



中国科学技术大学  
University of Science and Technology of China

# 人工智能讲义

## 博弈与对抗性搜索

April 2, 2022

# Outline

- ① 博弈论简介
- ② 博弈、MDP 和搜索
- ③ 博弈树：描述博弈的扩展形式
- ④ 基于博弈树的推理

# 囚徒困境：问题描述

## 囚徒困境问题描述

- 假设有两个疑犯被警察抓住。并且被分开关押在不同的囚室。警察强烈怀疑他们和一场抢劫案有关，但是，没有充足的证据。然而，他们都拒捕的事实也是可判刑的。两个疑犯都被告知以下结果：
- “如果你坦白，而另外一人抵赖，则你马上释放；另外一人将承担全部罪行，将会被判刑 10 年
- 如果你们都坦白，你们的罪行将被证实。但由于你们有认罪的表现——判刑 4 年。
- 如果你们都不坦白，那么没有证据证明你们的抢劫罪，我们将以拒捕罪控告你们——判刑 1 年。
- 另外一方也正在接受这样的审讯。你是坦白还是抵赖？”

## 囚徒困境：形式化描述

		疑犯2	
疑犯1	抵赖	抵赖	坦白
	坦白	-1, -1	-10, 0
		0, -10	-4, -4

## 描述博弈的收益矩阵：解释说明

- 适用于 **二人** 博弈；
- 行代表一个 player1，列代表另一个 player2；行表示 player1 的不同策略/行动；列表示 player2 的不同策略/行动；
- 每个单元格的数对含义为 (player1 的收益, player2 的收益)；
- 一个简单的表格清晰地描述了复杂的博弈问题。

# 囚徒困境问题：求解过程

疑犯2			
	抵赖		
疑犯1	抵赖	-1, -1	-10, 0
疑犯1	坦白	0, -10	-4, -4

疑犯2			
	抵赖		
疑犯1	抵赖	-1	-10, 0
疑犯1	坦白	0	-4, -4

疑犯2			
	抵赖		
疑犯1	抵赖	-1, -1	-10, 0
疑犯1	坦白	0, -10	-4, -4

疑犯2			
	抵赖		
疑犯1	抵赖	-1, -1	-10, 0
疑犯1	坦白	0, -10	-4, -4

疑犯2			
	抵赖		
疑犯1	抵赖	-1, -1	-10, 0
疑犯1	坦白	0, -10	-4, -4

疑犯2			
	抵赖		
疑犯1	抵赖	-1, -1	-10, 0
疑犯1	坦白	0, -10	-4, -4

疑犯2			
	抵赖		
疑犯1	抵赖	-1, -1	-10, 0
疑犯1	坦白	0, -10	-4, -4

# 完美信息下的静态博弈

## 博弈的基本要素

- 参与者/player:  $P_1, P_2, \dots, P_n, n \geq 2$ ;
- 策略集/行动集: 每个参与者都有一个行动备选项/可能的策略, 策略选中后执行就变成了“行动”;  $P_i$  的行动集合记为  $\{a_{i1}, a_{i2}, \dots\}$ ;
- 收益/payoff:  $P_i(a_{1j_1}, a_{2j_2}, \dots, a_{nj_n})$  表示参与者  $P_i$  在所有人的策略为  $(a_{1j_1}, a_{2j_2}, \dots, a_{nj_n})$  时的收益。

## 进一步解释

- “完美信息”是指每个参与者对博弈信息完美了解; 了解别人如同了解自己;
- “静态博弈”是指所有参与者同时选择策略并行动;
- 参与者都是“理性的”: 追求自己利益的最大化; 也知道其他人都是如此;
- 独立决策: 参与者之间不协商。

# 详解博弈求解过程

		疑犯2	
		抵赖	坦白
疑犯1	抵赖	-1	-10, 0
	坦白	0, -10	-4, -4

		疑犯2	
		抵赖	坦白
疑犯1	抵赖	-1, -1	-10, 0
	坦白	0, -10	-4, -4

寻找给定其他参与者的策略时的最佳应对

- 如上图所示，当疑犯 2 采用“抵赖”策略时，疑犯 1 的 **最佳应对**是“坦白”；
- 最佳应对：**针对参与人  $P_2$  的策略  $T = \text{'抵赖'}$ ，若参与人  $P_1$  采用策略  $S = \text{'坦白'}$  产生的收益大于或等于自己的任何其他策略，则称参与人  $P_1$  的策略  $S$  是参与人  $P_2$  的策略  $T$  的最佳应对。 $P_1(S, T) \geq P_1(S', T)$ , 其中  $S'$  是参与人  $P_1$  的所有策略。
- 最佳应对一定存在，不一定唯一。**
- 严格最佳应对**（最佳应对的定义中  $\geq$  变成  $>$ ）不一定存在。

