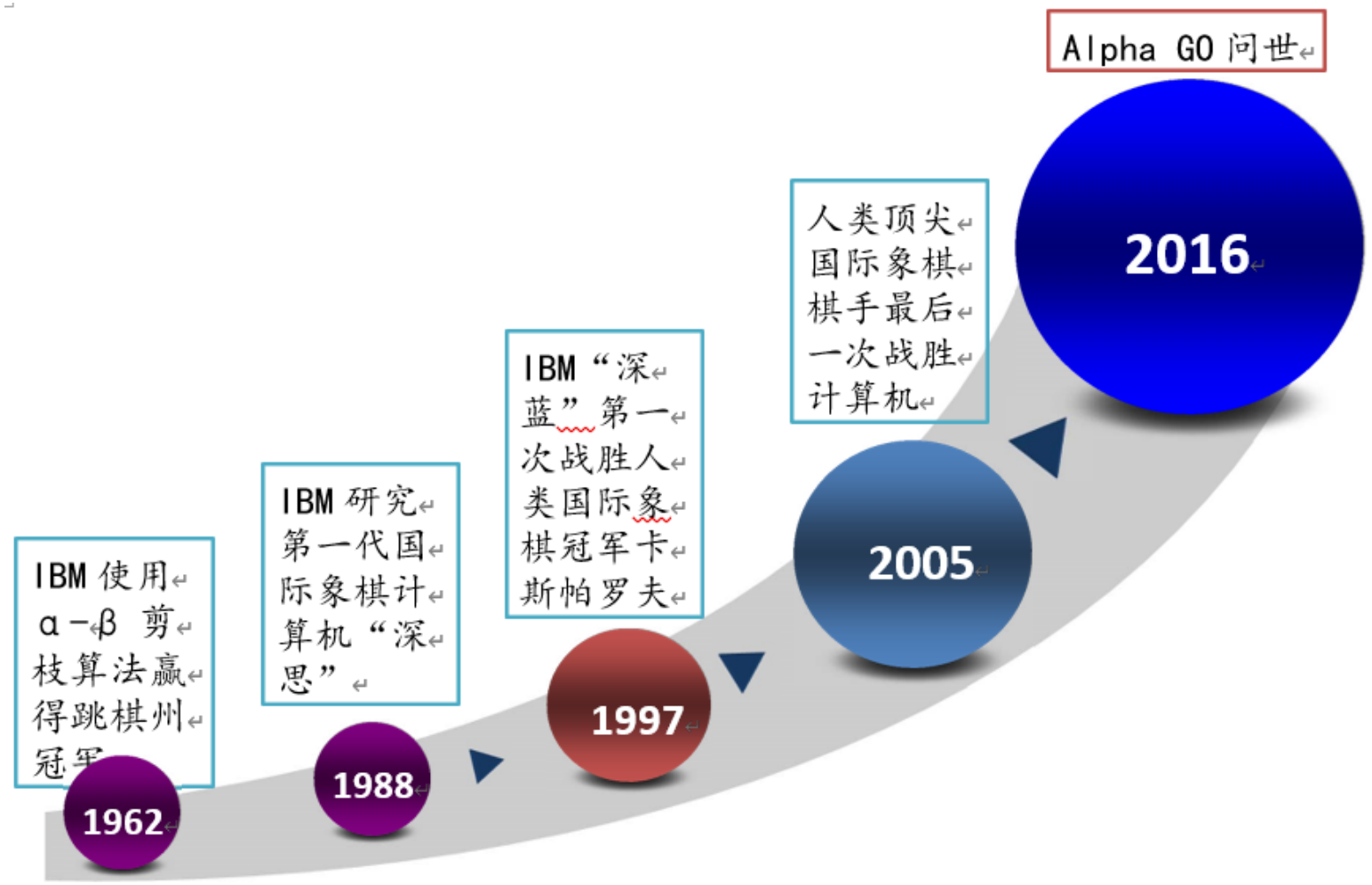
（1）**α剪枝**：若任一极小值层节点的β值小于或等于它任一先辈极大值层节点的α值，即α（先辈层）≥β（后继层），则可中止该极小值层中这个节点以下的搜索。该节点最终的倒推值就确定为这个β值。

