



人工智能导论

腾讯云人工智能特色班课程

主讲人：高 灿，致腾楼936

(davidgao@szu.edu.cn)

时 间：周三晚上11-12节 致理楼L1-306（理论）

周三晚上13-14节(单) 致腾楼318（实验）



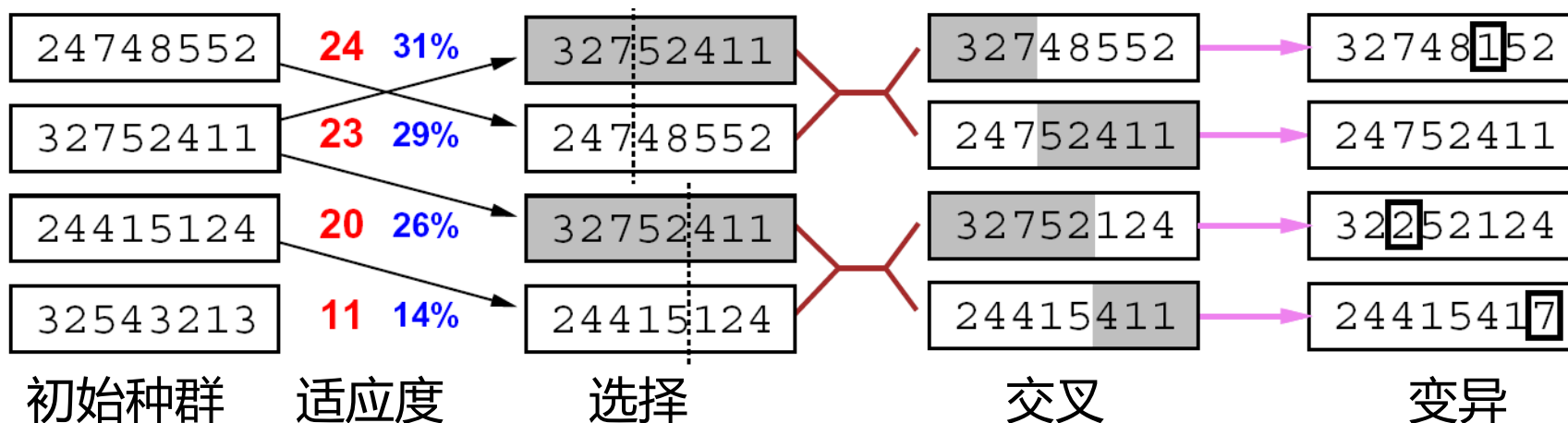
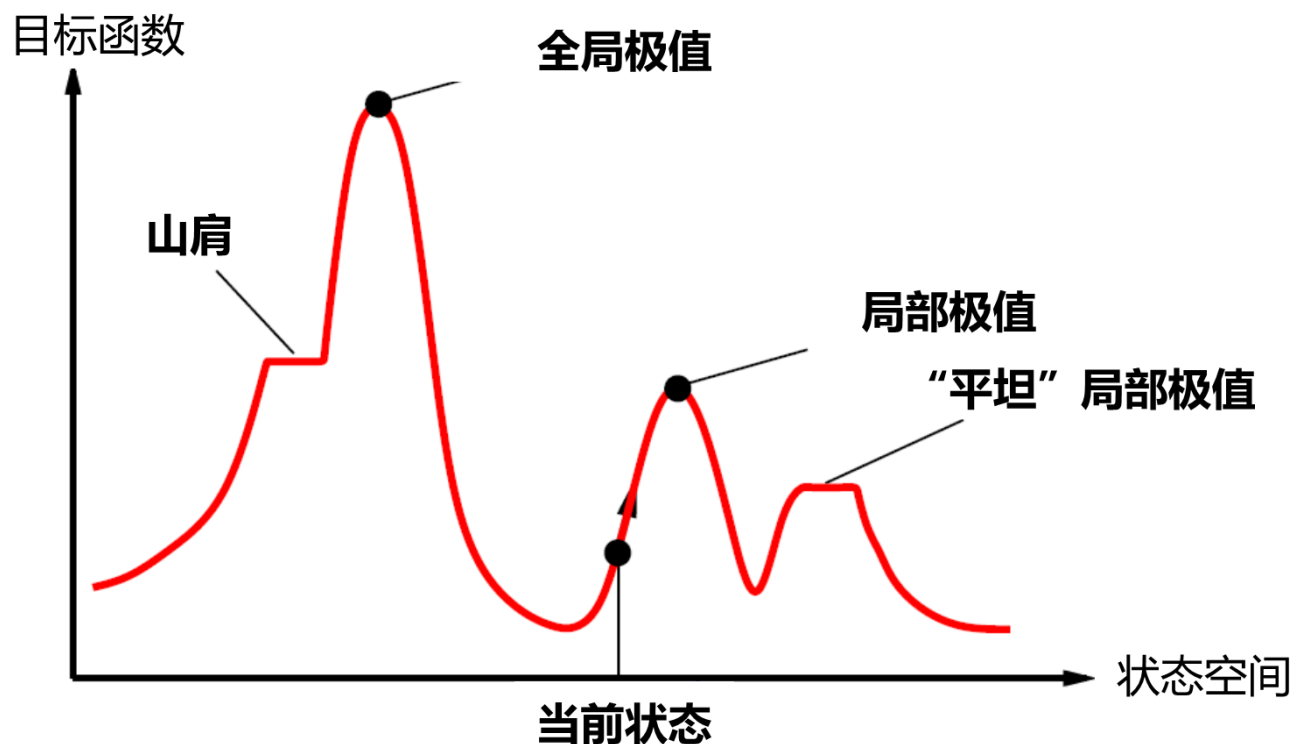
深圳大学 计算机与软件学院

College of Computer Science and Software Engineering of Shenzhen University

知识回顾

局部搜索

- 爬山法
- 模拟退火
- 局部束搜索
- 遗传算法



目录

1

博弈

2

Minimax搜索

3

Alpha-beta剪枝

4

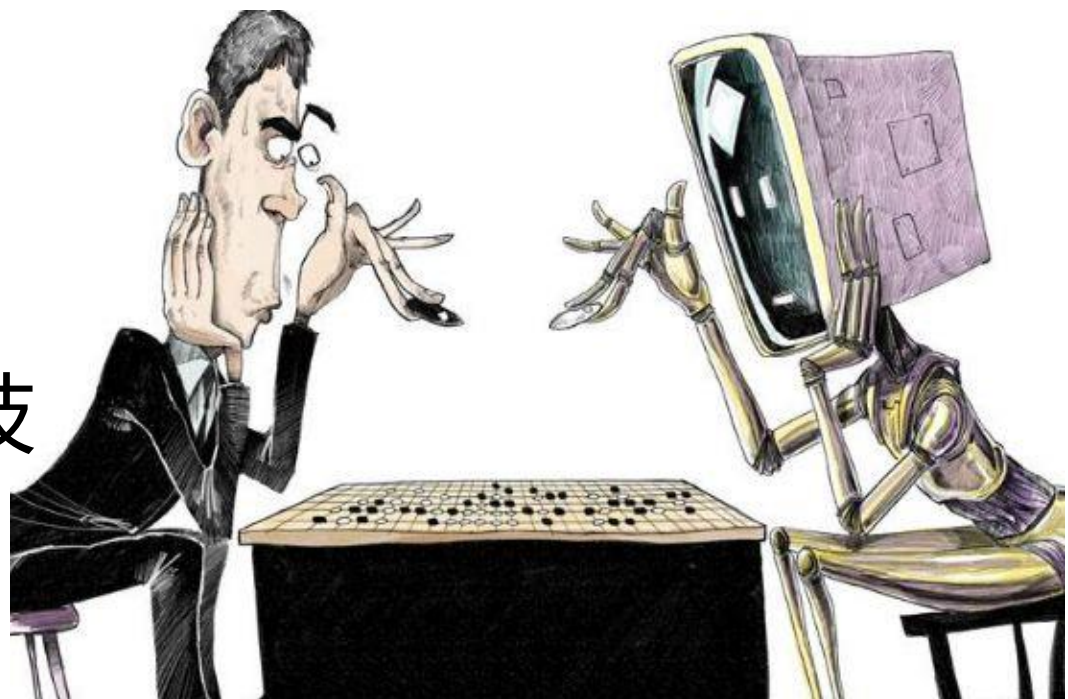
评估函数

5

博弈扩展-AlphaGo

6

习题及实验



1. 博弈

黑白棋(1988):

李开复开发的“奥赛罗”人机对弈系统击败世界锦标赛冠军Brian Rose

西洋跳棋(1994):

Schaeffer设计的Chinook跳棋程序第一次战胜了人类世界冠军

国际象棋(1997):

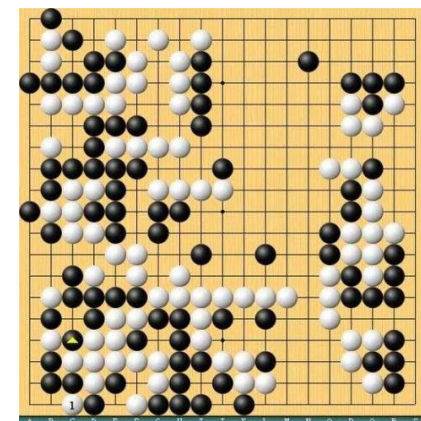
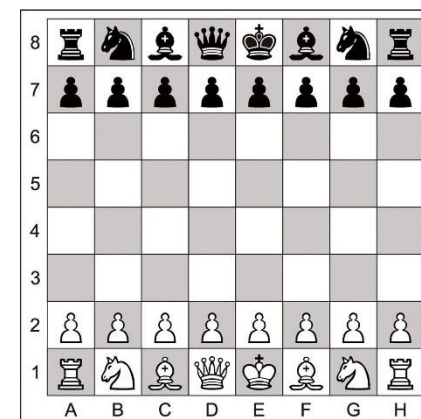
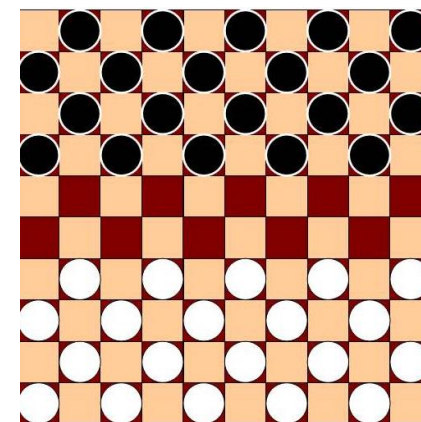
IBM深蓝打败国际象棋大师卡斯帕罗夫

中国象棋(2006):

计算机浪潮天梭终战胜中国象棋大师联盟

围棋(2016):

Alpha Go 战胜世界冠军李世石



1. 博弈

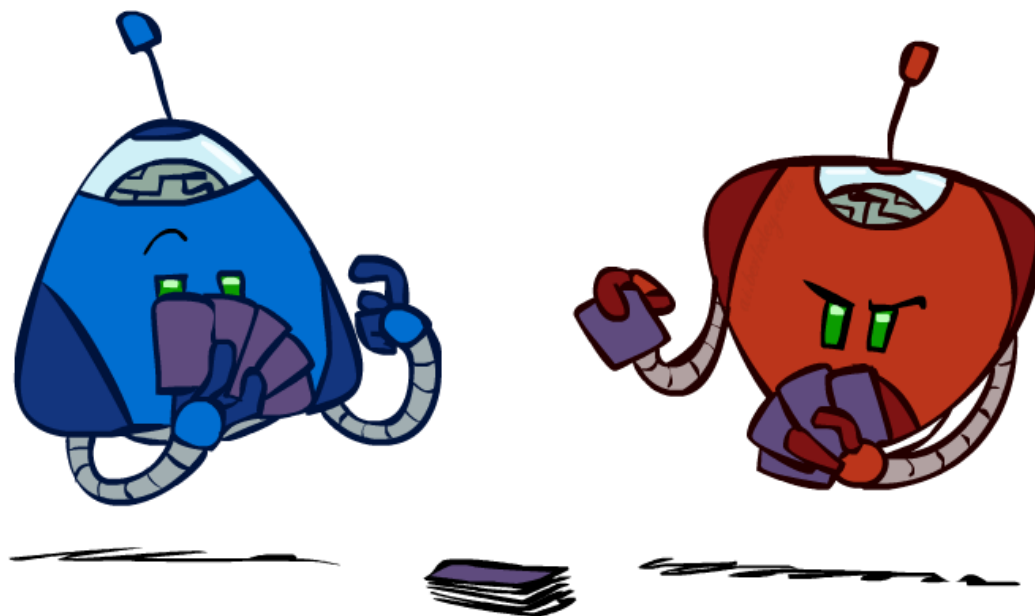
博弈(Game)的种类:

玩家数目?

确定的?

信息完整性?

零和(对抗性)?



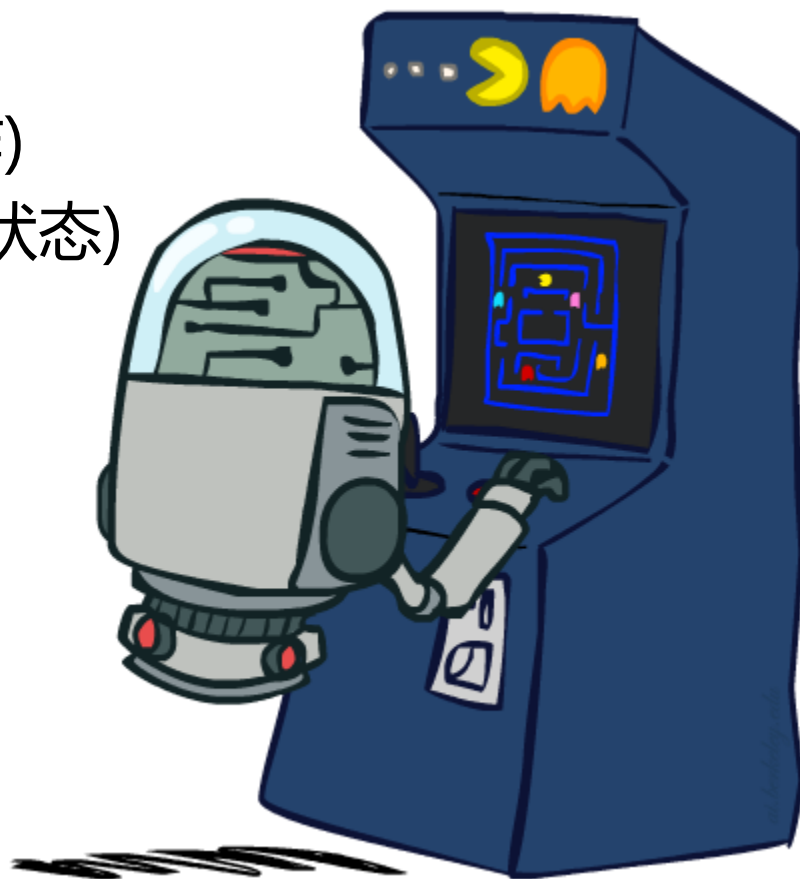
任务	观察度	智能体	确定性	信息
字谜游戏	全部	单个	确定	离散
象棋	全部	多个	确定	离散
扑克	部分	多个	随机	离散
医学诊断	部分	单个	随机	连续
图像分析	全部	单个	确定	连续
AI助手	部分	多个	随机	离散

1. 博弈

确定性博弈

- 状态空间: S (S_0 开始)
- 玩家: $P = \{1 \dots N\}$ (一般轮流动作)
- 行动: A (取决于参与者和当前的状态)
- 状态转换函数: $S \times A \rightarrow S$
- 效用函数: $S \times P \rightarrow R$
- 终止测试: $S \rightarrow \{t, f\}$