# 详解博弈求解过程

		疑犯2	
		抵赖	坦白
疑犯1	抵赖	-1, -1	-10, 0
	坦白	 0, -10	 -4, -4

		疑犯2	
		抵赖	坦白
疑犯1	抵赖	-1, -1	-10, 0
	坦白	 0, -10	-4, -4

最佳应对  $\Rightarrow$  占优策略

- 如上图所示，不管疑犯 2 采用哪种策略，疑犯 2 的最佳应对都是“坦白”，所以“坦白”是疑犯 1 的占优策略；
- 参与者的占优策略  $S$ ，是指该策略对于其他参与者的所有策略组合都是最佳应对；
- 参与者的 **严格**占优策略  $S$ ，是指该策略对于其他参与者的所有策略组合都是严格最佳应对；
- 如果参与人有严格占优策略，则可预期他会采取该策略。
- 占优策略可以用来简化求解过程。

# 详解博弈求解过程

如果两个人都有严格占优策略，则可以预计他们均会采取严格占优策略；

如果只有一个人有严格占优策略，则这个人会采取严格占优策略，而另一方会采取此策略的最佳应对（一定会有！）

如果两个人都没有严格占优策略呢？  
(从哪开始搜索？)

## 办法

- 混合策略；
- 纳什均衡。

# 三客户博弈：没有占优策略

## 三客户博弈问题描述

- 假设有两家公司，都希望和 A、B、C 三个大客户之一洽谈生意。每家公司都有三种可能的策略：是否找客户 A、B 或 C。它们决策的考量如下：
  - 若两家公司都找同一个客户，则该客户会给每个公司一半的业务。
  - 公司 1 规模太小，以至于不能靠自身找到客户源。所以，只要它和公司 2 分别寻找不同的客户洽谈生意，则公司 1 获得的收益将会是 0（生意做不成）。
  - 假设公司 2 单独寻找客户 B 或 C 洽谈生意，则会得到客户 B 或 C 的全部业务。但是 A 是一个大客户。寻找客户 A 洽谈生意时，必须和公司 1 合作才能接下业务。
  - 因为 A 是一个大客户，和它做生意的收益是 8（假设两家公司合作，则每家公司会得到收益 4）。但是，和 B 或 C 做生意的收益价值是 2（合作的话，每个公司收益是 1）。

## 三客户博弈：描述与求解

		公司2		
		A	B	C
公司1	A	4, 4	0, 2	0, 2
	B	0, 0	1, 1	0, 2
	C	0, 0	0, 2	1, 1

		公司2		
		A	B	C
公司1	A	😊4, 4😊	0, 2	0, 2
	B	0, 0	😊1, 1😊	0, 2😊
	C	0, 0	0, 2😊	😊1, 1

### 解释说明

- 两家公司都没有严格占优策略
- 如何讨论博弈的走向（结果）？
- 若博弈可以重复（假设了解了对方行动之后，可以撤回决定，重做决定），结果不确定！

# 纳什均衡

## 纳什均衡认知

- 假定参与者  $P_1$  选择策略  $S$ , 参与者  $P_2$  选择策略  $T$ 。若  $S$  是  $T$  的最佳应对, 且  $T$  也是  $S$  的最佳应对, 则称策略组  $(S, T)$  是一个纳什均衡。

## 纳什均衡的理解

- 在均衡状态, 任何参与人都没有动机 (理性的理由) 去换一种策略;
- 纳什均衡可以被看成是一种信念上的均衡互;
- 因为是最佳应对, 谁也不可能通过单方面改变策略而得到额外好处, 尽管如果两人都改变可能都会更好 (相比都不改变而言)。