（2）**β剪枝**：若任一极大值层节点的α值大于或等于它任一先辈极小值层节点的β值，即α（后继层）≥β（先辈层），则可以中止该极大值层中这个节点以下的搜索。这个MAX节点的最终倒推值就确定为这个α值。


α
14 ⩾0
32 =1
最好走步
β
9 ⩽0
13 =0
25 ⩽1
31 =1
α
5 ⩾0
8 =0
12 ⩾3
21 ⩾1
24 =1
30 ⩾6
7 ⩽-3
20 =1
27 ⩽6
29 =6
β
2 ⩽0
4 =0
11 =3
18 ⩽4
23 ⩽-3
16 ⩽5
1
3
6
10
15
17
19
22
26
28
0
5
-3
3
3
-3
0
2
2
-3
0
-2
3
5
4
1
-3
0
6
8
9
-3

4 博弈与搜索

当前进展



α-β剪枝算法在很长一段时间里都是棋类算法的代表，取得了一系列的成就。

从国际象棋到围棋

- 但在围棋领域，由于胜负状态太复杂，无法找到有效的启发方式，即使经过α－β剪枝，也很难实现搜索，一直无法突破。
- 2006年，法国一个团队提出使用“蒙特卡洛树搜索”方式，以“信心上限决策”打分，使计算机围棋能力得到质的提升。为后续方法奠定基础。

- 2013年，CrazyStone程序达到人类业余5段水平
- 2016年，Alpha Go问世，将深度学习、价值网络、蒙特卡洛树搜索技术融合，战胜人类顶尖棋手。

蒙特卡洛树

- Alpha Go/Zero是几种方法集成而成的一个工程杰作。Alpha Go/Zero的核心组件包括：
  - 蒙特卡洛树搜索——使用PUCT函数的一种树遍历的特定变体
  - 残差卷积神经网络——使用policy network（策略网络）和value network（价值网络）来进行比赛的评估和落子先验概率的估计
  - 强化学习——通过自我对局来训练网络

https://blog.csdn.net/qq_16137569/article/details/83543641

蒙特卡洛树

- 基本概念

蒙特卡洛树搜索算法中，最优行动会通过一种新颖的方式计算出来。

蒙特卡洛树搜索会多次模拟博弈，并尝试根据模拟结果预测最优的移动方案。

蒙特卡洛树

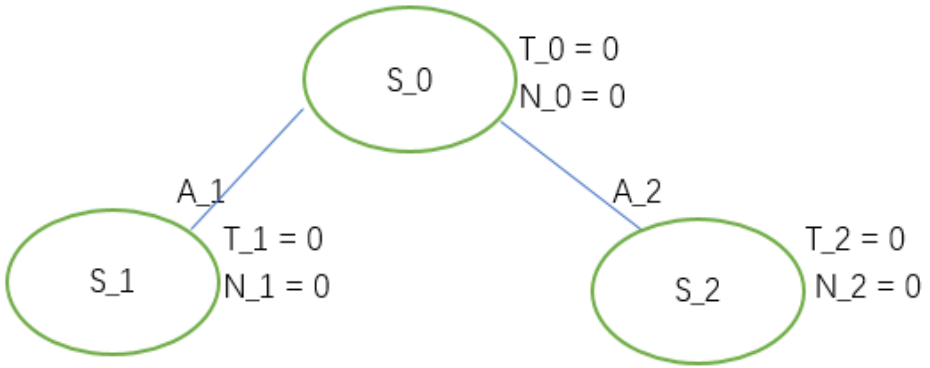
- **原理**
  - 蒙特卡洛树搜索的主要概念是搜索，即沿着博弈树向下的一组遍历过程。
  - 单次遍历的路径会从根节点（当前博弈状态）延伸到没有完全展开的节点，未完全展开的节点表示其子节点至少有一个未访问到。
  - 遇到未完全展开的节点时，它的一个未访问子节点将会作为单次模拟的根节点，随后模拟的结果将会反向传播回当前树的根节点并更新博弈树的节点统计数据。
  - 一旦搜索受限于时间或计算力而终止，下一步行动将基于收集到的统计数据进行决策。