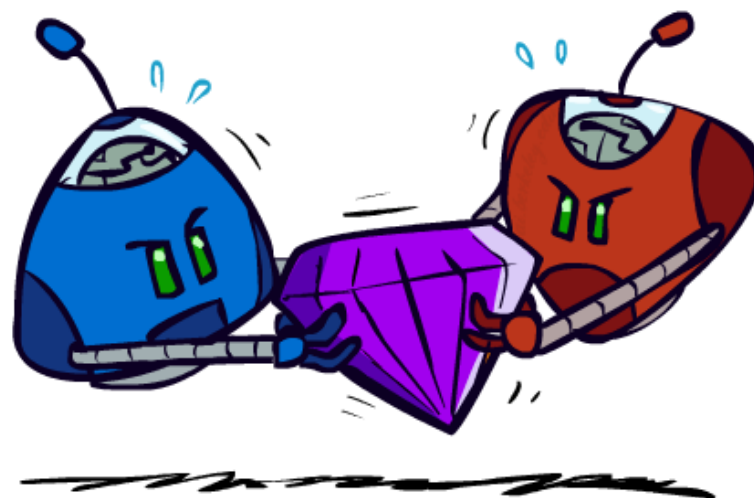
玩家的目的是找到策略: $S \rightarrow A$



1. 博弈

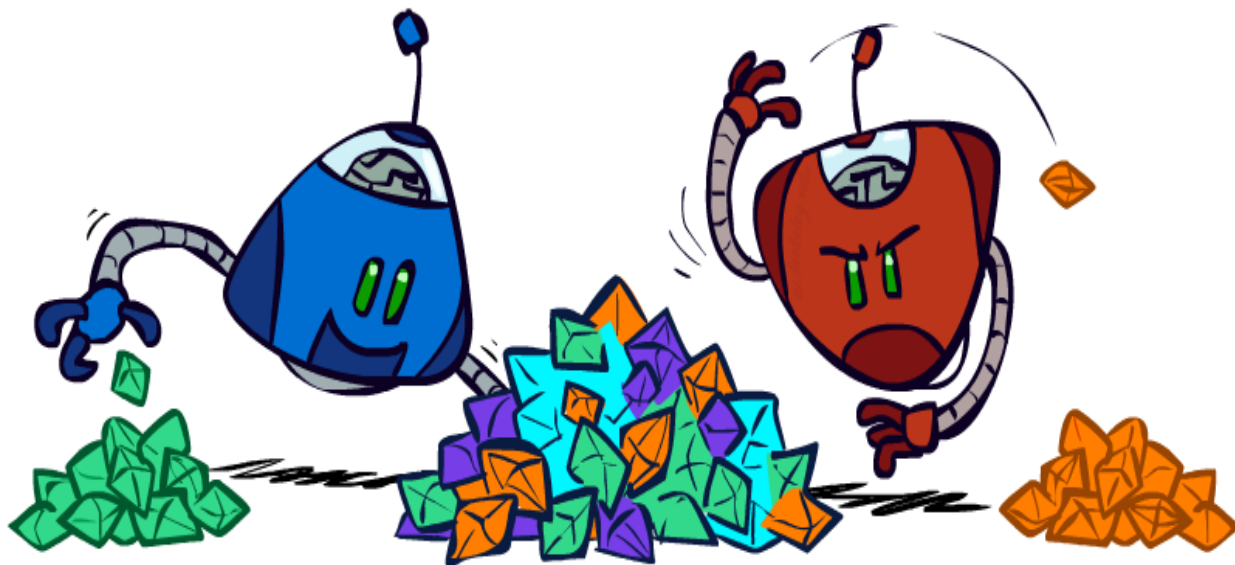
零和博弈

- 效益互斥
- 最大化最小化评估值
- 竞争对抗

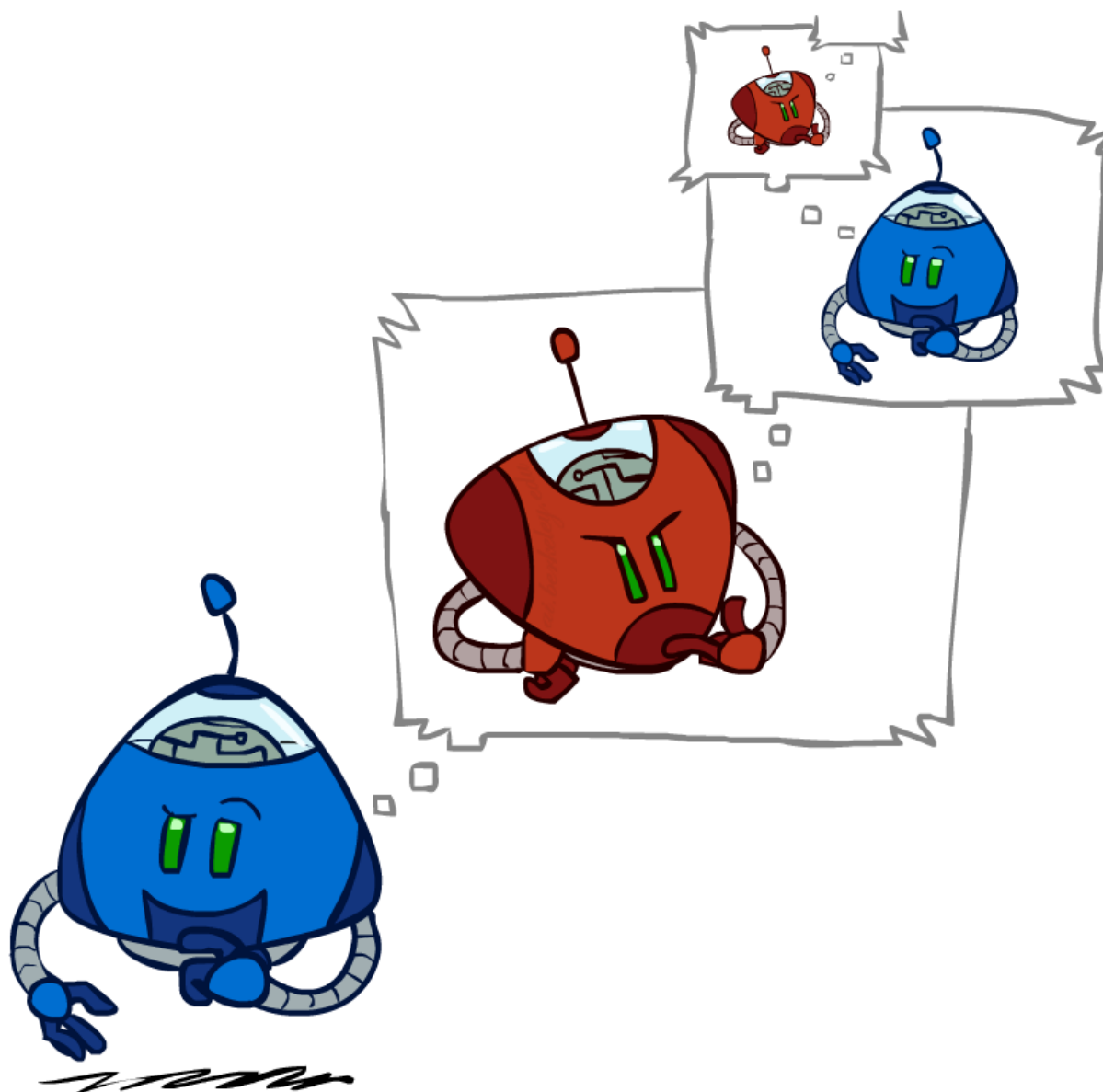


一般博弈

- 效益独立
- 个体最大化评估值
- 合作、中立或竞争



1. 博弈 对抗性



1. 博弈

对抗性博弈（限定）：在一定规则条件下，有完备信息的，确定性的，轮流行动的，两个游戏者的零和游戏 -- 《人工智能-一种现代方法》

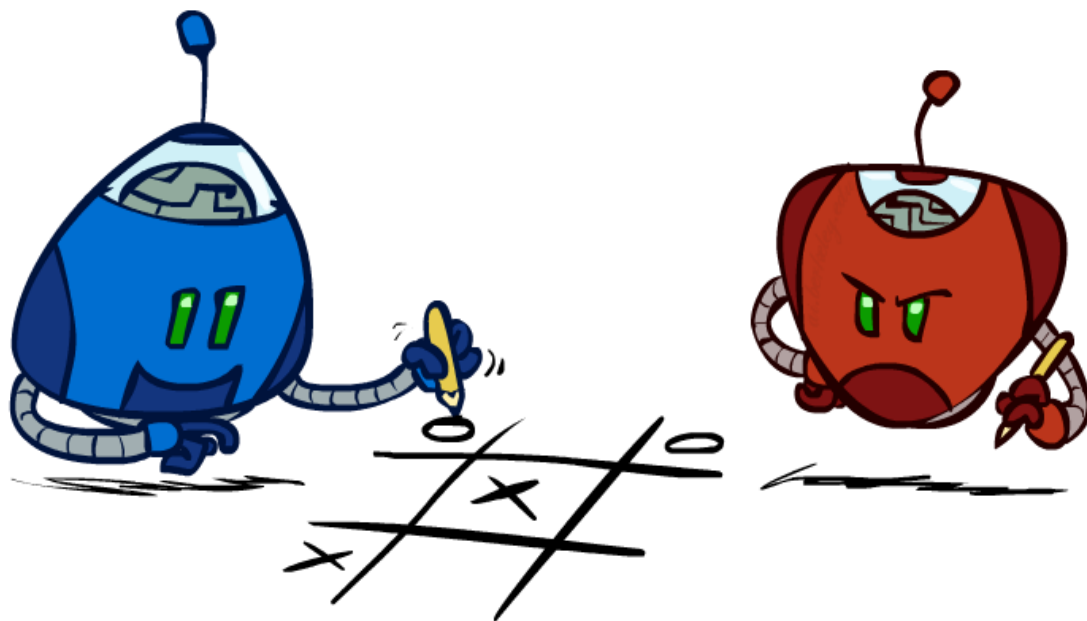
对抗性博弈特点：

信息完整

确定的

双方轮流

零和



百度百科（可参看一般定义）：

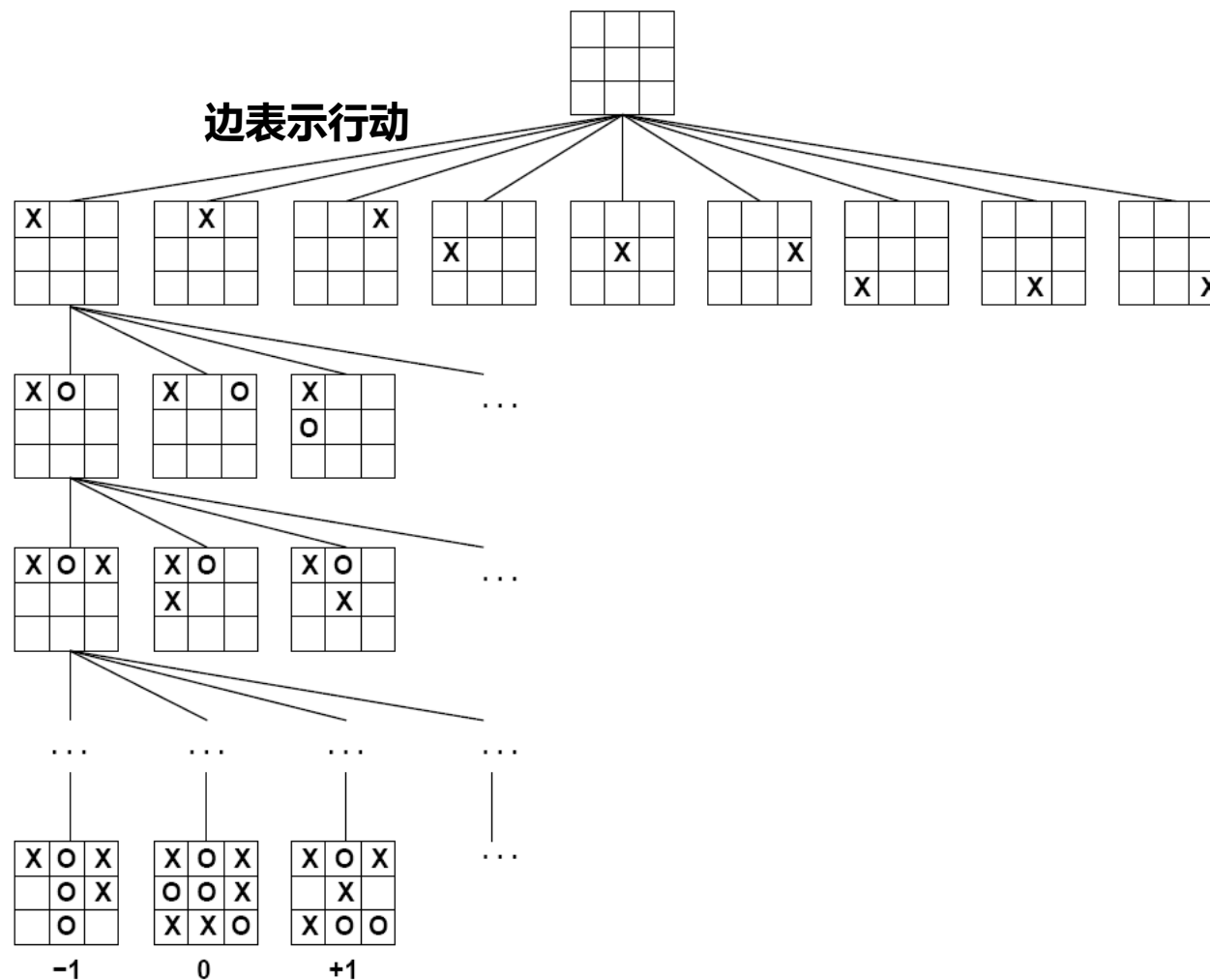
<https://baike.baidu.com/item/%E5%8D%9A%E5%BC%88/4669968?fr=aladdin>

1. 博弈

博弈树



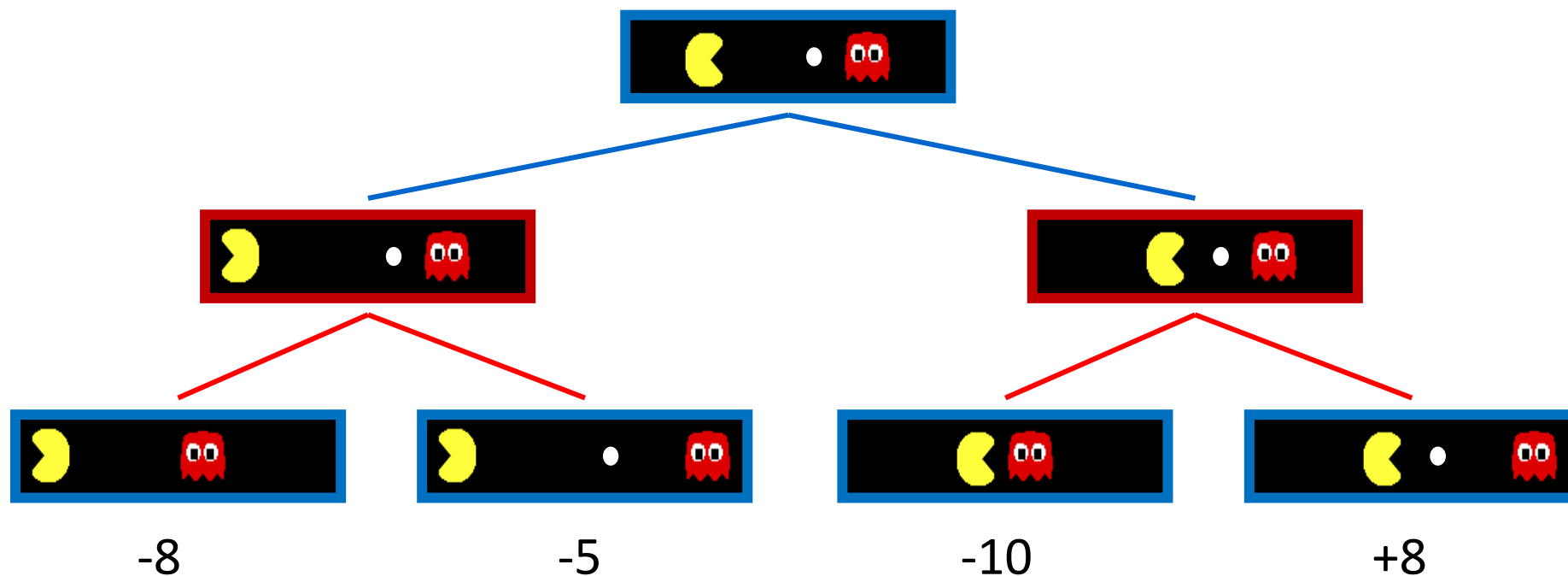
井字棋游戏的博弈树（部分）



1. 博弈

博弈树

吃豆人博弈树（部分）



2. 对抗搜索

Minimax算法:

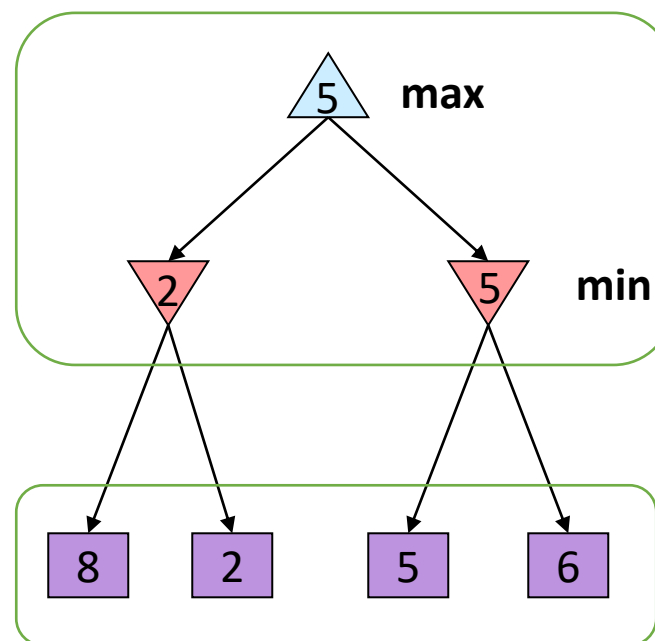
在双人游戏中找到最优步骤的方法

- Max: 最大化效益(AI, ▲□)
- Min: 最小化效益(Player, ▼○)

算法特点

- 状态空间是树
- 玩家交替轮换
- 计算每个结点的极小极大值:
 - 对手可实现的最佳效益

最小最大值:迭代计算



末端效益

2. 对抗搜索

Minmax实现 (搜索)

def max-value(state):

 initialize $v = -\infty$

 for each successor of state:

$v = \max(v, \text{min-value}(\text{successor}))$

 return v

def min-value(state):

 initialize $v = +\infty$

 for each successor of state:

$v = \min(v, \text{max-value}(\text{successor}))$

 return v

递归定义

什么时候返回值?

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

2. 对抗搜索

Minmax实现 (计算)

def value(state):

if the state is a terminal state: return the state' s utility

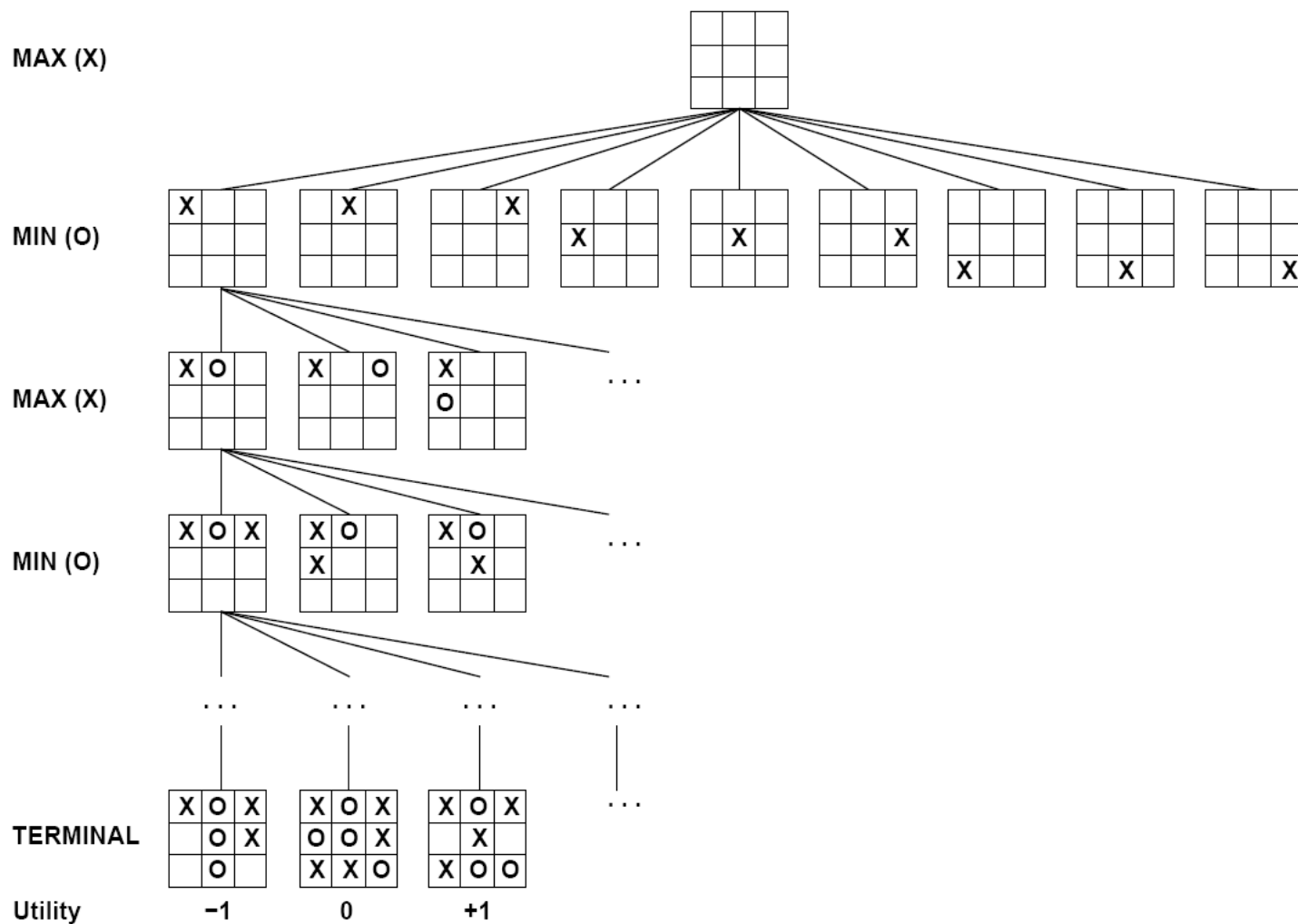
if the next agent is MAX: return max-value(state)

if the next agent is MIN: return min-value(state)

什么方式搜索?

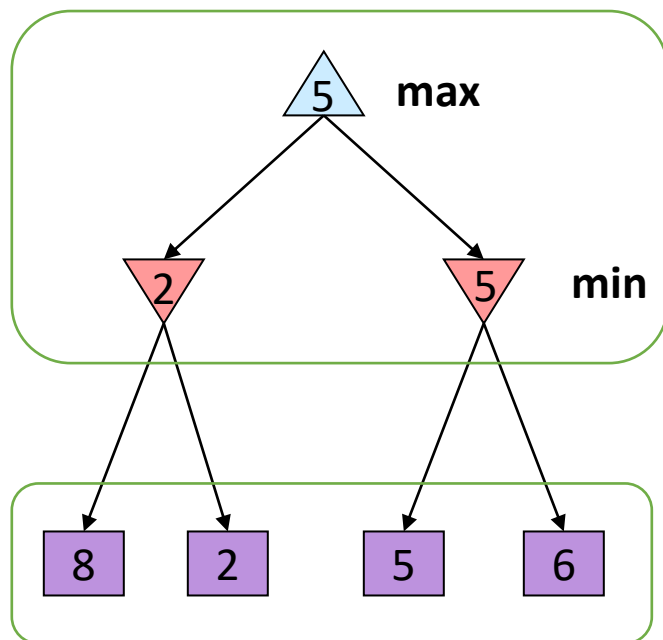
如何计算数值?