## 求解博弈问题：纳什均衡

如果两个人都有严格占优策略，则可以预计他们均会采取严格占优策略；

如果只有一个人有严格占优策略，则这个人会采取严格占优策略，而另一方会采取此策略的最佳应对（一定会有！）

如果两个人都没有严格占优策略，则寻找纳什均衡。若存在一个纳什均衡，该均衡对应合理结果；若存在多个纳什均衡均衡？

若不存在纳什均衡？

# 零和博弈

## 硬币配对游戏：零和博弈的例子

- 两个参与者甲乙各持一枚硬币，同时选择自己硬币的朝向/正反面；
- 若两人选择的朝向相同，则己获胜，赢得甲的硬币；反之，甲赢得己的硬币。

		参与人乙	
		正面H	反面T
参与人甲	正面H	-1, +1	+1, -1
	反面T	+1, -1	-1, +1

不存在一组互为最佳应对（纯策略意义下的纳什均衡）。

# 引入混合策略

## 思考

- 若这样的博弈重复进行若干次，你会如何考虑自己的策略？
- 预测对方采用不同策略的概率，据此确定自己的策略
- 不要让对方了解自己采用不同策略的概率

## 混合策略

- 此时，“策略”可以看成是在两种固定策略（纯策略）之间选择的概率。

## 混合策略均衡：互为最佳应对

- 在各自概率策略的选择下，双方的收益期望互为最大（任何单方面改变不会增加收益）
- 纳什的奠基性贡献：证明了具有有限参与者和有限纯策略集的博弈一定存在纳什均衡（包括混合策略均衡），纳什均衡定理
- 一般来说，找到混合策略的纳什均衡是很困难的，但在某些特定条件下可能有系统的方法。

# 混合策略均衡

## 石头、剪刀、布

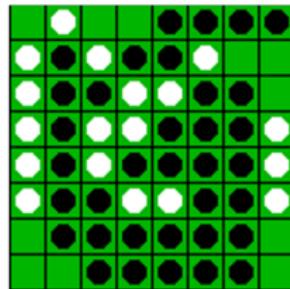
- 当你知道对手的策略是  $(p_1, p_2, p_3)$  之后，你一定 **不会输**！
- 你的最佳应对是什么？
- 双方的策略是互为 **最佳应对/纳什均衡**吗？
- 你是理性的人，当你不知道对手的策略（习惯）时，该如何选择策略？**最佳应对  $(1/3, 1/3, 1/3)$** ，不要让对方知道你的概率（习惯），否则你的对手稍微聪明一点，你就输了。

# 动态博弈

## 动态博弈：参与者依次行动

- 与静态博弈的区别在于各个参与者的行动不是同时的
- 静态博弈时，我们把策略和行动等同看待，因为参与者“同时”进行决策，没有观察到其他参与人采取了什么行动，所以实施行动时没有针对性。**同步**
- 动态博弈中，后行动的参与人采取行动之前已观察到了先行动参与人的行动结果，所以可以根据观察到的结果采取有针对性的行动。**异步**
- 行动是“做什么”；策略是“在什么情况下做什么”。

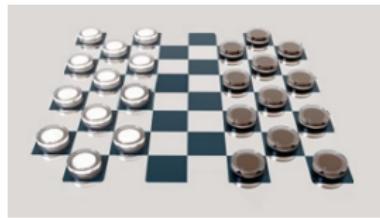
# 对抗性智力游戏



## Othello/翻转旗: Murakami vs. Logistello

- 1997: The Logistello software crushed Murakami by 6 games to 0
- 动态博弈，两个参与者依次执行行动；
- 零和博弈；

# 西洋跳棋



Mr. Tinsley VS. Chinook

- Name: Marion Tinsley
- Profession: 数学老师
- Hobby: 西洋跳棋
- Record: 45 年仅仅输掉 7 次棋
- 1992 年 8 月, Mr. Tinsley 对上 Chinook, 4 胜 2 负 33 平, Tinsley 40 多年遇到的第 6、7 负, 保持了 40 多年的世界冠军纪, 对手对上他, 祈求的是“平局”, 胜利是想都不敢想的, 1995 年去世。

# 国际象棋



Kasparov

5'10"  
176 lbs  
34 years  
50 billion neurons  
2 pos/sec  
Extensive  
Electrical/chemical  
Enormous

Height  
Weight  
Age  
Computers  
Speed  
Knowledge  
Power Source  
Ego

6' 5"  
2,400 lbs  
4 years  
32 RISC processors  
+ 256 VLSI chess engines  
200,000,000 pos/sec  
Primitive  
Electrical  
None