MCTS举例

树的初始状态：

T 表示总的 value，N 表示被访问的次数（visit count）。

A表示动作（action）



算法步骤：

1. Tree traversal:

   $$UCB1(S_i) = \bar{V}_i + c\sqrt{\frac{\log N}{n_i}}, c$$

   其中$\bar{V}_i$表示S_i状态的平均value
2. Node expansion
3. Rollout (random simulation)
4. Backpropagation

MCTS举例

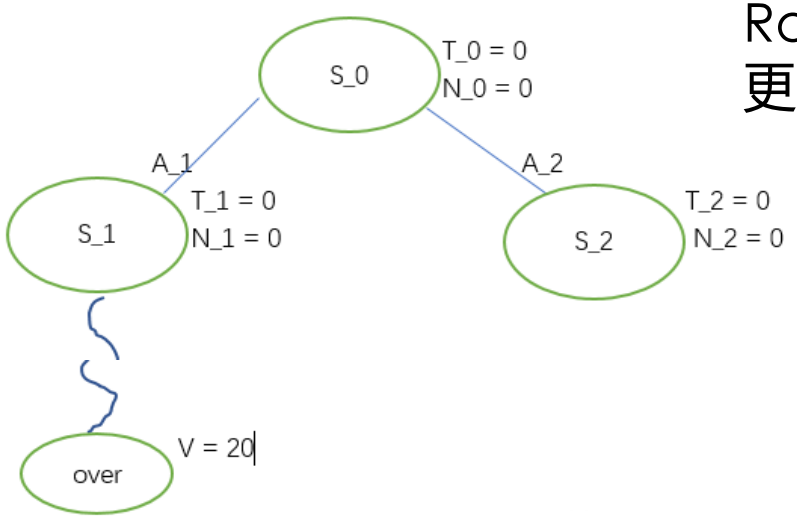
第一次迭代（iteration）

从状态S0开始，要在下面两个动作中进行选择（假设只有两个动作可选），选择的标准就是UCB1(Si) 值。显然可算得：UCB1(S1)=UCB1(S2)=∞

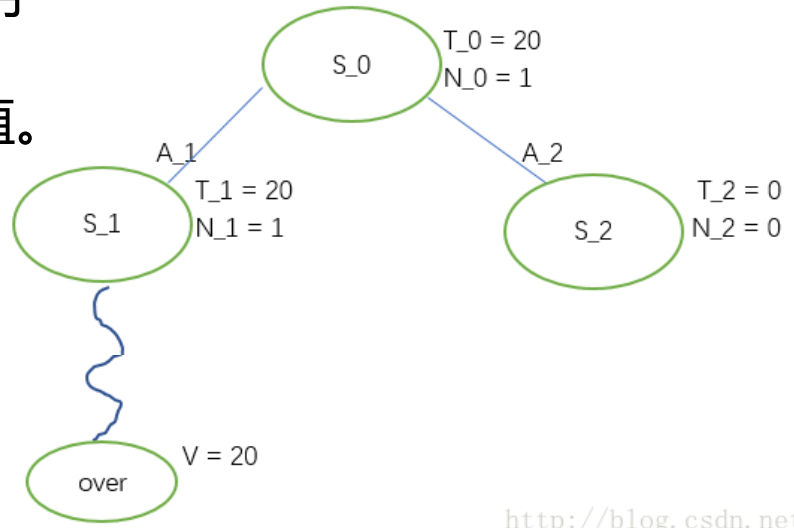
这种情况下，我们就按顺序取第一个，即A1。从而，达到状态S1。

按照步骤1，2的流程图，我们现在需要判断目前的结点S1（current node)是不是叶节点，这里叶节点是指其没有被展开（expansion）过。显然，此结点没有被展开过，所以是叶节点。接下来，按照流程图，需要判断结点S1被访问的系数是否为0。是0，所以要进行Rollout。