- 博弈
- Minimax
- Alpha-b



知识回顾

- 博弈
- Minimax搜索
- Alpha-beta剪枝



def max-value(state):

 initialize $v = -\infty$

 for each successor of state:

$v = \max(v, \text{min-value}(\text{successor}))$

 return v

def min-value(state):

 initialize $v = +\infty$

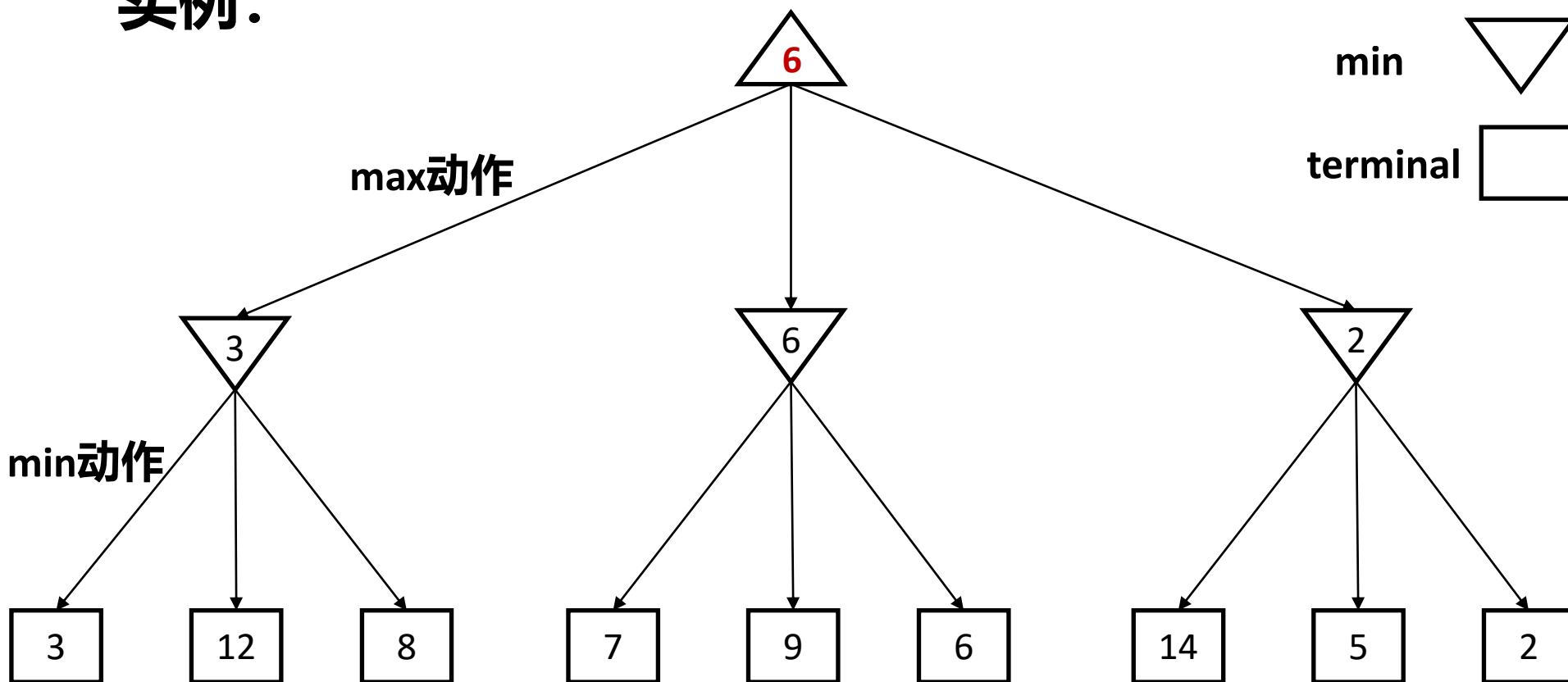
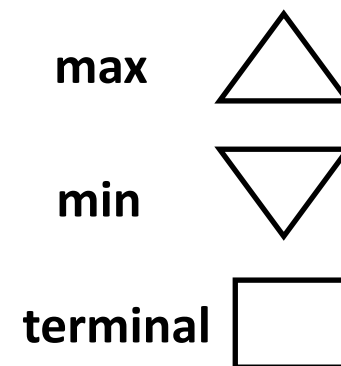
 for each successor of state:

$v = \min(v, \text{max-value}(\text{successor}))$

 return v

2. 对抗搜索

实例：



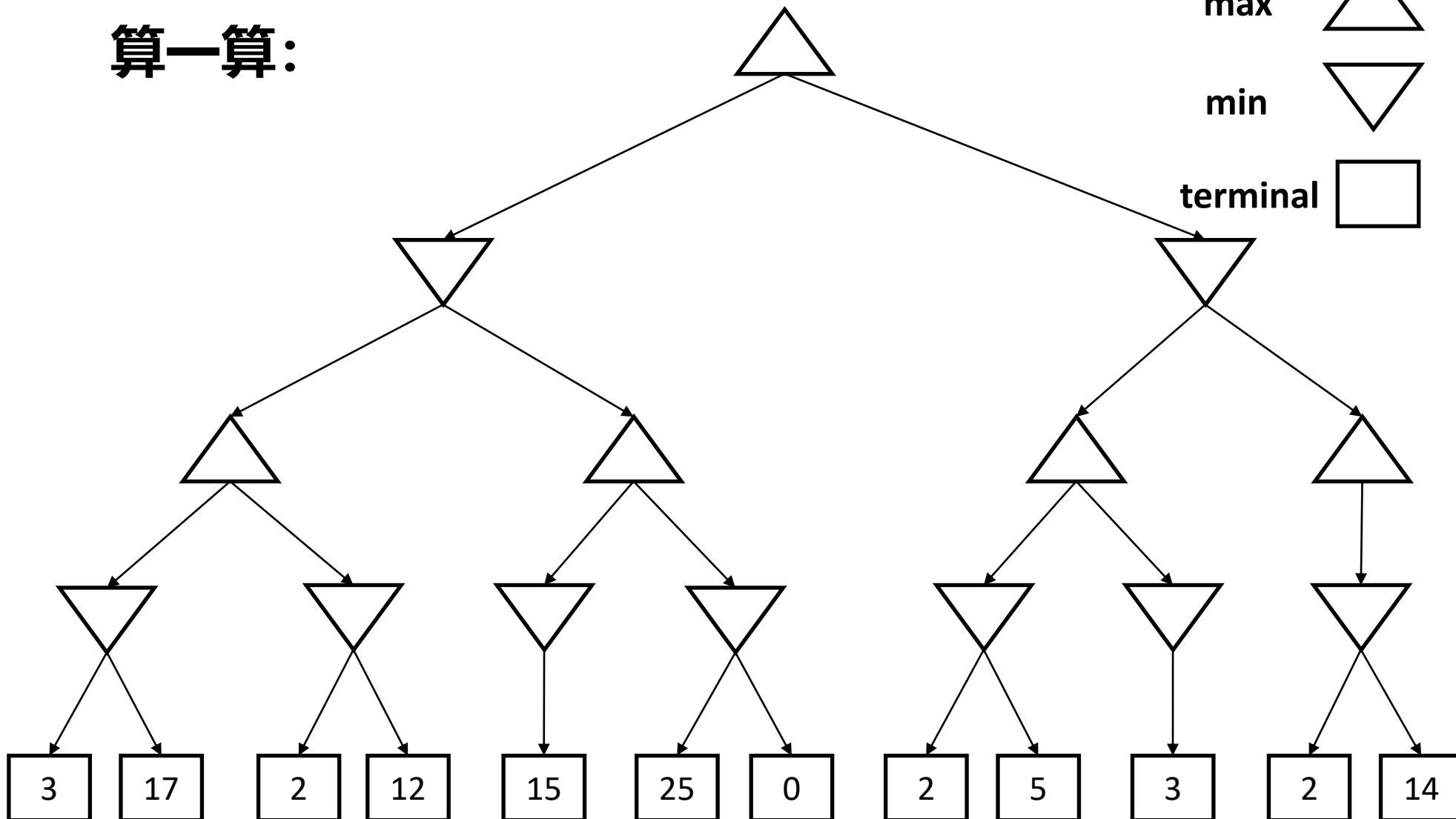
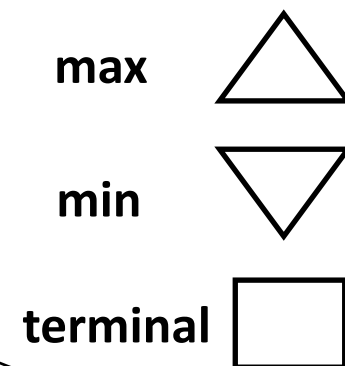
状态评价

搜索：自顶向下

计算：自底向上


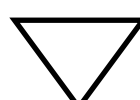

2. 对抗搜索(Minimax)

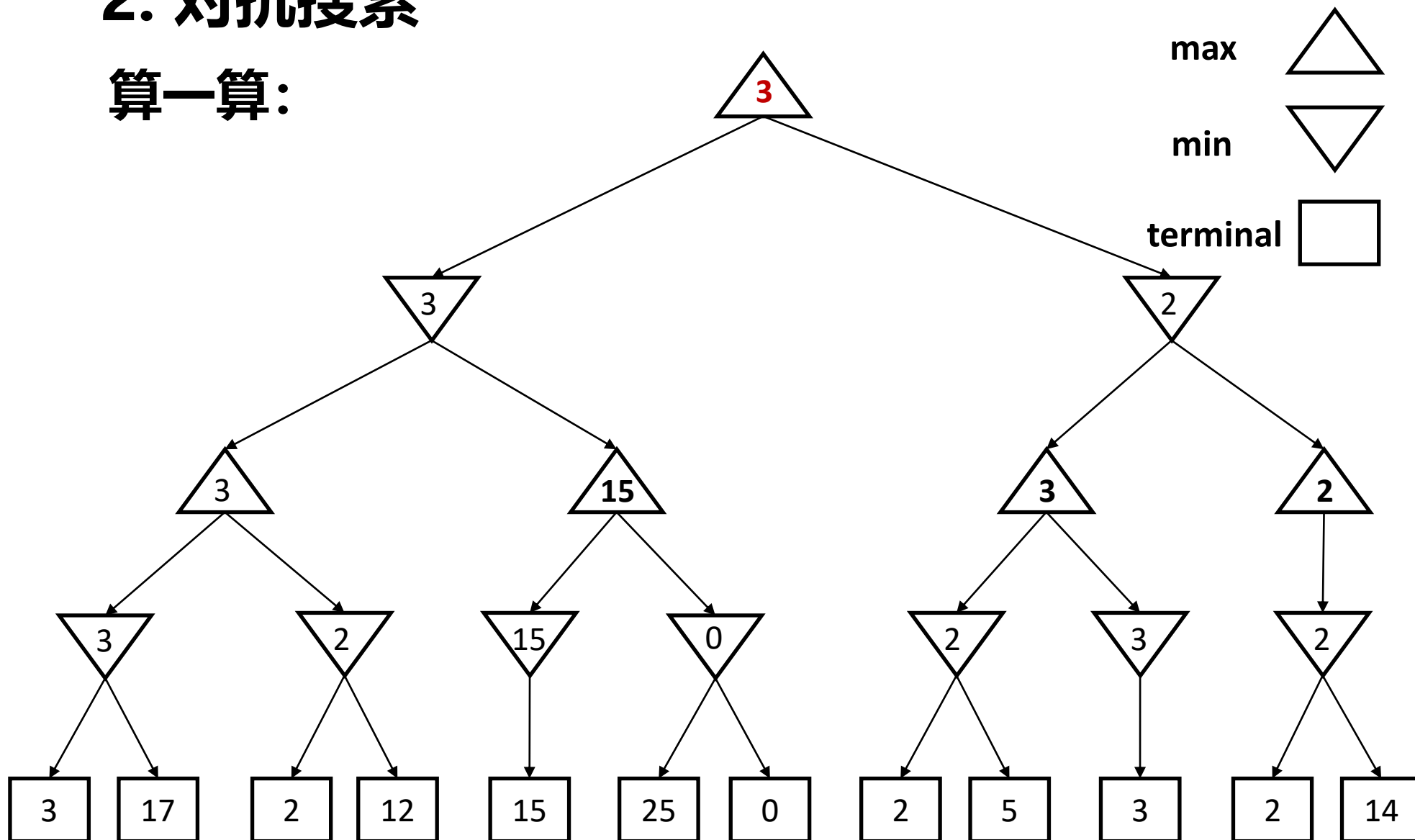
算一算：



2. 对抗搜索

算一算：

max 
min 
terminal 



2. 对抗搜索

算法分析：

搜索方法：类似DFS

时间复杂度： $O(b^m)$

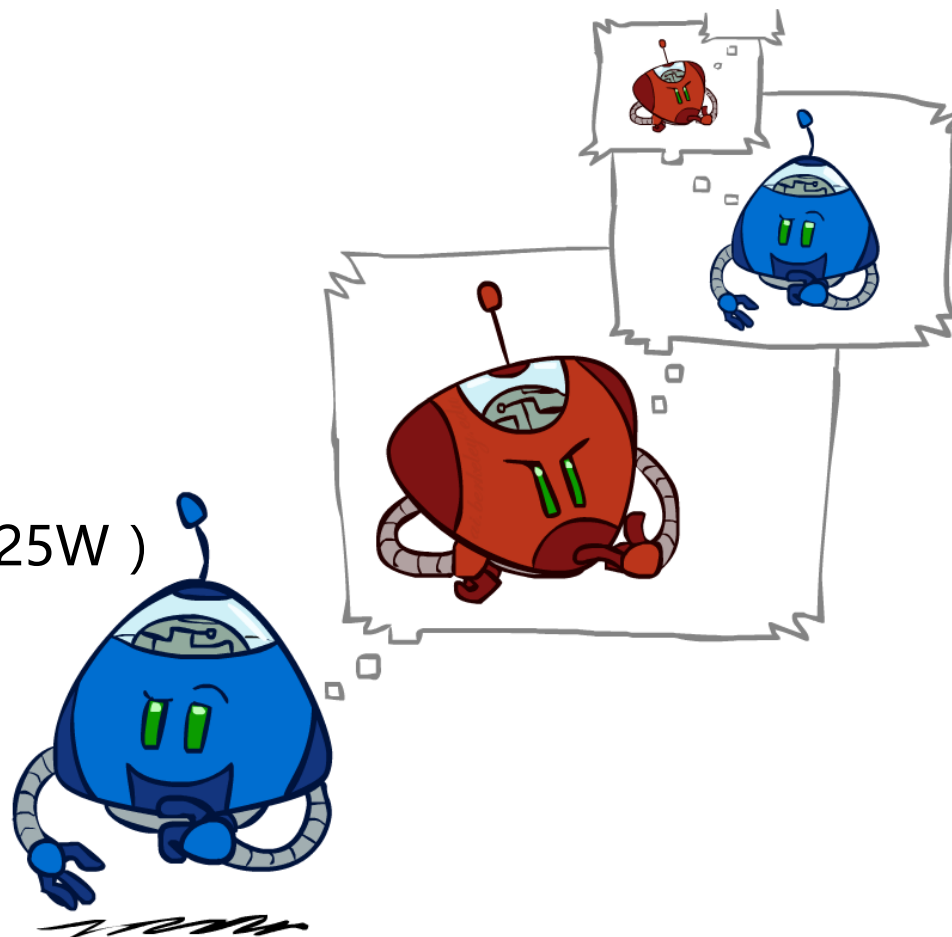
空间复杂度： $O(bm)$

面临的问题：

搜索空间巨大（4步搜索空间 $50^4 = 625W$ ）

获得精确的解完全不可能

**是否真的需要搜索
完整的博弈树？**



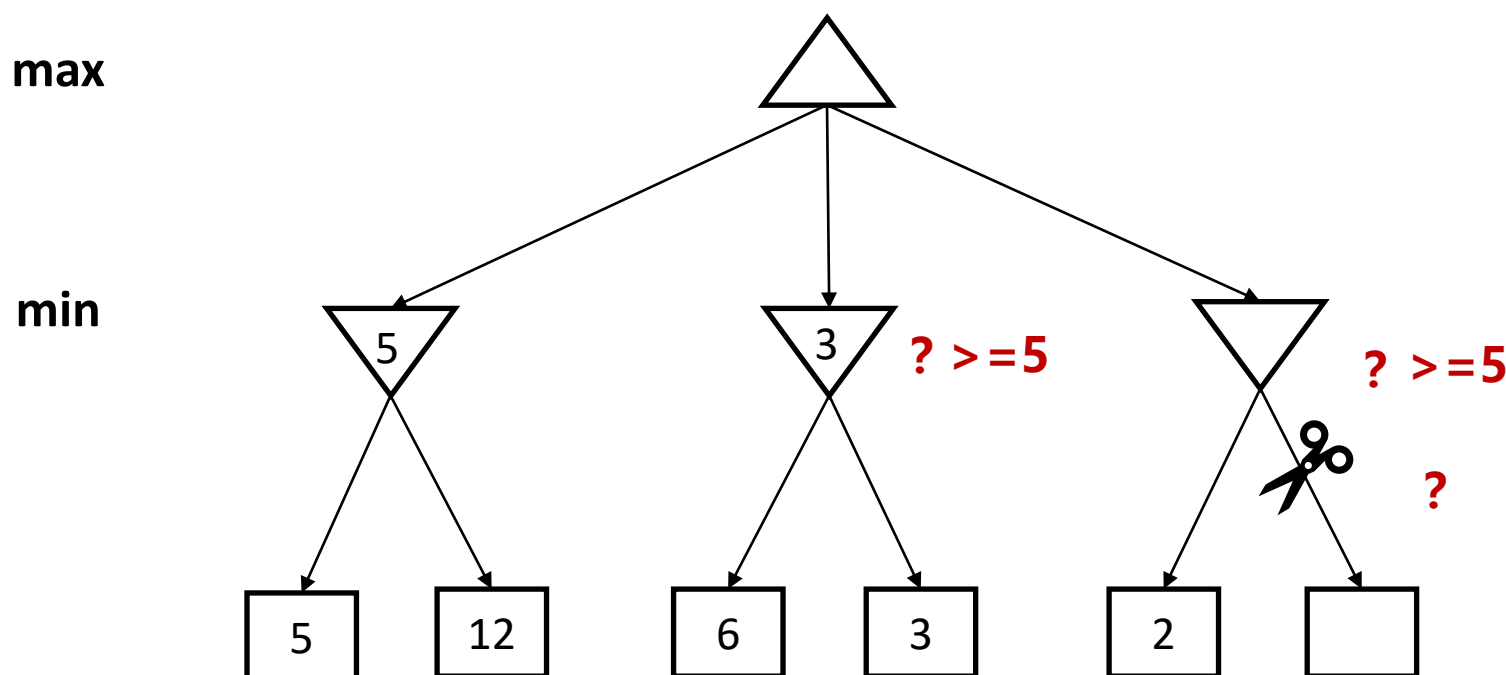
3. Alpha-beta剪枝

- 极小极大算法是一种在两人博弈中寻找最优步骤的方法。
- Alpha-beta剪枝是一种找到最佳minimax步骤的方法，同时可以避免搜索**不可能被选择**的步骤的子树。
- Alpha-beta剪枝的名称源于搜索过程中传递的两个边界参数，这些边界基于已经看到的搜索树部分来限制候选步骤。
- **Alpha是可能步骤的最大下界**
- **Beta是可能步骤的最小上界**
- **任何新节点被认为是步骤的可能路径结点当且仅当**

$$\alpha \leq Value(N) \leq \beta$$

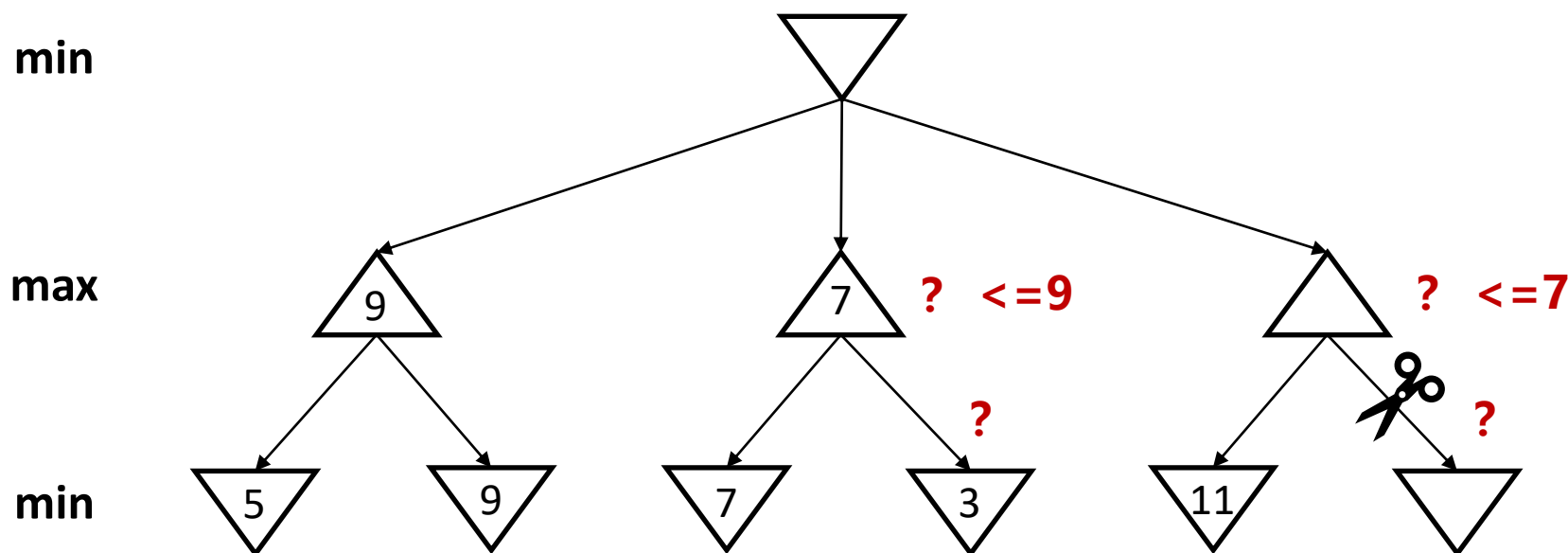
3. Alpha-beta剪枝

极小层:



3. Alpha-beta剪枝

极大层:



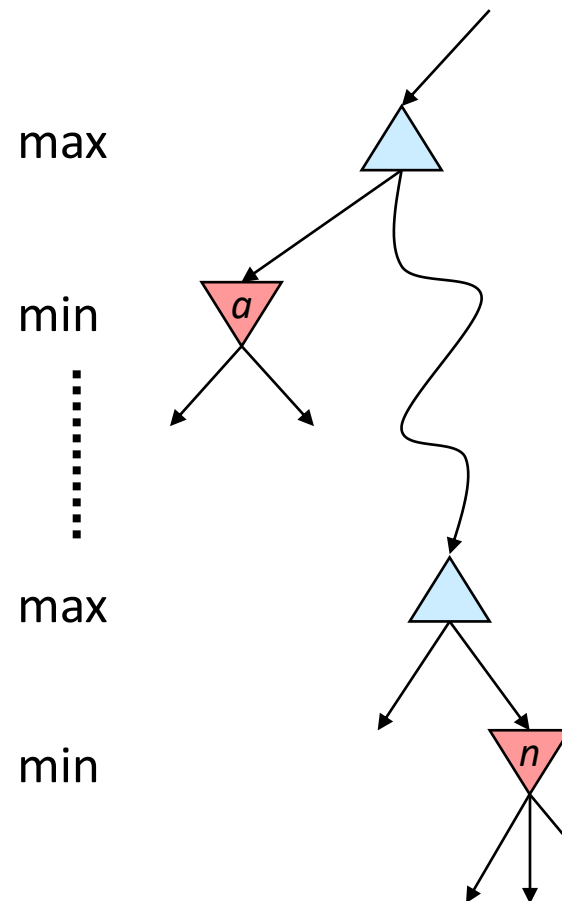
3. Alpha-beta剪枝

剪枝情况（最小值）：

- 计算结点n的最小值
- 考虑结点n的子结点
- 结点n对上一层有影响（max）
- 如果a是max层当前最大值结点
- 如果n比a差，max层可避免选择n
- n的子结点都不需考虑

**max层只考虑较当前
min层更好的结点**

**min层只考虑较当前
max层更差的结点**



3. Alpha-beta剪枝

Alpha-beta实现

def max-value(state, α , β):

 initialize $v = -\infty$

 for each successor of state:

$v = \max(v, \text{value}(\text{successor}, \alpha, \beta))$

 if $v \geq \beta$ return v

$\alpha = \max(\alpha, v)$

 return v

α : 从路径到根max最好选择

β : 从路径到根min最好选择

def min-value(state, α , β):

 initialize $v = +\infty$

 for each successor of state:

$v = \min(v, \text{value}(\text{successor}, \alpha, \beta))$

 if $v \leq \alpha$ return v

$\beta = \min(\beta, v)$

 return v

3. Alpha-beta剪枝

参数(α , β)迭代:

- 极大层节点参数设置 ($-\infty$, $+\infty$)
- 极小层节点参数设置 ($-\infty$, $+\infty$)
- 末端结点迭代更新极小极大层

初始化结点:



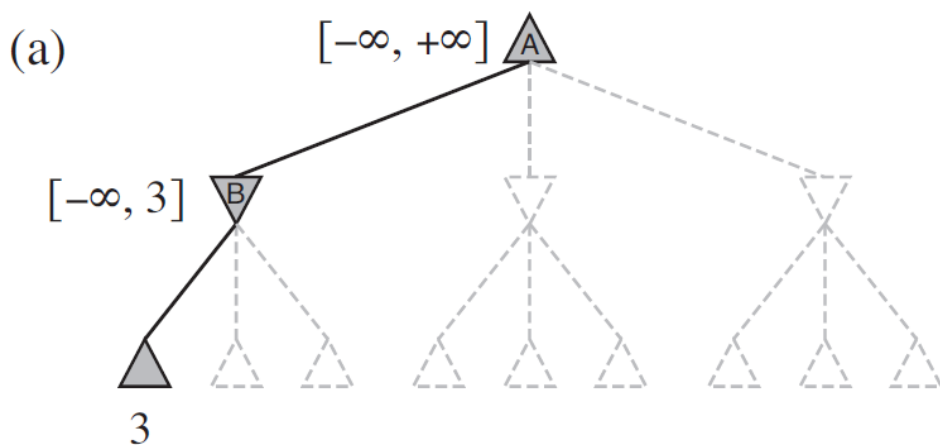
候选结点:



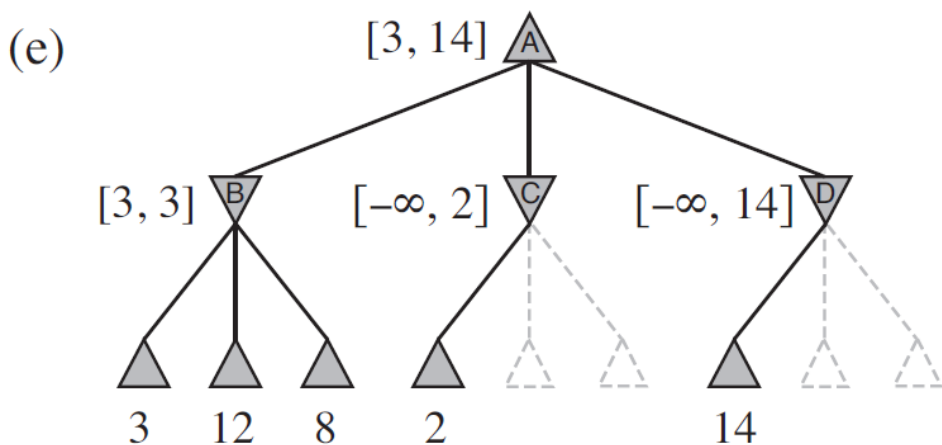
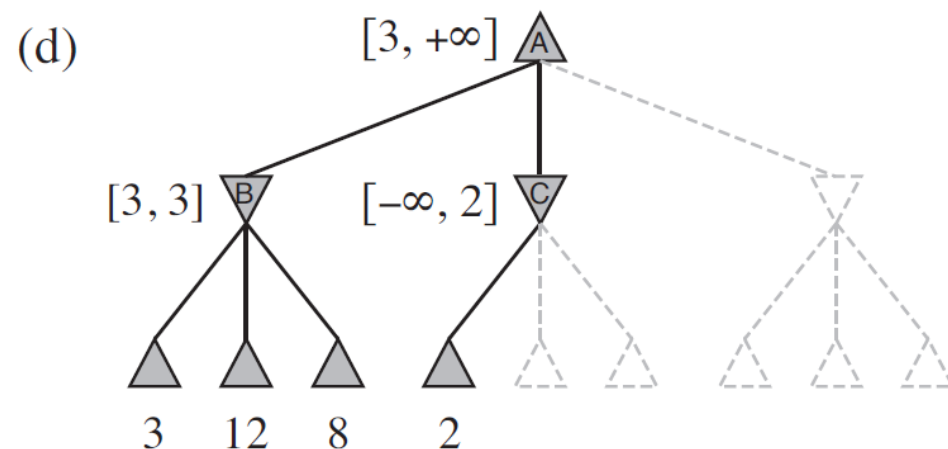
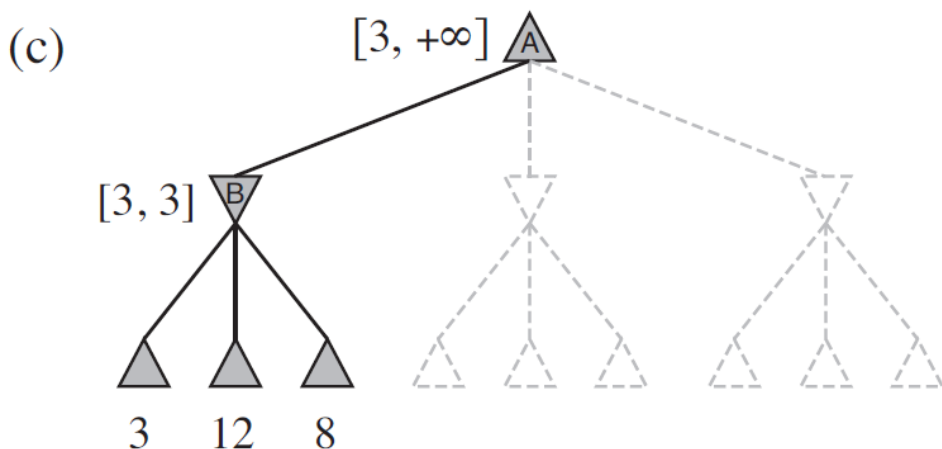
剪枝结点:



3. Alpha-beta剪枝

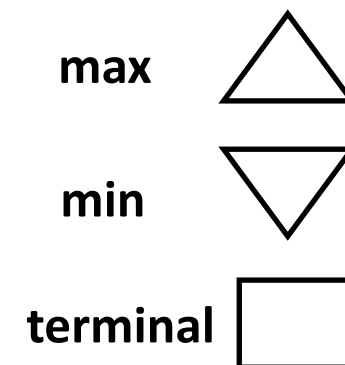
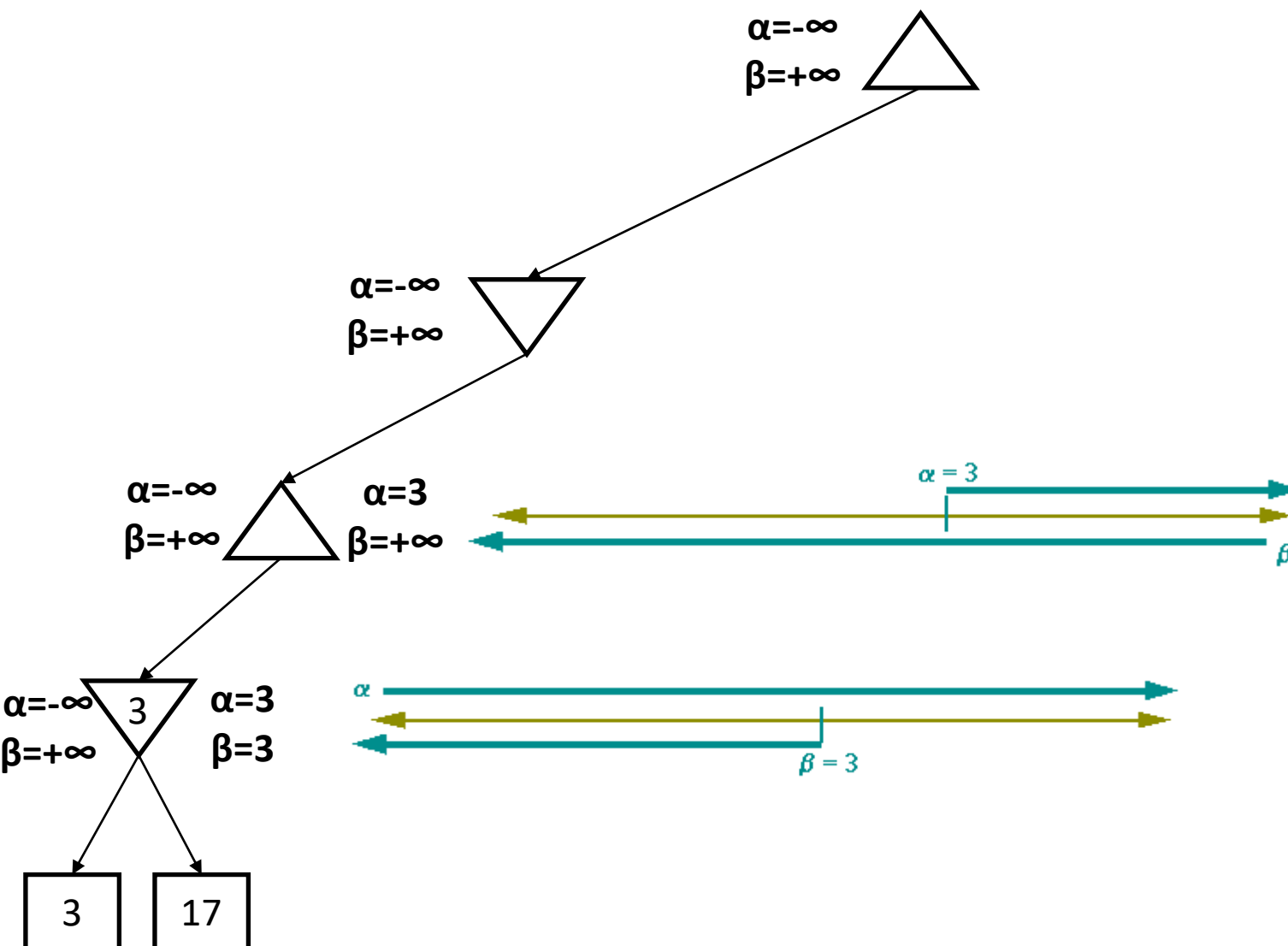


3. Alpha-beta剪枝

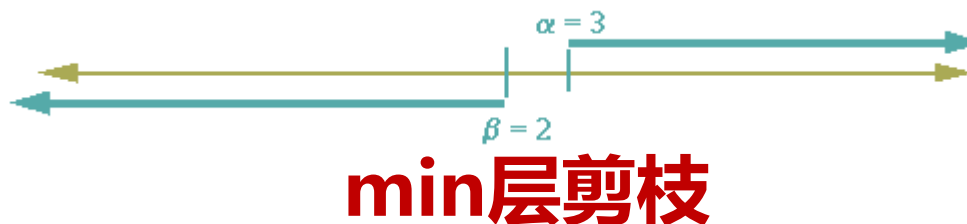
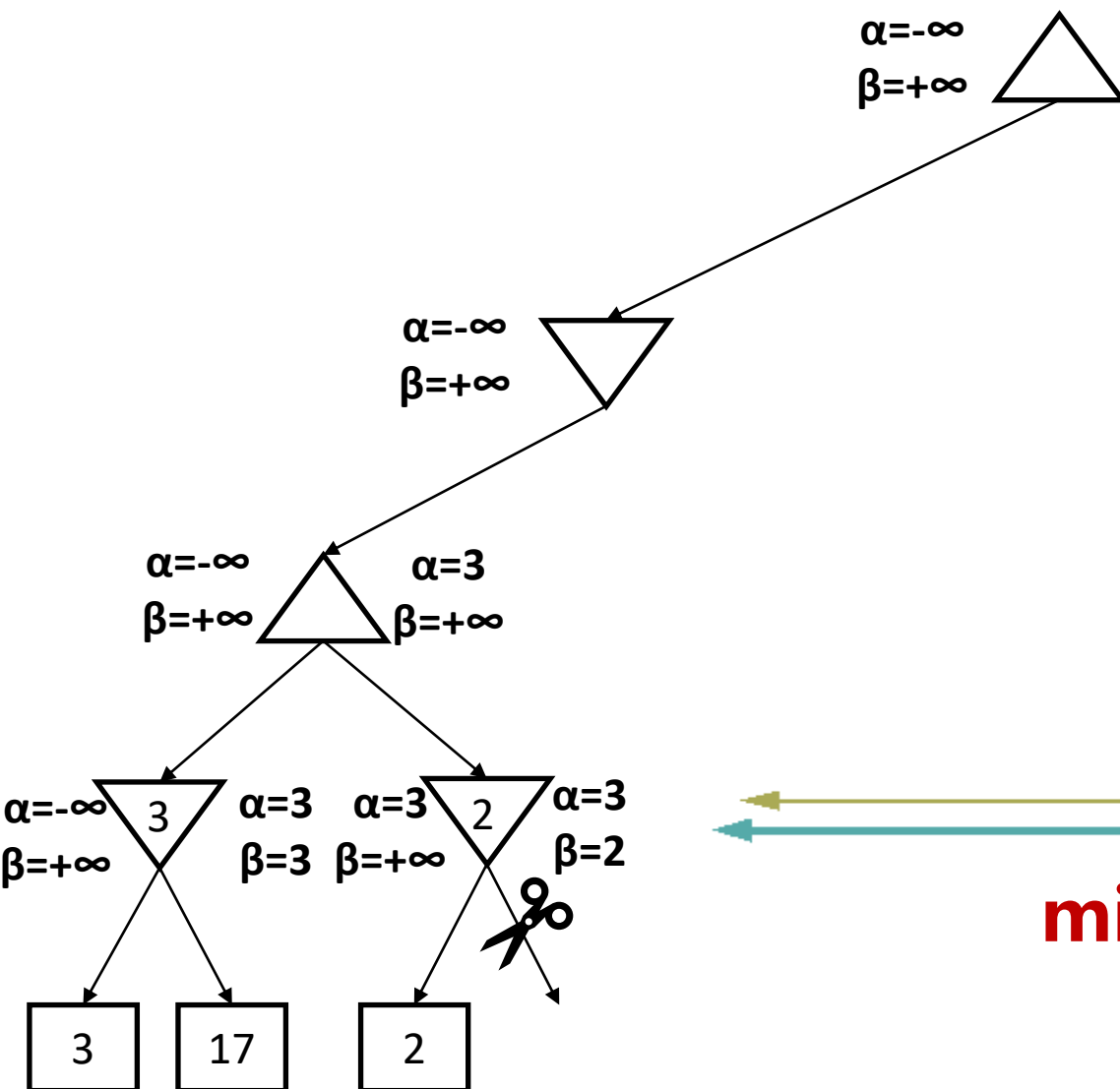
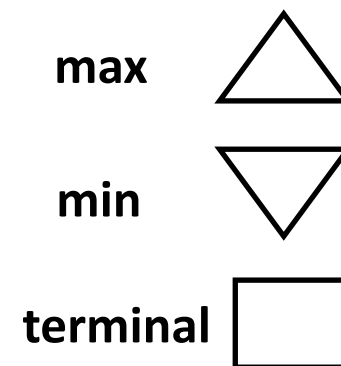


$$\begin{aligned} \text{MinMax}(\text{root}) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) = \max(3, z, 2) = 3 \end{aligned}$$