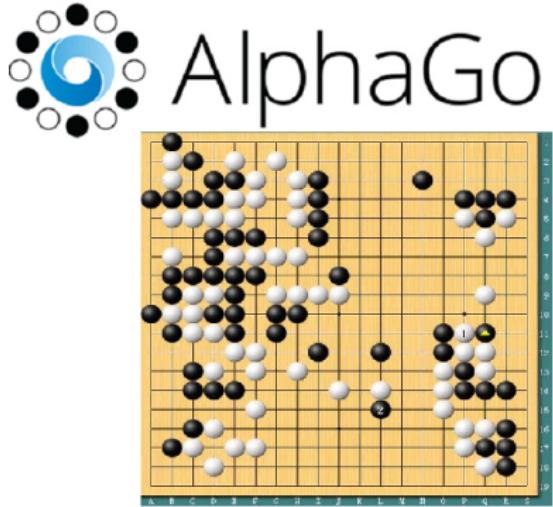
Deep Blue

1997: Deep Blue 3胜1负2平

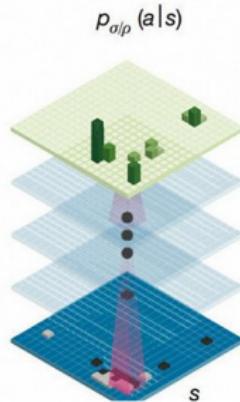
卡斯帕罗夫 VS. 深蓝

- 1997: Deep Blue 3 胜 1 负 2 平

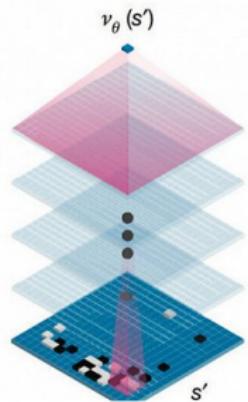
# Alpha Go



Policy network

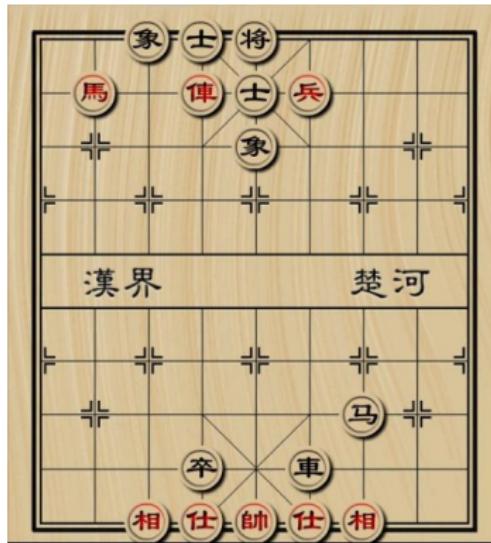


Value network



异步行动：两个棋手依次落子。动态博弈的例子

# 例子：中国象棋



## 中国象棋：博弈描述

- 参与者/agent:  $\{Red, Black\} \triangleq \{P_1, P_2\}$
- 状态  $s$ : 棋局中任何一个“局面”(棋子分布情况)
- 策略/行动: 参与者  $P_i$  可以执行的让棋子进行的“合法”移动。
- 终态: 某个参与者胜利或平局;
- 效用函数: 胜利的参与者获得奖励  $r = +1$ , 失败者获得  $-r = -1$  的奖励; 平局都获得 0 的奖励; 零和。

## 说明

- 行动、策略、混合策略: 策略被执行就变成行动, 混合策略是各种行动的执行概率。
- 零和博弈
- 动态博弈

# 动态博弈与马尔科夫决策过程

MDP  $\Rightarrow$  动态博弈

- **完美信息下的动态博弈**: 每个参与者知道一切可以知道的公共信息;
- 将对手视为 MDP 建模中无法观察和了解的未知因素;
- 完美信息下的动态博弈就是一个参与者和所有未知因素综合而成的虚拟“虚拟参与者”之间的二人博弈，也就是 MDP。

下棋并非 MDP，而是强化学习

- 不知道转移概率
- 不知道每步/每次行动的奖励
- 需要逐步去了解和学习

# 博弈问题描述为搜索问题

## 博弈即搜索

- 状态：参与者的行动/策略及其各自对应的收益；
- 后继函数：参与者可能的各种新策略；
- 初态：随机或某个给定初态；
- 终态：均衡态（不管任何一个参与者如何调整策略/行动，各自的收益不再增加）；
- 路径耗散：单步路径耗散为常数  $c$