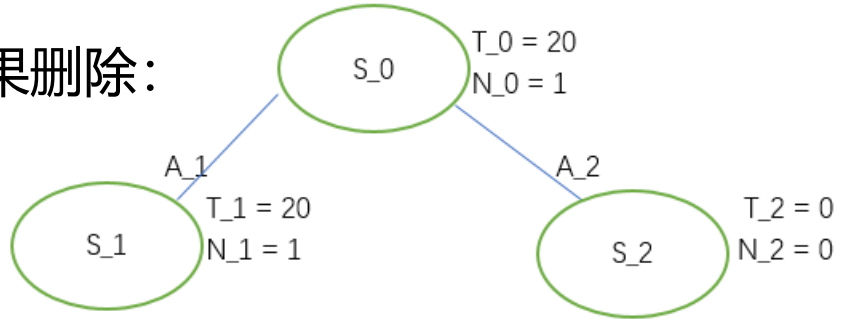
MCTS举例



接下来，进行步骤4 Backpropagation，即利用Rollout最终得到的value来更新路径上每个结点的T,N值。

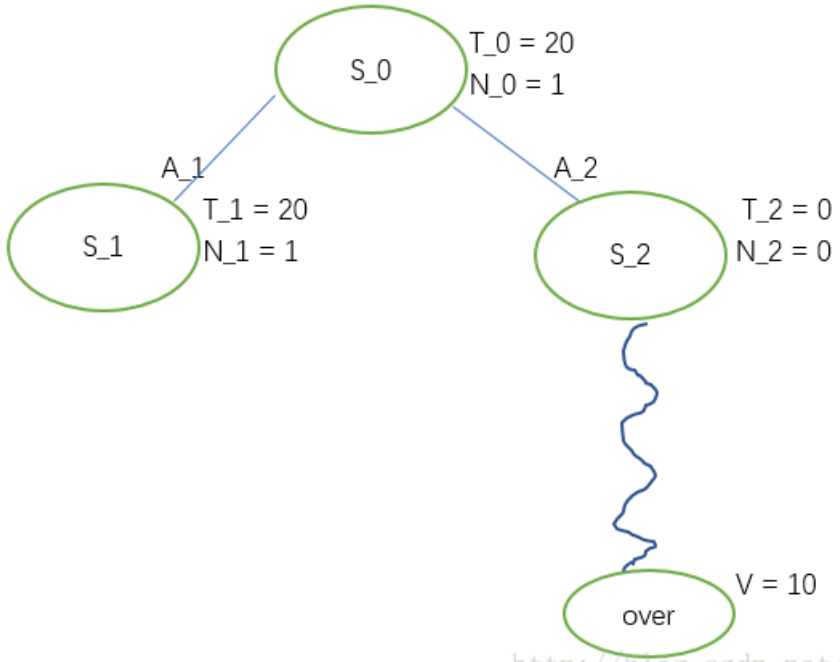


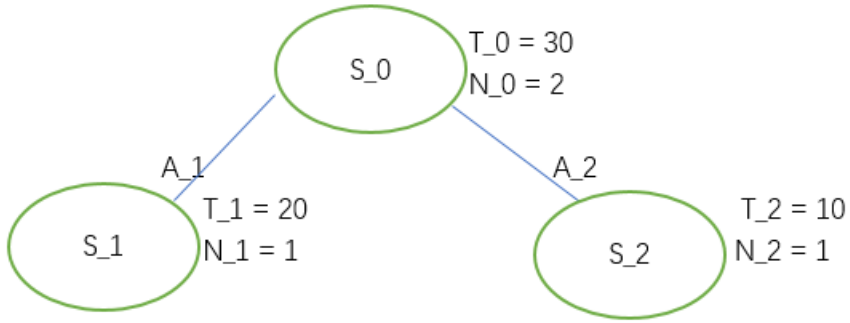
之后把Rollout的结果删除：



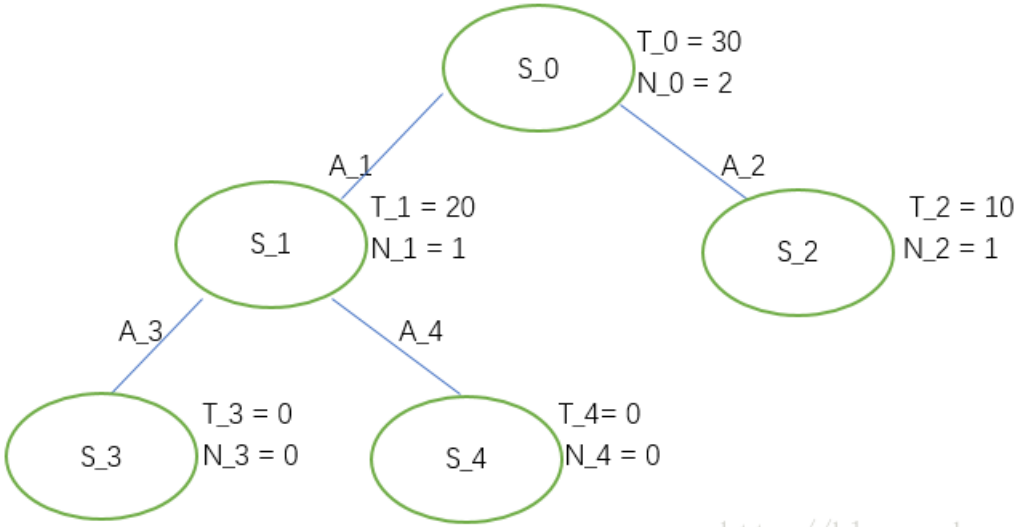
MCTS的想法就是要从S0出发不断的进行迭代，不断更新结点值，直到达到一定的迭代次数或者时间