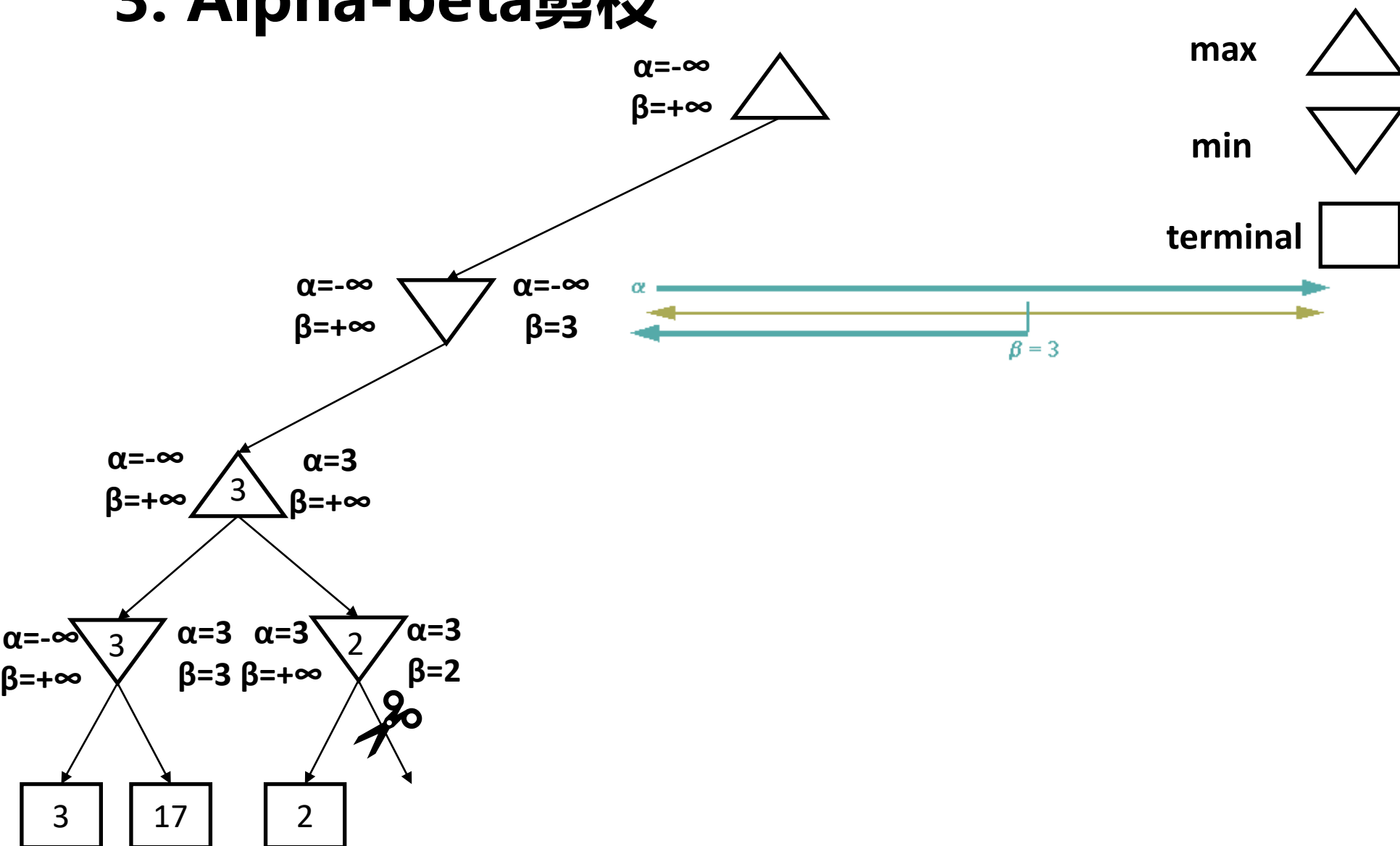
3. Alpha-beta剪枝



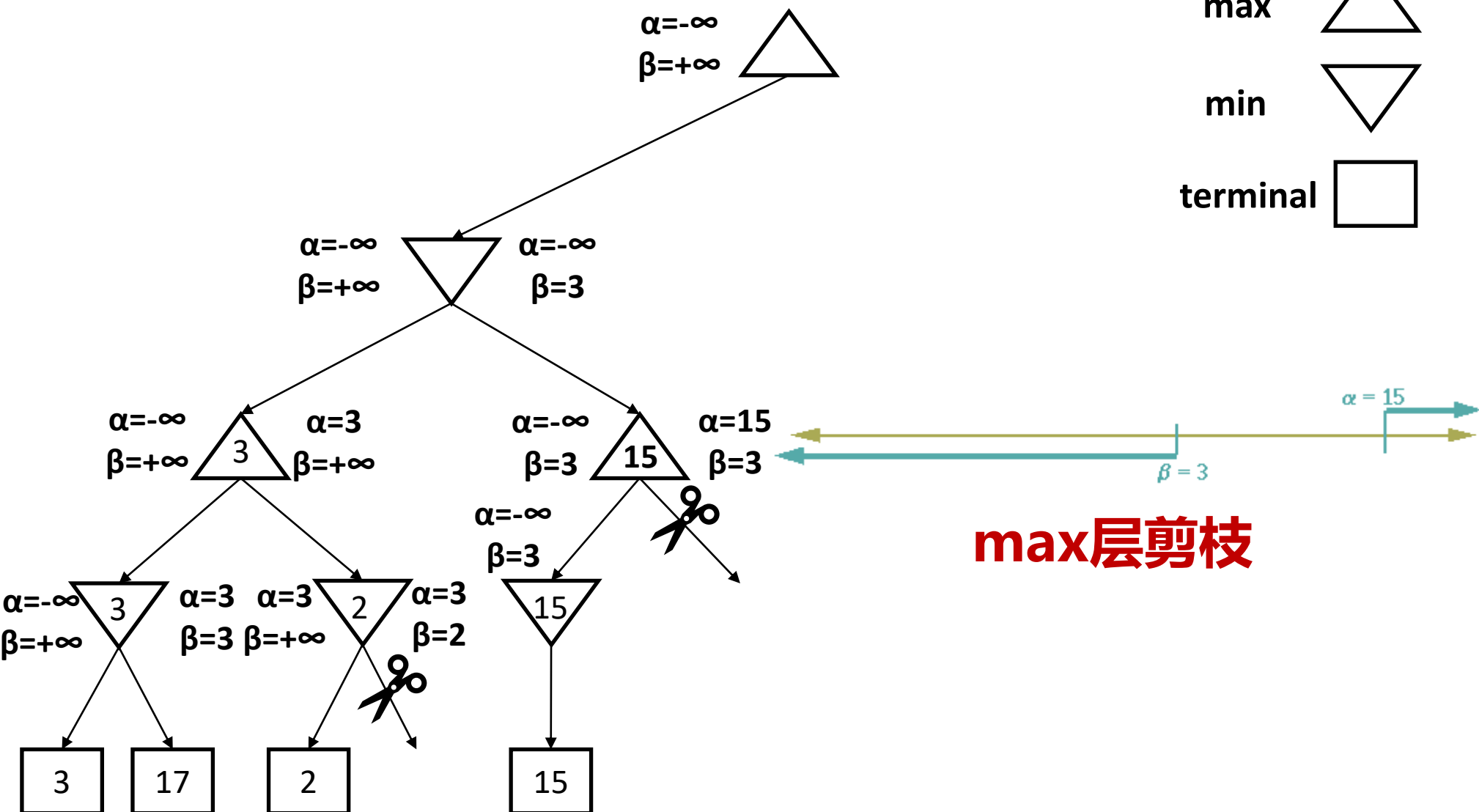
3. Alpha-beta剪枝



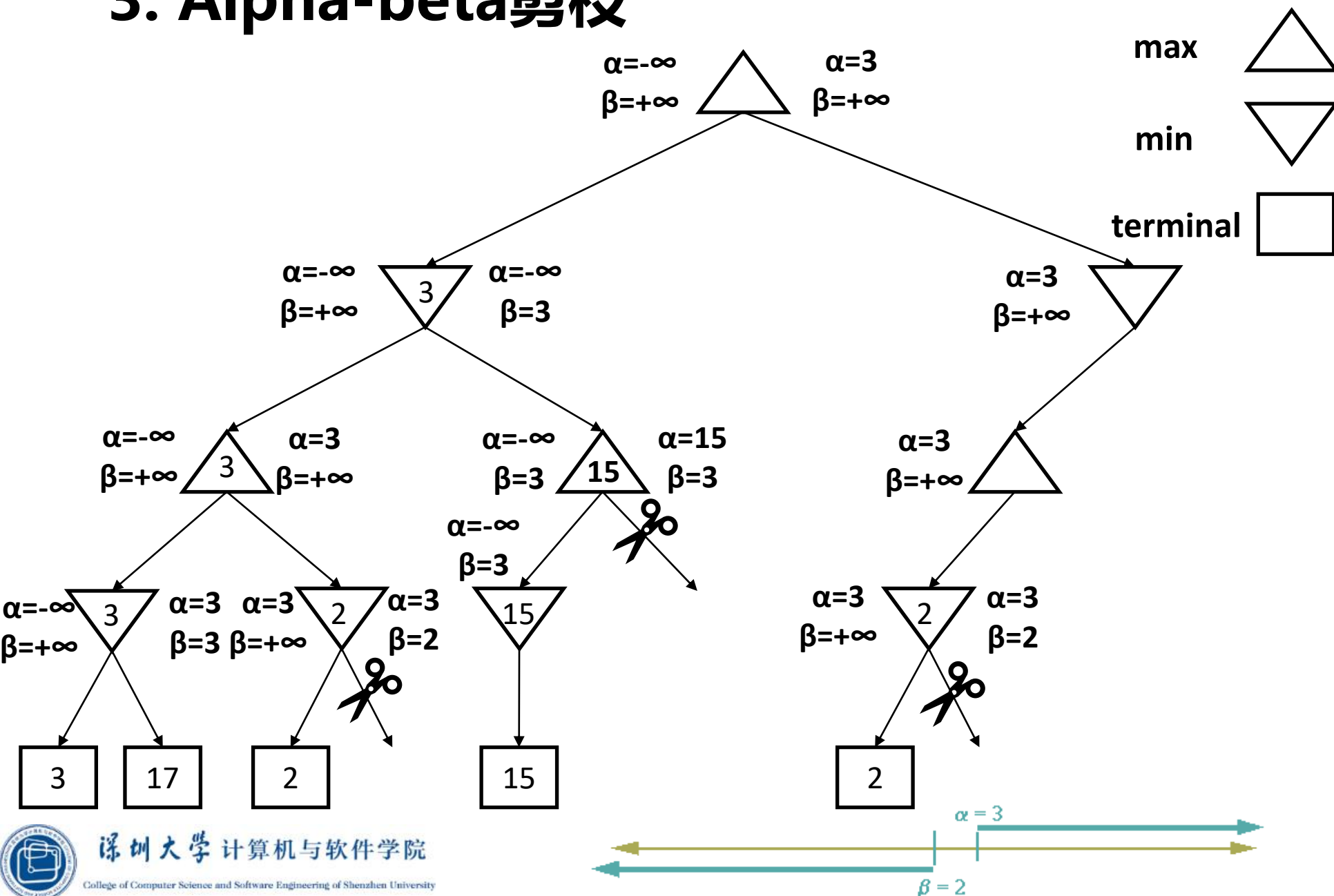
3. Alpha-beta剪枝



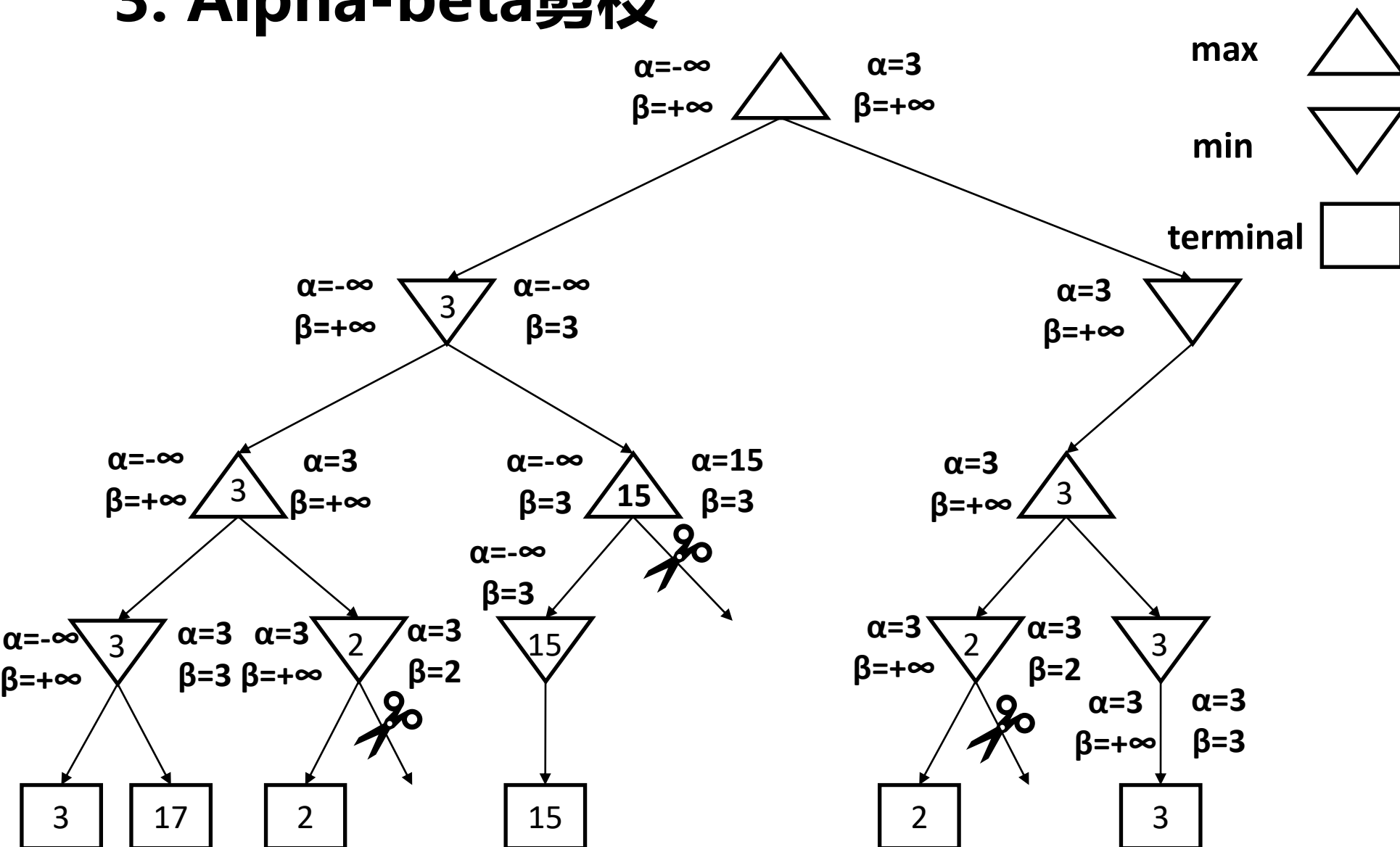
3. Alpha-beta剪枝



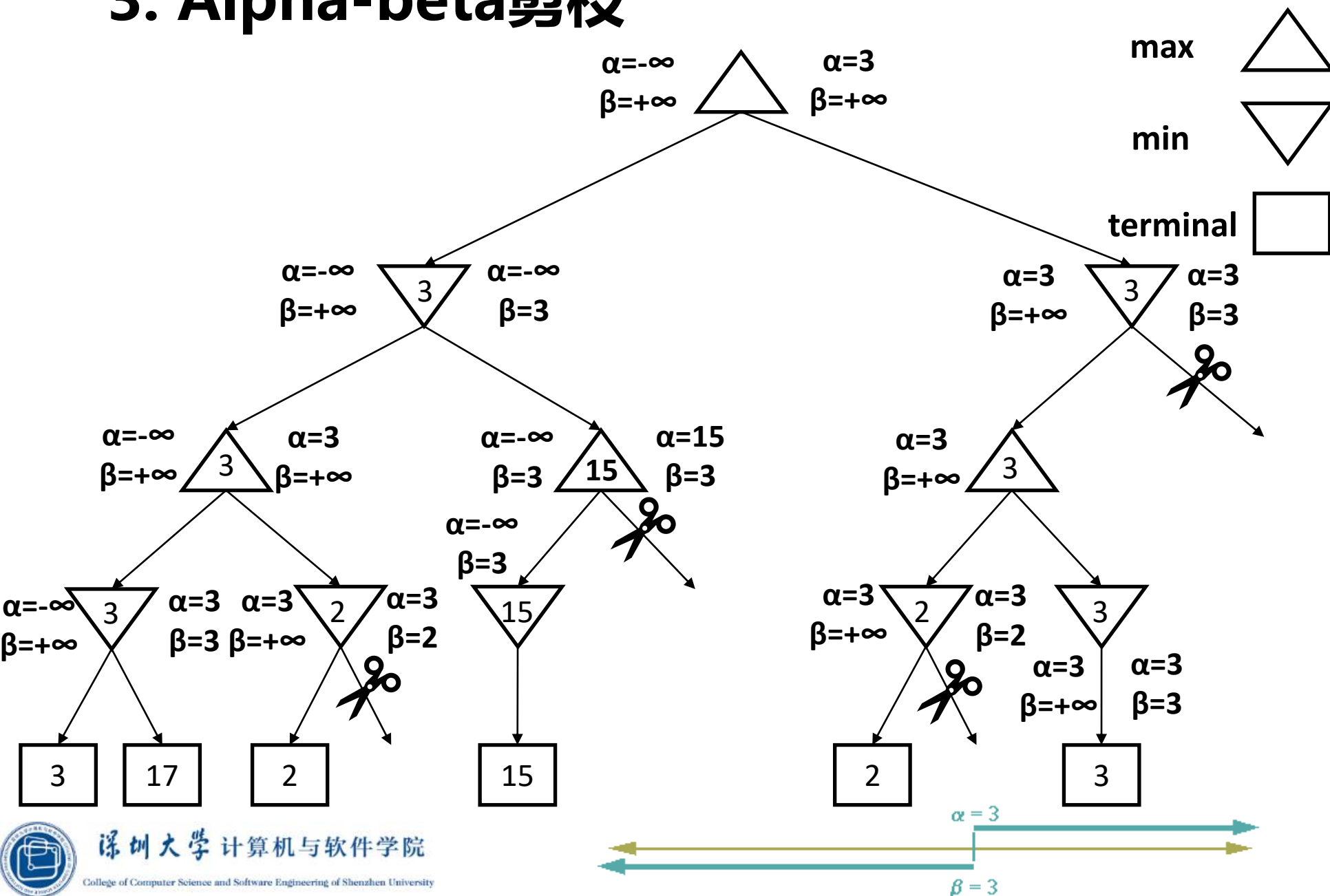
3. Alpha-beta剪枝



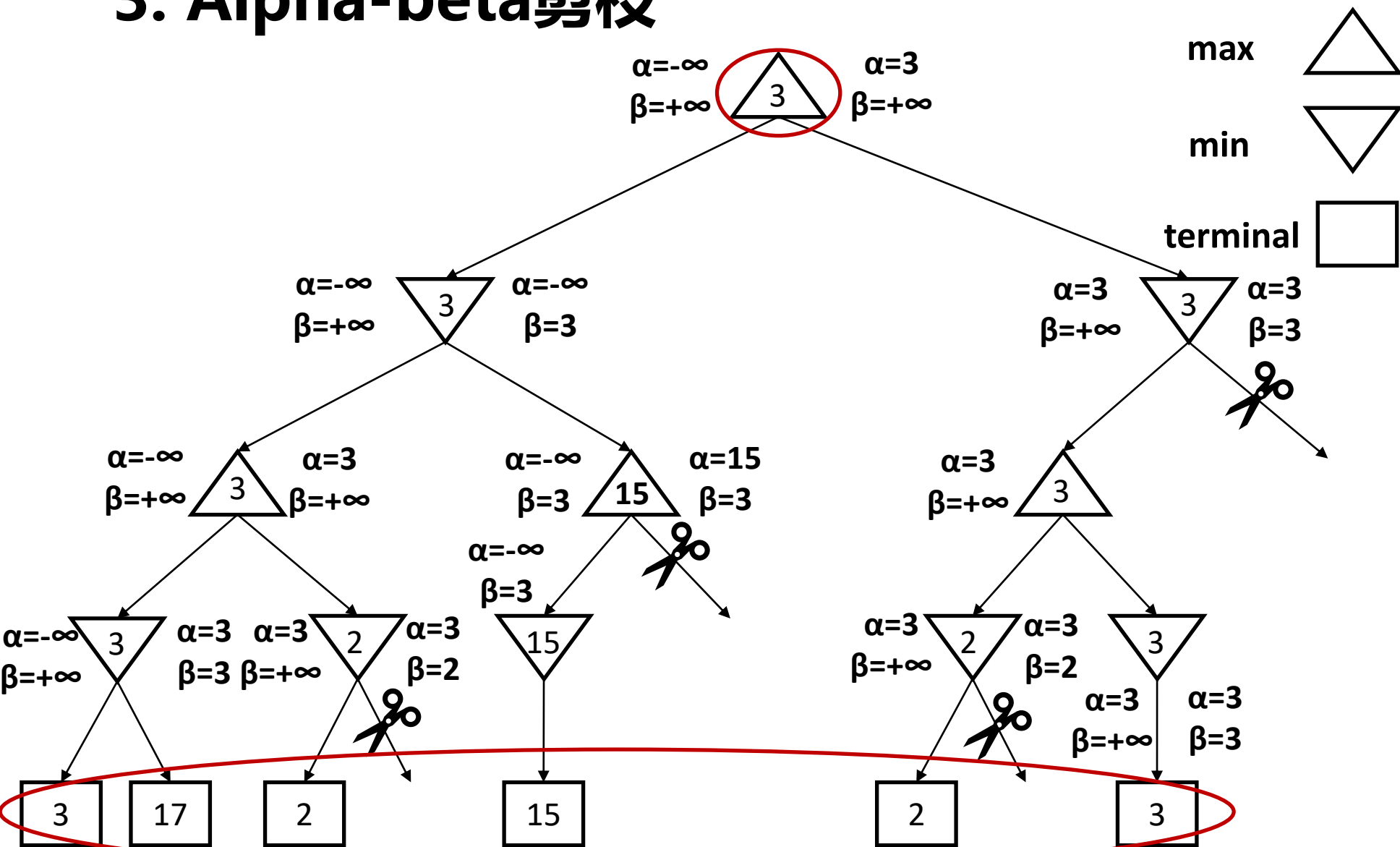
3. Alpha-beta剪枝



3. Alpha-beta剪枝



3. Alpha-beta剪枝



返回值为3，访问6个末端结点（完整12个）

3. Alpha-beta剪枝

Alpha-beta剪枝性质

- 剪枝不影响根结点minimax的值
- 中间结点值可能不同
- 子结点的次序影响剪枝的效率
 - 最好效果: $O(b^{m/2})$
 - 搜索代价还是过高, 需进一步优化

4. 评估函数

- Minimax和Alpha-beta都以末端结点效益为基础
- 末端结点**状态评价**至关重要
- 评估函数用于评估对局形式，选择于AI 最有利步骤
- 评估函数一般定义：

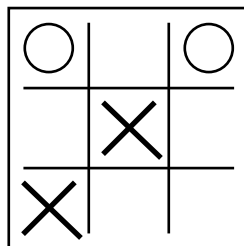
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

其中， s 为末端状态， $f_i(s)$ 为某种评估指标值， w_i 为该指标的权重。

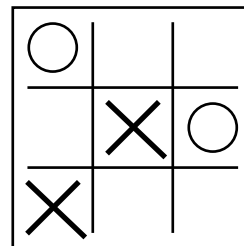
- 根据具体问题设计

× : AI

○ : Player



$E(s_1)=$

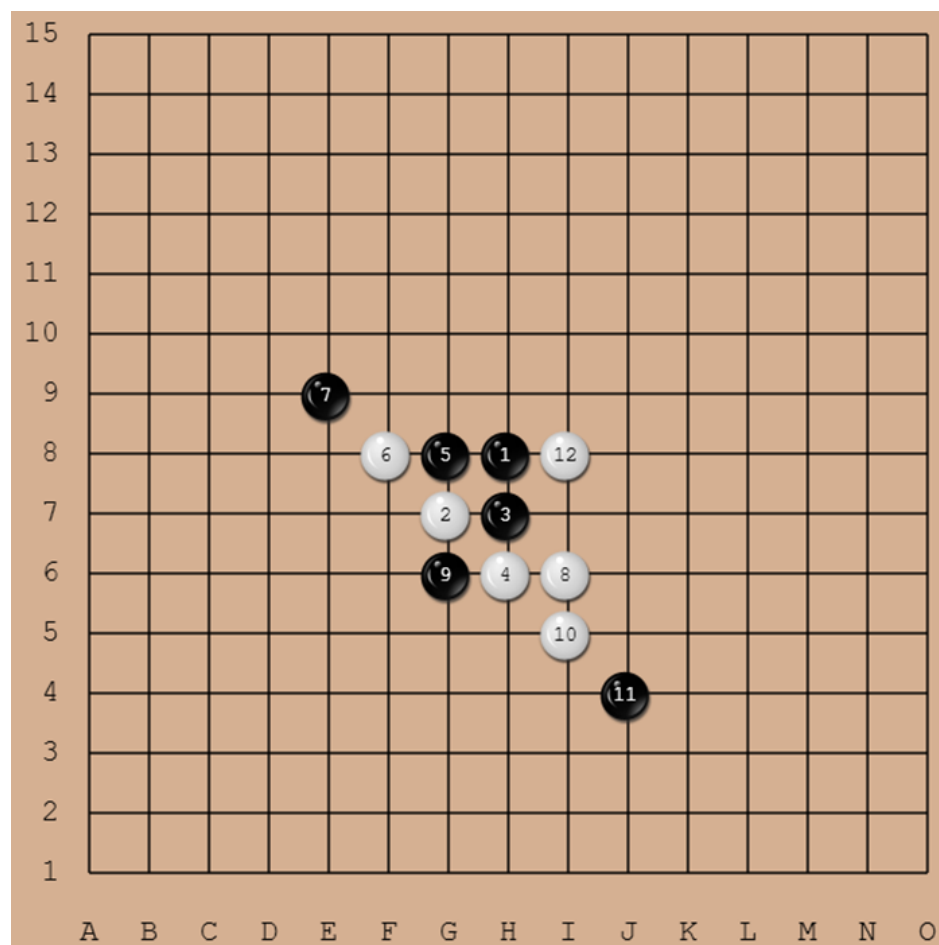


$E(s_2)=$

哪种状态更好？

5. 博弈扩展

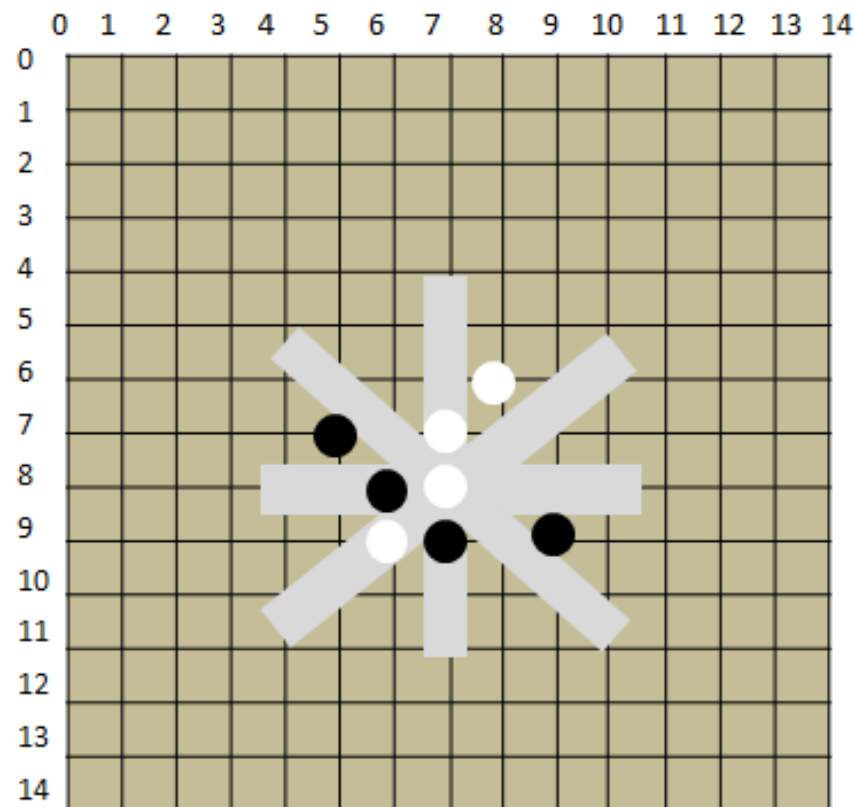
五子棋是博弈游戏当中较简单的一种玩法，规则、赢法都很简单，其 AI 的设计也相对简单。在实现过程中可以运用**深度搜索**算法，**极小值极大值**算法，**Alpha-beta 剪枝**算法和评估函数等。



5. 博弈扩展

五子棋规则

五子棋的规则（黑棋先行）实际上有两种：**有禁手**和**无禁手**。由于无禁手的规则比较简单，因此被更多人所接受。其实，对于专业下五子棋的人来说，一般**采用**有禁手规则。



5. 博弈扩展

评估函数

“有禁手”规则比较复杂，涉及到比较多下棋方面的技巧，而且对算法的思路没有丝毫影响，所以下面我们主要考虑无禁手规则下的AI设计。为了便于设计，简单规定如下棋型，其中**0表示空格**，**1表示黑棋**，**2表示白棋**。

棋型	解释	棋型	解释	棋型	解释	棋型	解释
成5	11111	跳活5	01101110、01110110	眠4	211110、011112	跳眠5	21101110、01101112
活4	011110	跳活4	0101110、0110110、0111010	眠3	21110、01112	跳眠4	0101112、2101110...
活3	01110	跳活3	010110、011010	眠2	2110、0112	跳眠3	010112、210110...
活2	0110	跳活2	01010	眠1	210、012	跳眠2	01012、21010
活1	010	跳死5	21101112、21110112	死2	2112	跳死2	21012
死4	211112	跳死4	2101112、2110112、2111012	死1	212	跳死1	212
死3	21112	跳死3	210112、211012				

5. 博弈扩展

评估函数

不同的棋型，其优先级不同。对于每一种特定的棋型，都需要相应的估值来反映其优劣情况。对棋型进行了以下打分：

棋型	分数	棋型	分数	棋型	分数	棋型	分数
成5	10000	跳活5	900	眠4	800	跳眠5	800
活4	1000	跳活4	800	眠3	40	跳眠4	700
活3	100	跳活3	60	眠2	3	跳眠3	35
活2	10	跳活2	6	眠1	0	跳眠2	2
活1	1	跳死5	600				
死4	0	跳死4	500				
死3	0	跳死3	0				
死2	0	跳死2	0				
死1	0	跳死1	0				

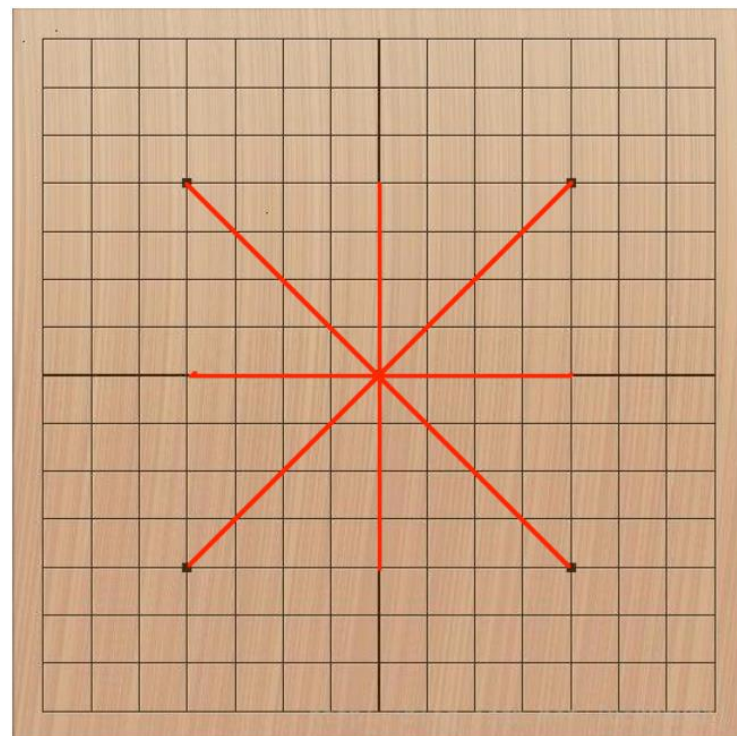
5. 博弈扩展

评估函数

赢法数组: 博弈类游戏的结束一定是因为某一方触发了获胜条件或是双方在规定时间、范围内没有分出胜负, 即打成平手; 因此我们需要记录所有的获胜条件, 在对弈者每一步决策后判断其是否触发了获胜条件, 而存储获胜条件的结构我们就称之为赢法数组

- 胜负判定
- 评估指标

五子棋的所有赢法存入该数组,
根据最后一个落子的情况观察
水平, 竖直、45度和135度的线
如能成连续五个棋子, 可判胜负



5. 博弈扩展

评估函数

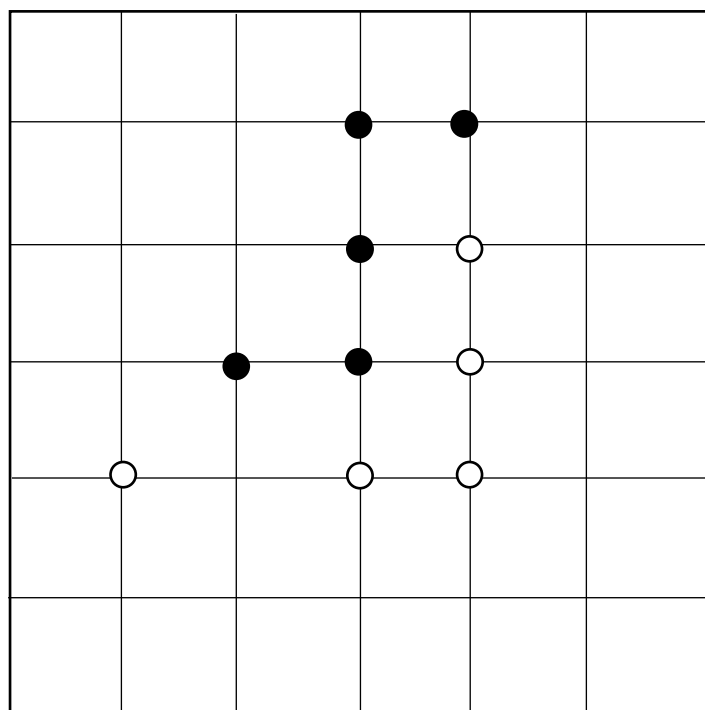
有了所有的赢法之后，我们就可以给出得分了：对于每一种赢法，我们需要统计该赢法的位置上己方棋子和对方棋子的数目，同样可以存储在一个数组中，然后在每一次需要对当前局势评估时，如果某种赢法上只有己方棋子或只有对方棋子，根据棋子的数目就是该点的权重，也就是 $f_i(s)$ 得出的值，在和预设的评分 w_i 相乘，将总分相加，就得出了当前局势下某一方的评估分数。

通过这样的方法，我们可以得到两个数字，分别是AI和玩家的当前优势值，我们将这两个数字做差，**返回 MAX - MIN 的值**，这就是该种走法的分数，也就是博弈树种叶子节点的值，值越大对AI更有利，值越小对玩家更有利。

5. 博弈扩展

实例

假设有这样一个7*7的棋盘，棋盘如下，黑子是AI,白子是玩家，提前两步预判当前棋局，请问下一步AI会走哪步？左下角坐标为 $(0, 0)$ ，右上角坐标为 $(6, 6)$ 。



首先列出下一步AI可能会下的坐标:

$(0,0), (0,1), \dots, (0,6),$
 $(1,0), (1,1), \dots, (1,6),$
 $(2,0), (2,2), (2,5), (2,6),$
 $(3,0), (3,1), (3,5), (3,6),$
 $(4,0), (4,1), (4,2), (4,5), (4,6),$
 $(5,0), (5,1), (5,2), (5,5), (5,6),$
 $(6,0), (6,1), \dots, (6,6)$

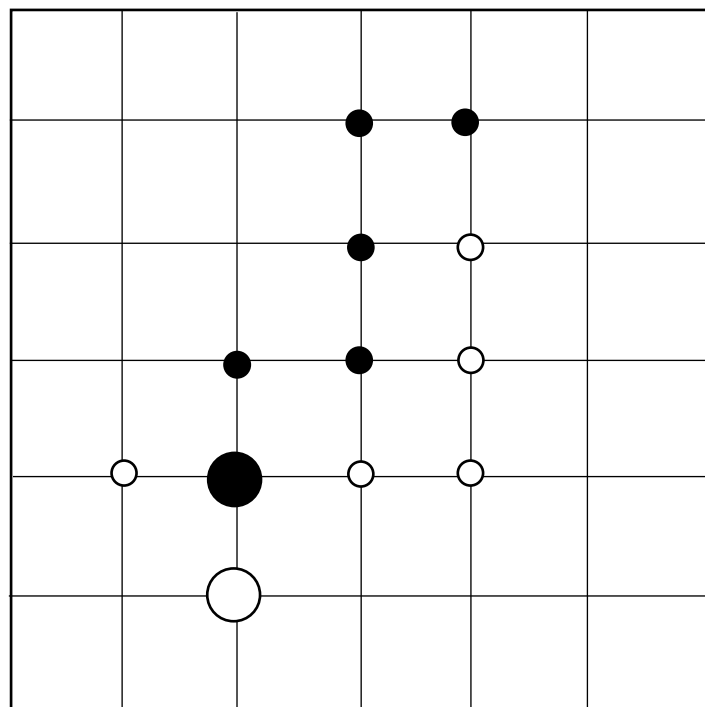
该步共有39种走法

对于每种走法，玩家都有38个点可下
那么提前两步就有 39×38 种走法

5. 博弈扩展

实例

若AI下的位置在 (2, 2) , 则列出所有玩家可能下的坐标, 以及按照之前所定的评估方法得到对应的局面评分



若玩家下的坐标为(1,2),计算当前局面评分:

1) AI-Max值:

水平: $3+10=13$

竖直: $3+40=43$

45度: $3+40+1=44$

135度: $1+1=2$

Max: $13+43+44+2=102$

2) 玩家-Min值:

水平: $1+3=4$

竖直: $1+40=41$

45度: $1+100=101$

135度: 10

Min: $4+41+101+10=156$

3) 得分: $\text{Score} = \text{Max} - \text{Min} = 102 - 156 = -54$

5. 博弈扩展

实例

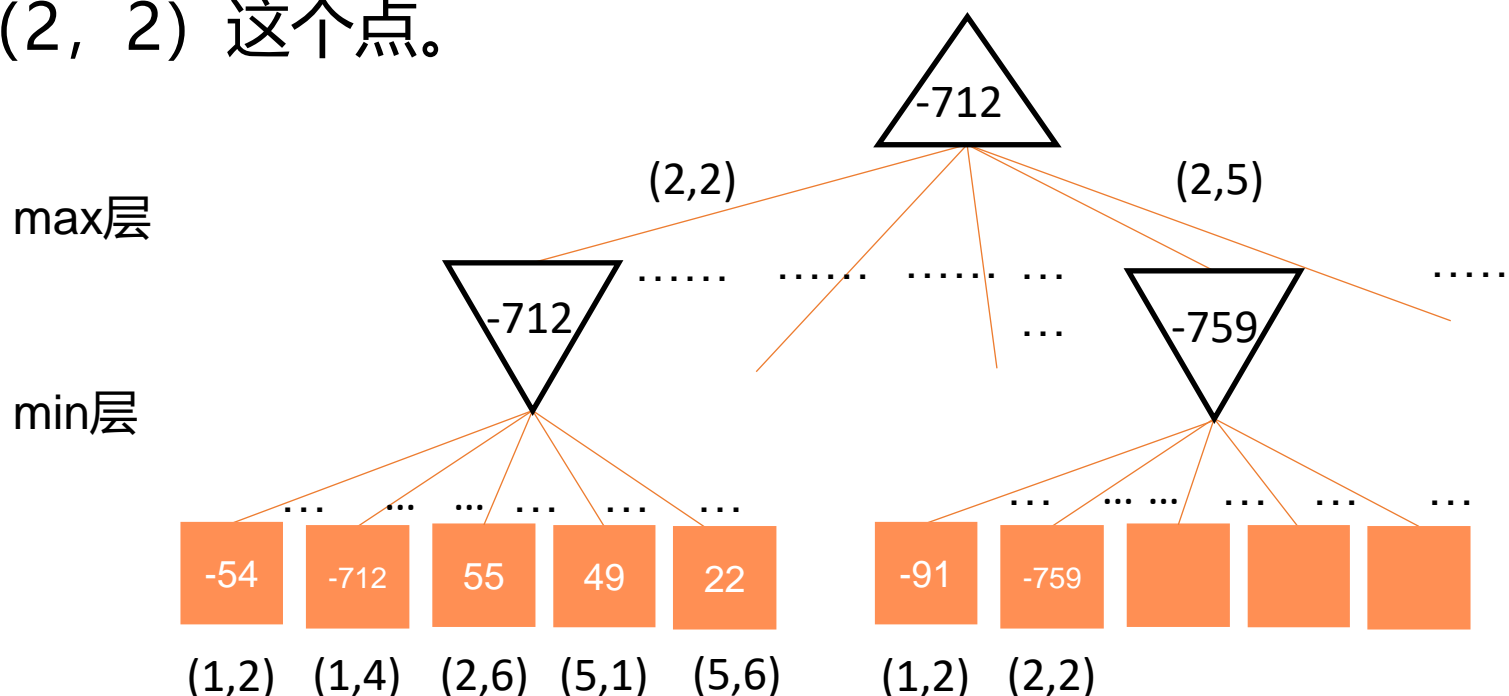
若AI下的位置在 (2, 2) , 则列出所有玩家可能下的坐标, 以及按照之前所定的评估方法得到对应的局面评分

坐标	E(s)	坐标	E(s)	坐标	E(s)	坐标	E(s)	坐标	E(s)	坐标	E(s)
(0,0)	53	(1,0)	53	(2,0)	52	(4,0)	52	(5,2)	44	(6,4)	51
(0,1)	26	(1,1)	39	(2,5)	11	(4,1)	46	(5,5)	41	(6,5)	12
(0,2)	51	(1,2)	-54	(2,6)	55	(4,2)	45	(5,6)	22	(6,6)	53
(0,3)	51	(1,3)	39	(3,0)	50	(4,5)	-42	(6,0)	53		
(0,4)	-408	(1,4)	-712	(3,1)	39	(4,6)	51	(6,1)	52		
(0,5)	52	(1,5)	47	(3,5)	37	(5,0)	52	(6,2)	52		
(0,6)	53	(1,6)	52	(3,6)	52	(5,1)	49	(6,3)	11		