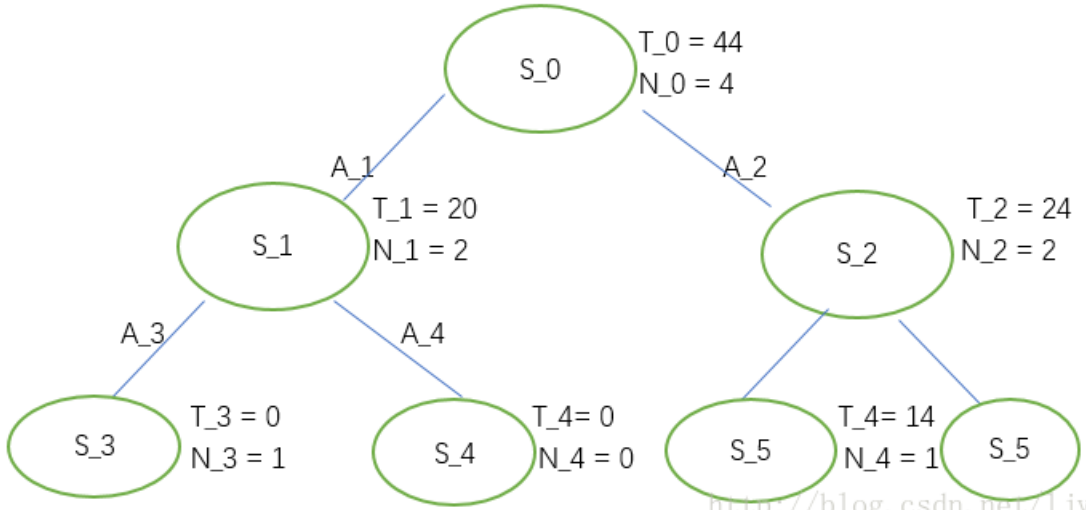
MCTS举例





MCTS举例

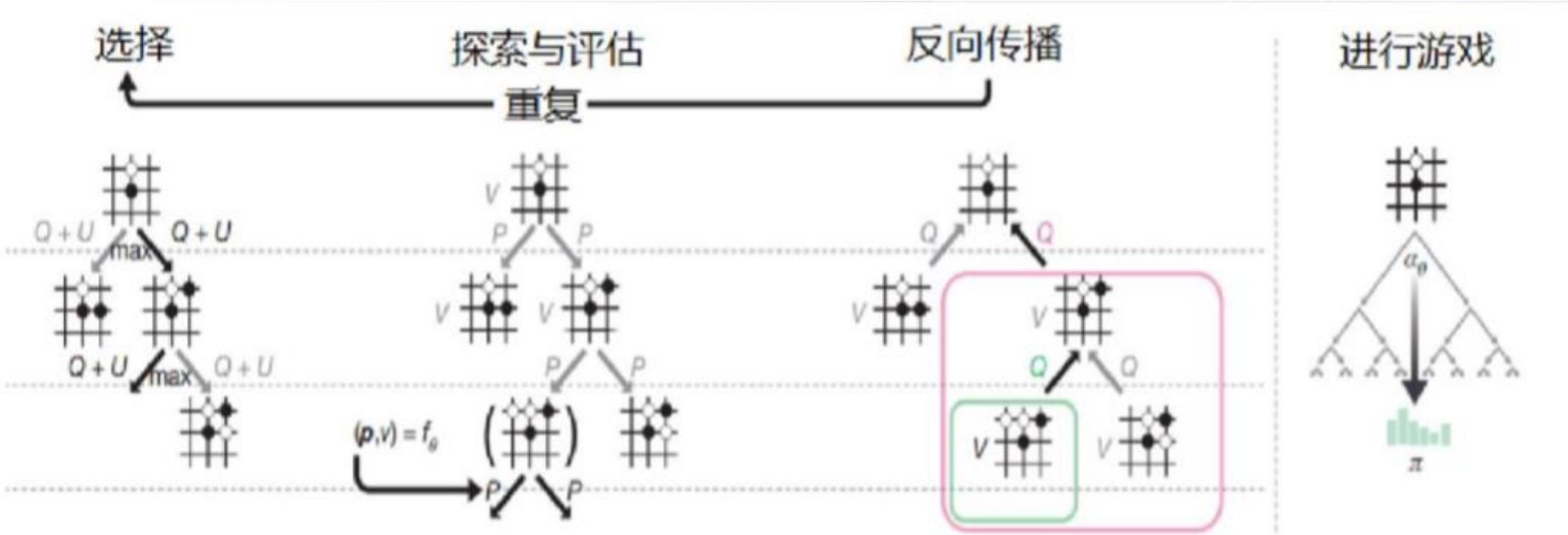




MCTS关键问题

- 什么是展开或未完全展开的博弈树?
- 在搜索过程中，向下遍历是什么意思？如何选择访问的下一个子节点?
- 什么是模拟?
- 什么是反向传播?
- 反向传播回的统计数据是什么，在展开博弈树结点更新的是什么?
- 最后的行动策略到底是如何选择的?

MCTS举例



五子棋自动对弈de搜索过程

强化学习做决策
蒙特卡洛做搜索

人工智能 = 算法 + 算力 + 数据

算力

Microsoft

背景 – 算力的重要性

- 1 块 K80 GPU 5 小时 53 分跑完 200 个 Epoch
- 本地 CPU 机器跑同样的 Training：耗时 8 天跑完 48 个 Epoch

"性价比上，同样的钱买GPU和买CPU，在做神经网络的时候，速度上大约有14.5~16.8倍的差距"

- AlphaGo在围棋中击败李世石(Lee Sedol)，需要使用1,920个CPU和280个GPU以训练模型，训练过程用电成本约为3,000美元。

4 本讲小结

1）博弈特征

“双人有限**零和**顺序游戏”，状态空间搜索，动态，预估对方

2）极大极小算法：宽度优先搜索

- MAX方、MIN方；MINMAX的基本思想
- 过程：宽度优先法生成规定深度的全部博弈树，然后对其所有端节点计算其静态估计函数值;从底向上逐级求非终结点的倒推估计值，至初始节点的倒推值f(s)为止，标记走步。

3）α-β剪枝算法：有限深度优先搜索

- 极大层的这个下界值为α、极小层的这个上界值为β
- α剪枝：α（先辈层）≥β（后继层），则可中止该极小值层中这个节点以下的搜索
- β剪枝： α（后继层）≥β（先辈层），则可中止该极大值层中这个节点以下的搜索

4）蒙特卡洛树算法

ALPHAGO、五子棋类似，只是规模不同；强化学习做决策、蒙特卡洛做搜索

5）算力