5. 博弈扩展

实例

重复以上流程，我们可以得到走两步之后所有局面评分，构造以下博弈树。上三角代表AI，下三角代表玩家。根据极大极小值搜索及Alpha-beta剪枝，我们可以得出AI下一步将下到 (2, 2) 这个点。





5. 博弈扩展

优化

到目前为止，我们使用传统的 Alpha-Beta 搜索结合估值函数的五子棋算法完成一个简单的五子棋对弈程序。虽然估值尽力做到细致、全面，但由于 Alpha-Beta 搜索存在博弈树算法中普遍存在的一个缺点——随着搜索层数的增加，算法的效率大大下降。所以搜索的效率还是不理想，五子棋程序的“智力”也不高。

因此，在上述基础上，我们继续研究，通过对 Alpha-Beta 搜索算法的优化与修正，针对五子棋本身的特点和规律，提出采取以下优化措施，显著地提高了五子棋程序对弈的水平 and 能力。



5. 博弈扩展 优化

1 减少搜索范围

五子棋棋盘大小为 15×15 ，传统算法中计算机每走一步都要遍历整个棋盘，对于棋面上所有空位都进行试探性下子并估值，这样大大影响算法的效率。根据经验，只要考虑距以**棋子为中心边长为4的正方形区域**即可。这样便缩小了搜索空间，提高搜索效率。



5. 博弈扩展 优化

2 设置下棋风格

对一个落子估值的时候，首先在己方的角度对其进行评估 $Value_Me$ ；随后在对手的角度再进行一次评估 $Value_Enemy$ ；通过 $Value = K1 * Value_Me + K2 * Value_Enemy$ 而获得最终的估值结果。其中 $K1$ 和 $K2$ 是一对关键系数，当 $K1 / K2$ 越大的时候，就表示计算机落子的攻击性更强，反之则表示计算机落子考虑较多的是阻断对方的落子，防守性更强些。 $K1$ 、 $K2$ 的值可以由玩家设定，也可以由计算机根据当前棋面自己决定。与传统算法相比，这样得到的五子棋程序更加智能。



5. 博弈扩展 优化

3 利用多线程

我们其实可以利用多核技术，利用多个线程，让算法实现并行计算，提高AI的速度。我们在第一层用一个线程分配器把第二层的候选节点分配给多个线程，每个线程包含着从第二层一个候选节点开始的搜索，然后等所有线程结束后，将所有线程的结果进行汇总，选出最大值。并行的程序，可以将速度提高一倍左右。



5. 博弈扩展 优化

4 增大搜索层数

理想状态下，为了尽可能提高计算机下棋的“智力”，搜索层数应该越大越好。实际上，由于博弈树异常庞大，搜索层数的增加将会导致算法的效率大大下降。提出了多线程、分布式并行的方法。其缺点是比较消耗资源。但设计者可以根据已有的资源条件，**调整设置搜索层数**，最有效地进行搜索。



5. 博弈扩展 优化

5 使用置换表

在Alpha-beta搜索过程中，为了避免重复搜索已经搜索过的节点，加快搜索速度，可以使用一张表格记录每一节点的搜索结果，对任意节点向下搜索之前先察看记录在表中的这些结果。如果将要搜索的某个节点在表中**已有记录**，就直接**利用记录**下来的结果。我们称这种方法为置换表（Transposition Table, 简称 TT）。



5. 博弈扩展 优化

6 启发式搜索

五子棋游戏开局阶段有很多“定式”。将“定式”的格局及其走法当成棋谱保存在数据库中，若发现当前**格局存在于棋谱**中，则我们不需要搜索，**直接按照棋谱下棋**，即利用棋谱进行启发式搜索，这样便加快了搜索的速度。该方法的关键是棋谱的模糊匹配以及棋谱的存储结构。



5. 博弈扩展 优化

7 让AI自主学习，不再同一个地方犯错

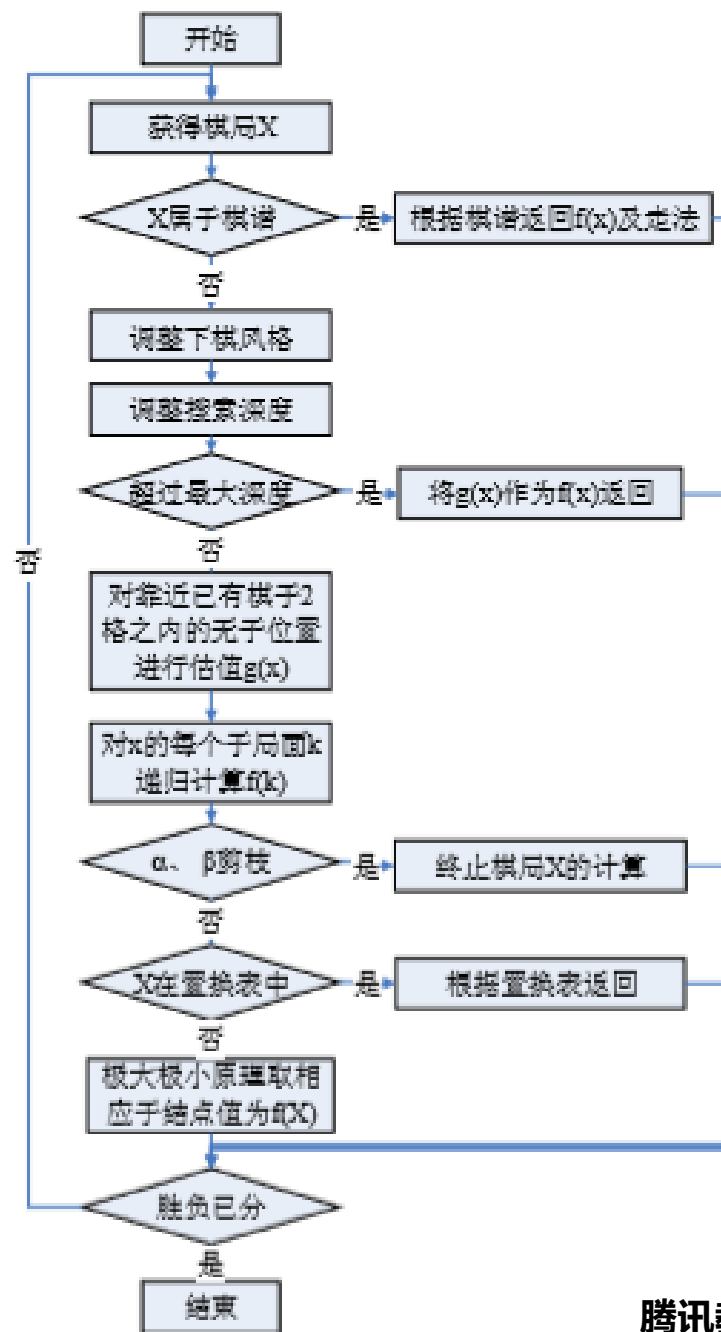
上面的算法还没有自主学习的能力,这样AI在下棋时还可能会重蹈覆辙。因此在每盘棋结束时,如果AI输,则进行大于搜索深度的步数回退。可以把倒数为搜索深度数目的局面定为目标局面,从倒数深度加一步局面进行预测,找到不会导出必败目标局面的局面。然后记录下这个局面和前面的局面,并据此修改评分函数。这样AI就不会犯曾经犯过的错误,达到自学习的效果



5. 博弈扩展 优化

总结以上优化措施，如图所示，我们可以得出智能五子棋博弈算法按深度优先扩展一个棋局的流程。

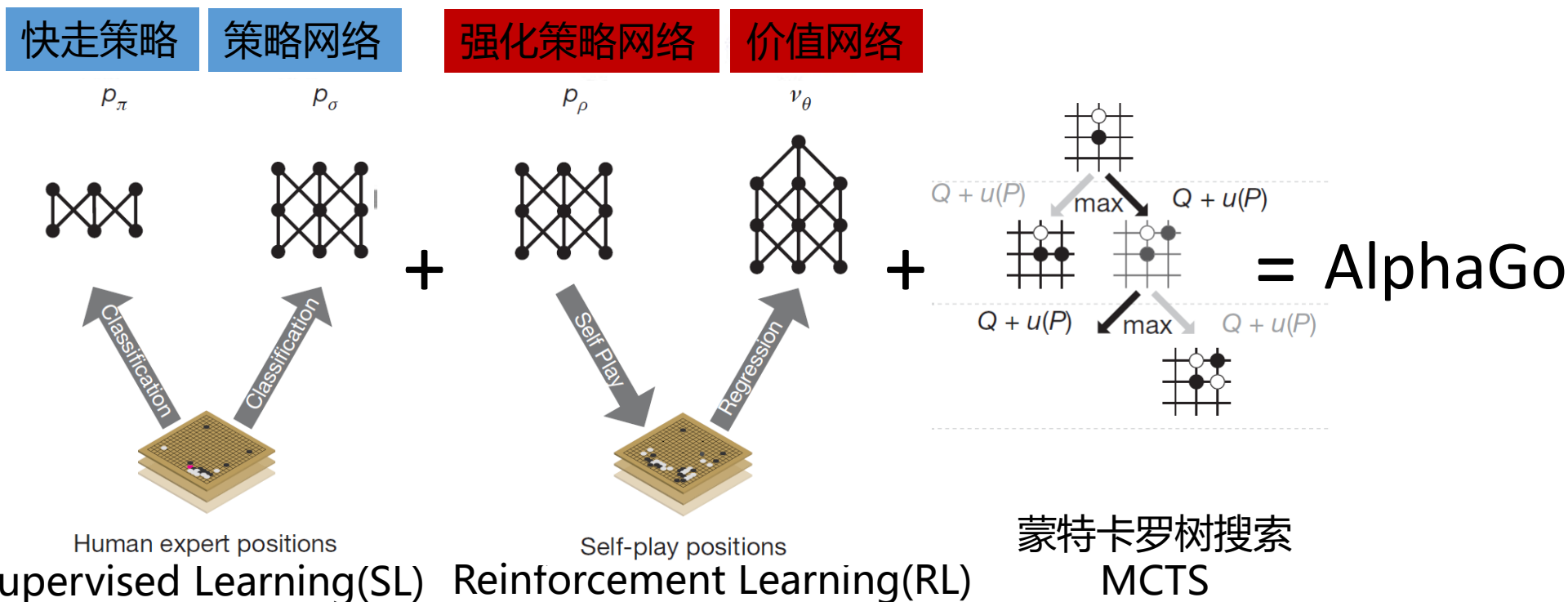
说明：若棋局 k 是棋局 x 的某个子节点，则 $g(k)$ 表示对当前棋面 k 应用估值函数得到的估值， $f(x)$ 表示对 x 的所有子节点应用最大最小原理后得到的值。该算法有效地减少了搜索的范围，增加了程序的智能。



5. 博弈扩展

AlphaGo

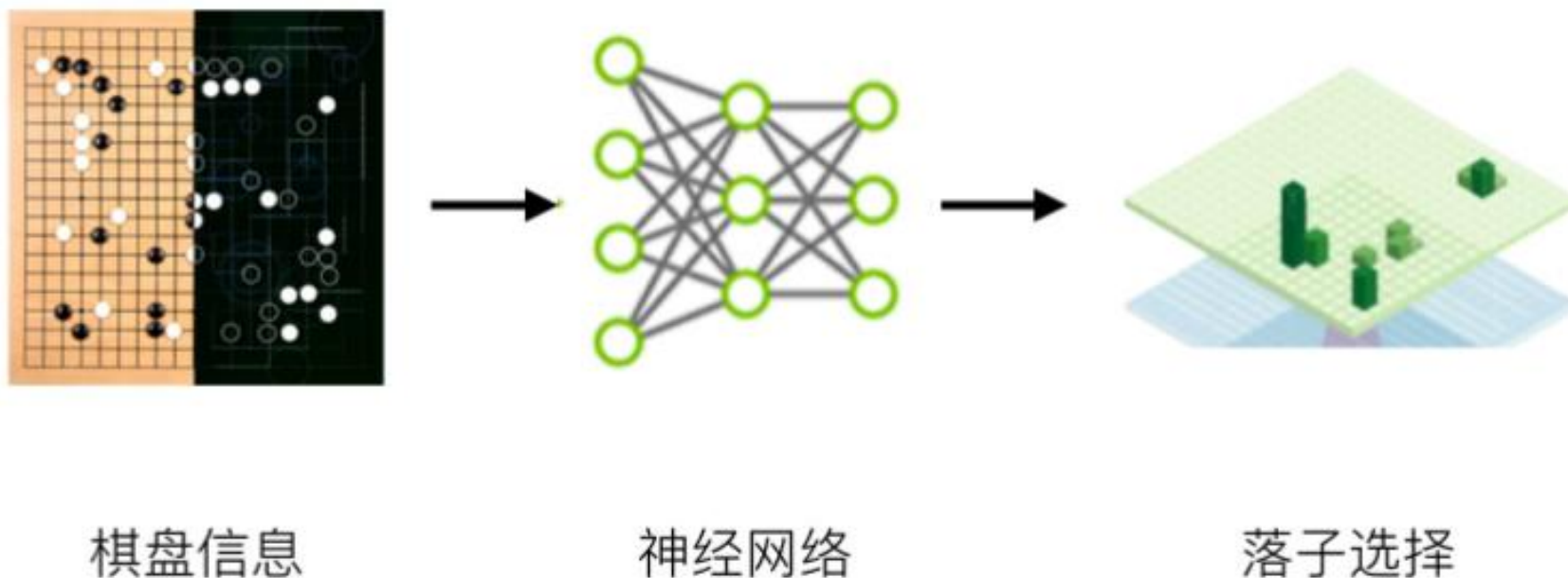
围棋是典型的对抗博弈游戏。棋盘是19*19，初始的分支因子为361，搜索空间巨大，而且评估函数设计十分困难（全局观），AlphaGo引入了更先进的博弈技术。



5. 博弈扩展

AlphaGo

策略网络 (Policy Network)：给定当前盘面状态作为输入，输出下一步棋在棋盘空白位置的**落子概率**。



5. 博弈扩展

AlphaGo

策略网络 (Policy Network) : 给定当前盘面状态作为输入, 输出下一步棋在棋盘空白位置的**落子概率**。

AlphaGo通过3万多专业棋手对局的棋谱来离线训练网络, 并通过3000万盘自我对弈, 强化策略网络 (3ms 每步) 。

快速走棋策略 (Rollout Policy): 当前盘面快速行棋至最终胜负结果。

利用局部特征和线性模型训练, 精度较策略网络低, 但速度快 ($2 \mu s$ 每步)

价值网络 (Value Network) : 根据当前的状态直接评估出结果的输赢概率。

5. 博弈扩展

AlphaGo

在线对弈过程：

1. 根据当前盘面已经落子的情况提取相应特征；
2. 利用**策略网络**估计出棋盘空白位置的落子概率
3. 根据落子概率计算此处往下发展的权重；
4. 利用**价值网络**和**快速走棋网络**分别判断局势，得分相加为此处最后走棋获胜的得分；
5. 利用第四步计算的得分来更新之前走棋位置的权重；
6. 根据各位置的权重，进行**蒙特卡罗树**搜索。

5. 博弈扩展

AlphaGo-蒙特卡罗树搜索

基本思想：多次模拟博弈，并尝试根据模拟结果预测最优的移动方案。

模拟博弈：它是一系列从当前节点（表示博弈状态）开始，于端节点结束并计算出博弈结果（**快速走棋网络**确定行动）。

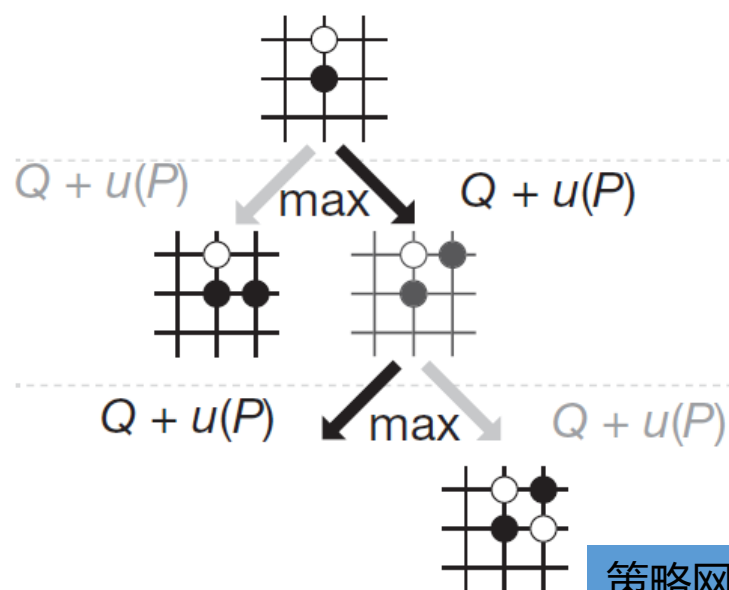
- 通过SL策略网络预测前L个行动步骤。如果它转到叶节点，且该节点的访问次数大于阈值，则展开下一个节点；
- 价值网络评估取胜率,快速走棋网络评估仿真结果，结合两个评估并预测接下来的L步骤；
- 将评估结果作为当前棋盘下一步行动的Q值。Q值越大，在以后的模拟中选择的次数越多；
- 结合Q值和SL策略网络，再次模拟。如果有两个相同的动作，则计算Q值的平均值。
- 重复以上步骤n次，选择模拟中最频繁选择的动作。

5. 博弈扩展

AlphaGo-蒙特卡罗树搜索

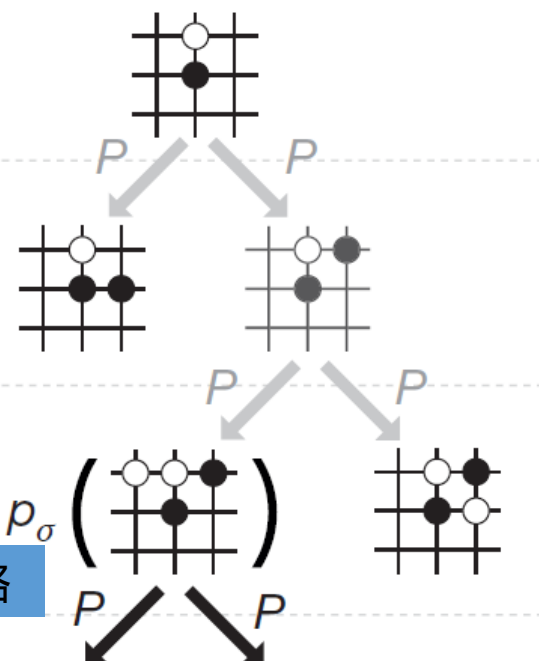
a

Selection

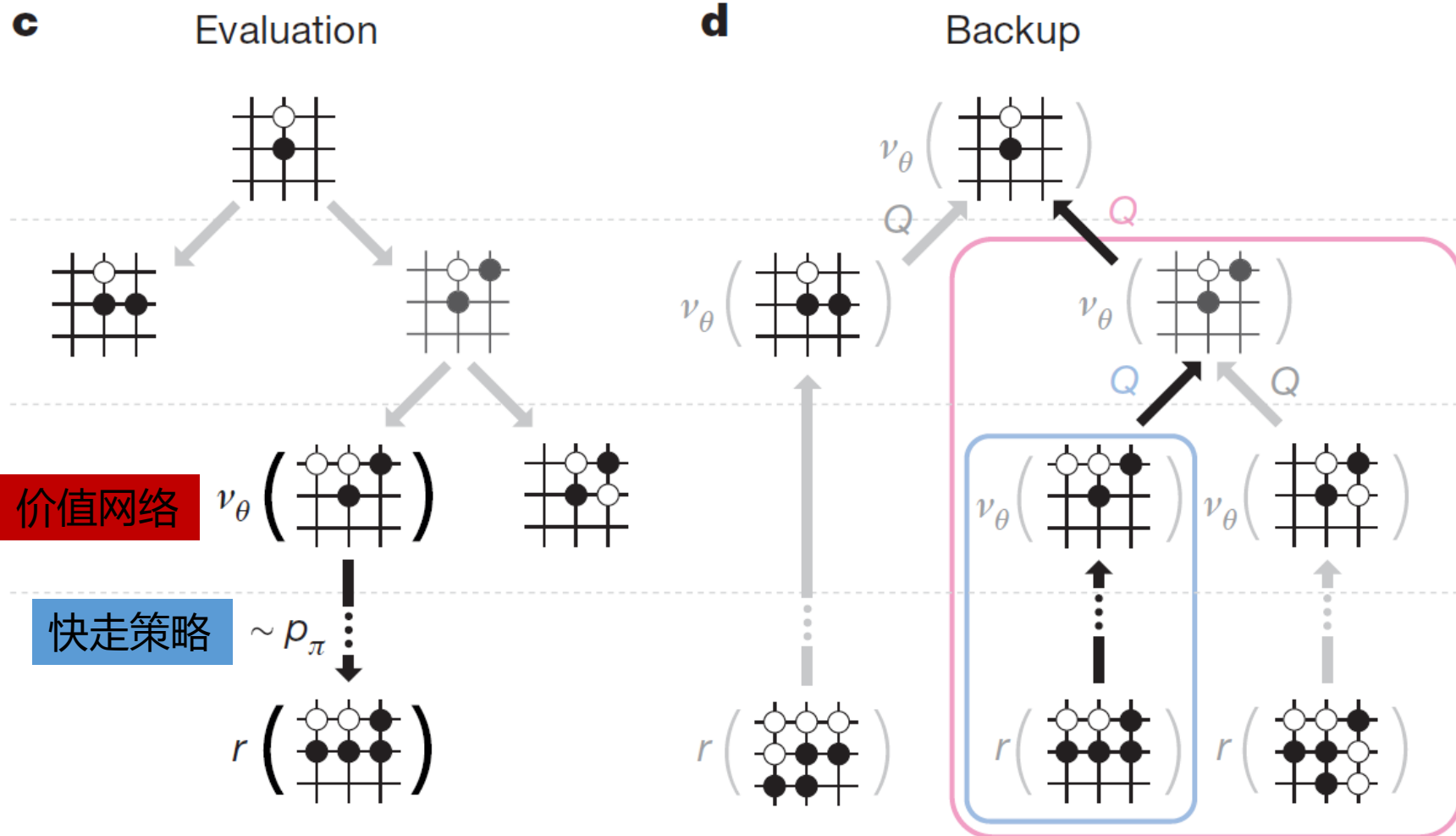


b

Expansion

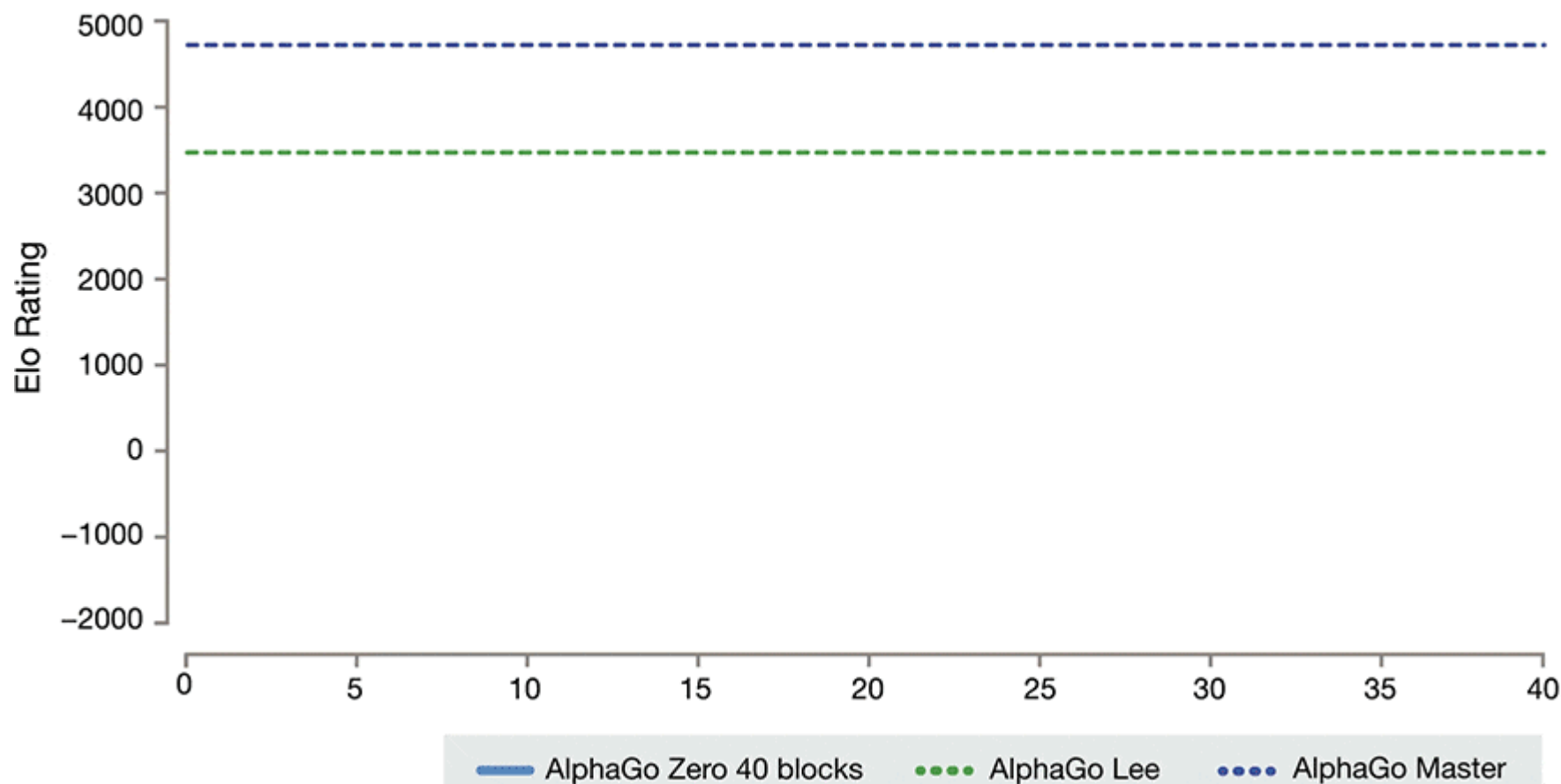


AlphaGo-蒙特卡罗树搜索



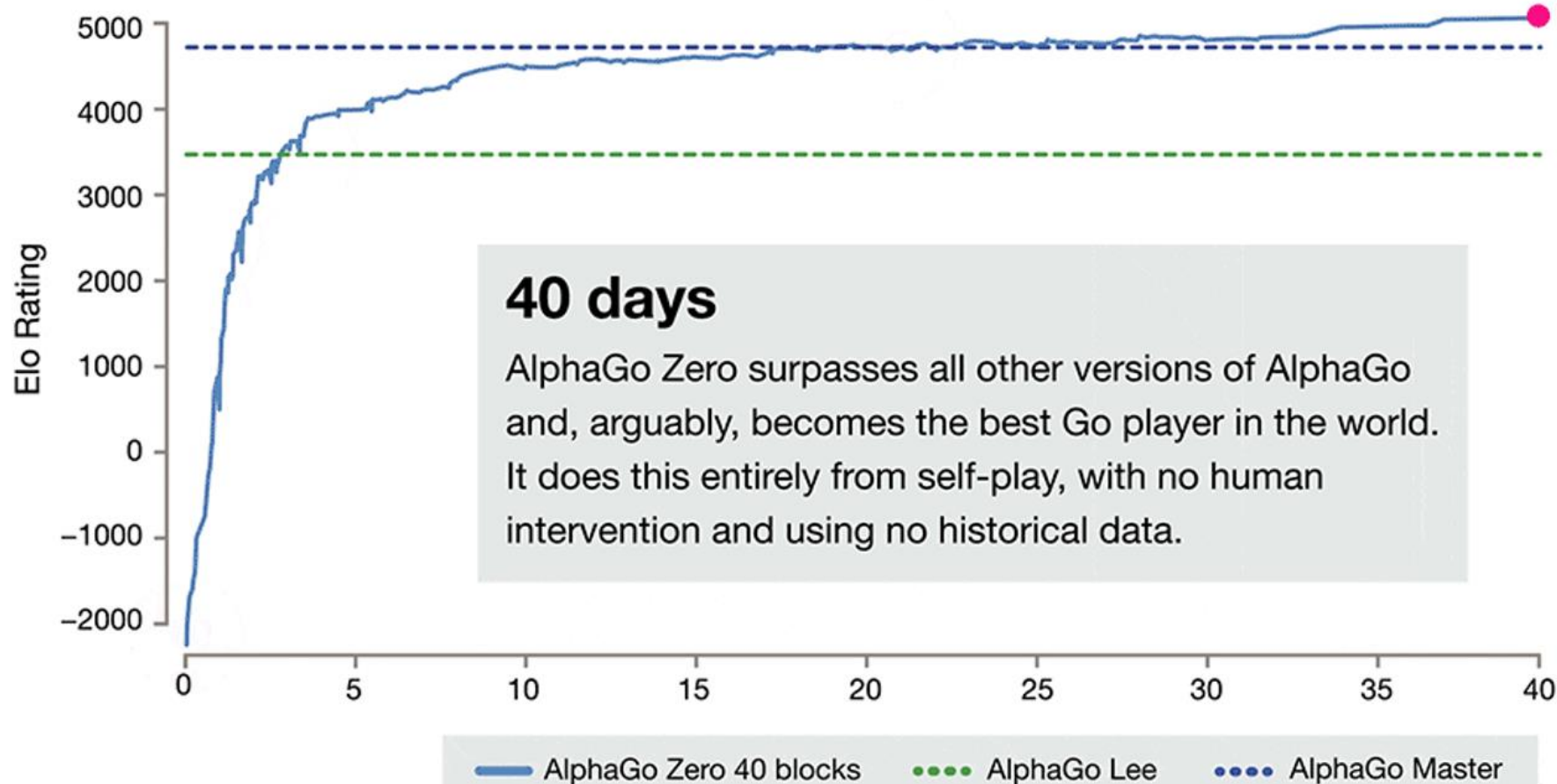
5. 博弈扩展

AlphaGo-Zero



5. 博弈扩展

AlphaGo-Zero

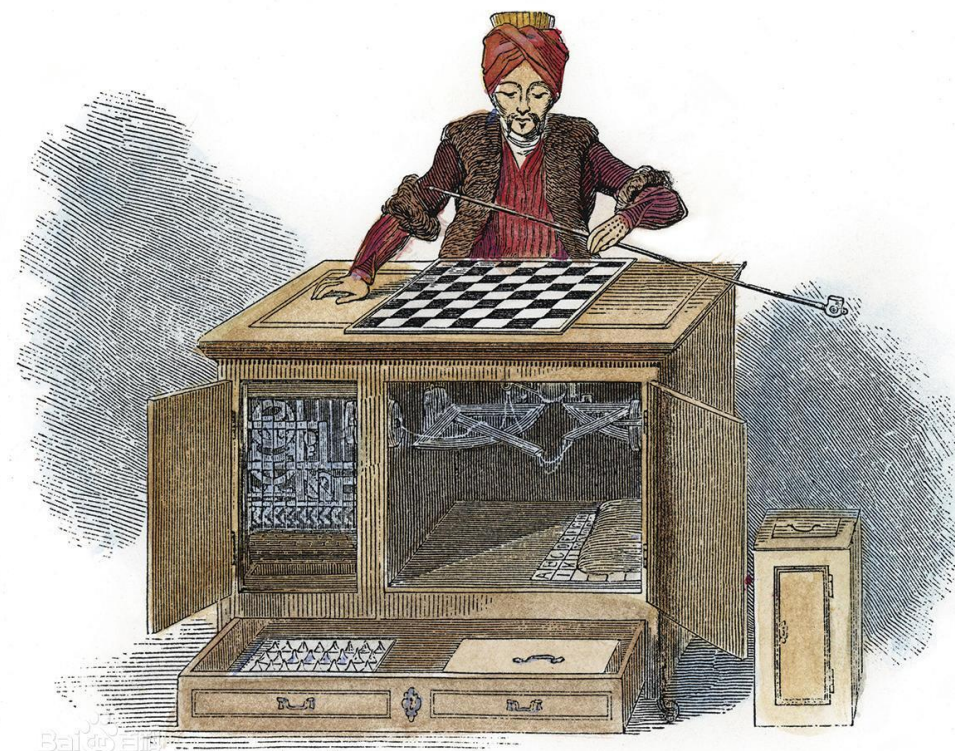


5. 博弈扩展

土耳其行棋傀儡

土耳其行棋傀儡是18世纪晚期的一个自动下棋装置，但后来被证明是一场骗局。它是奥地利的沃尔夫冈·冯·肯佩伦(1734–1804)在1770年为取悦玛丽娅·特蕾西娅女大公而建造并展出的，可以击败人类棋手，以及执行骑士巡逻。

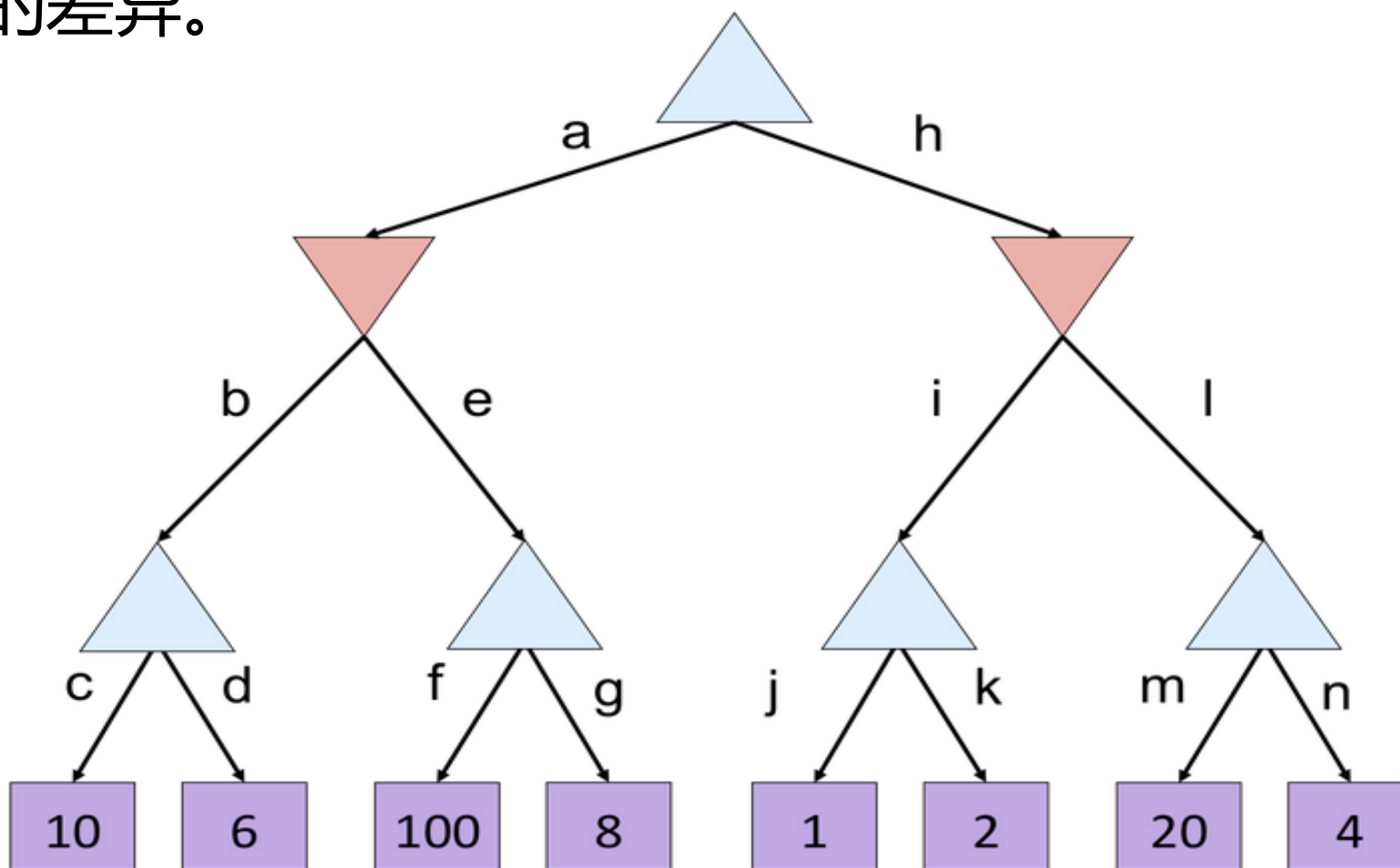
<https://baike.baidu.com/item/%E5%9C%9F%E8%80%B3%E5%85%B6%E8%A1%8C%E6%A3%8B%E5%82%80%E5%84%A1/1832164?fr=aladdin>



THE AUTOMATON CHESS PLAYER.

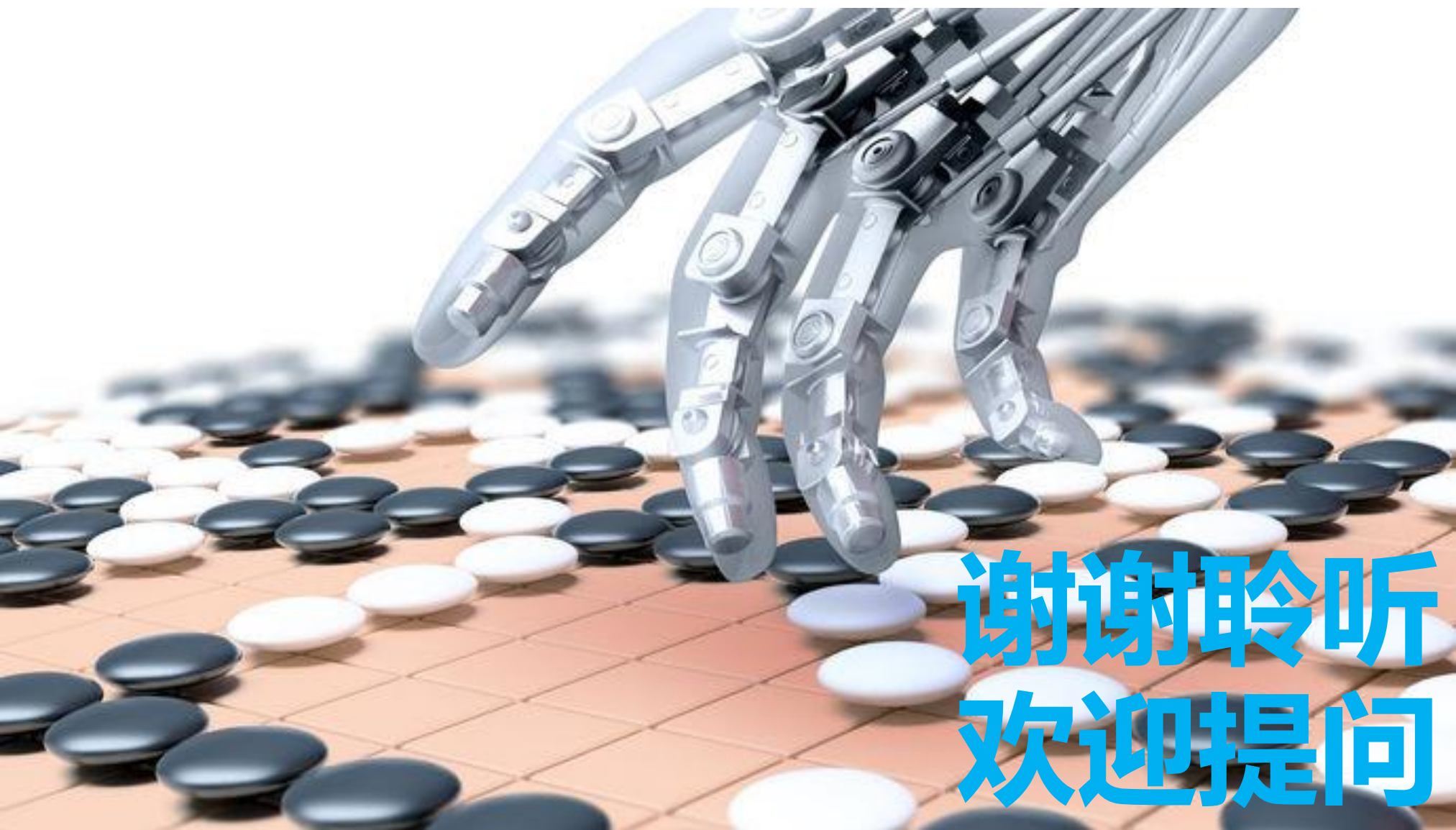
练习题

利用Alpha-beta剪枝进行搜索，并比较剪枝前后搜索的差异。





深圳大学
SHENZHEN UNIVERSITY



谢谢聆听
欢迎提问



深圳大学 计算机与软件学院

College of Computer Science and Software Engineering of Shenzhen University