## 解释说明

- 参与者、参与者的行动及其各自收益，是描述状态的向量的分量；
- 博弈问题描述为搜索问题时，解序列并不是追求的目标；“均衡的”终态才是关注的目标。
- 路径耗散在博弈中不大关注。

# 博弈与一般搜索问题的差异

效用函数可能只在终态是有意义

- 博弈中，中间状态的效用可能不重要，比如，下棋尽管丢了很多，但是最终早一步取得胜利

博弈有两个/多个相互“拆台”或“合作”的参与者交替“寻找”路径

- 而经典搜索问题只有一个参与者
- 博弈可理解为从初态开始的寻路过程中，两个参与者交替给路径添加一条边，但是二者的目标状态集合不一样。
- 参与者不关心路径的长度，不过分“纠结”眼前的利益大小，到达自己的目标状态就是“胜利”。
- 一个参与者不知道其他参与者“将来”会如何行动

需要新的方法来描述博弈搜索过程：博弈树

# 博弈评估与策略评估

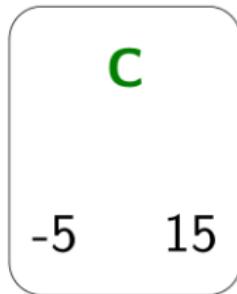
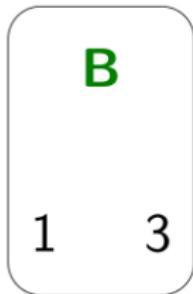
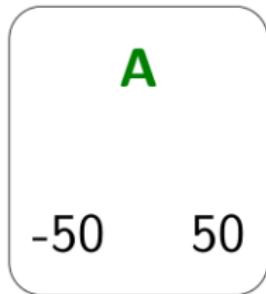
## 来自 MDP

- 策略是任意状态到行动的映射机制。执行策略会产生大量的随机路径，所有随机路径的期望收益就是 MDP 的策略收益。
- 策略评估是依据 MDP 策略收益来评价策略的优劣。

## 博弈评估

- 具有对抗性。
- 只能控制部分状态下的策略；因此博弈（的策略）评估包含对对手策略的“假设”。
- 博弈评估中，路径中“分支”不会额外付出“代价”作为参与者的“收益”，仅仅描述终态的收益在博弈树非叶结点上的聚集。

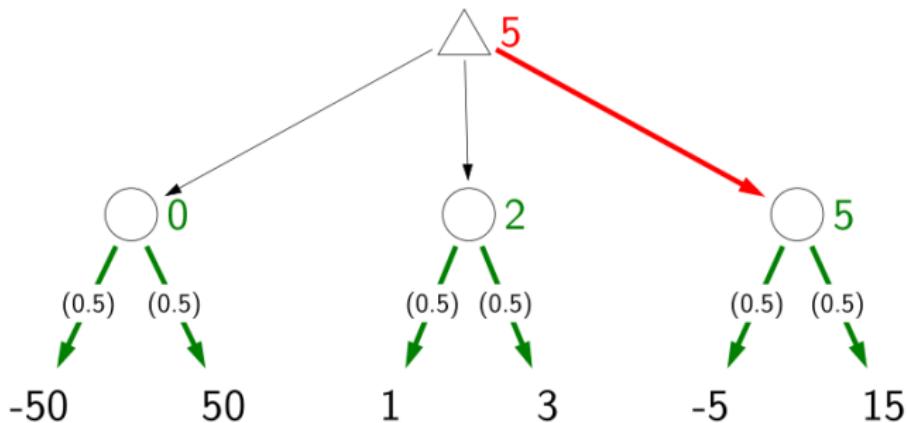
## 例子：对抗和非对抗游戏



从哪个盒子中抽取？

- 假设三个黑盒 A、B、C 中的数（钱）都是公开的，但是看不见；
- 甲一个人从中抽取一个数，获得相应的数额的钱，他该如何选择？
- 若甲乙两个人依次从中抽取，数字大则赢，各自该如何决策/行动？

## 一个人玩的情况

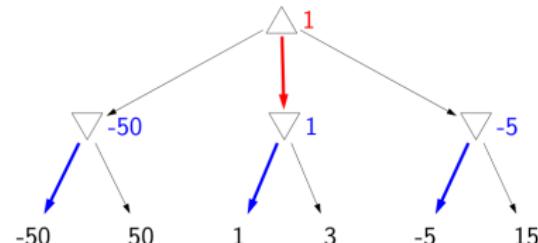
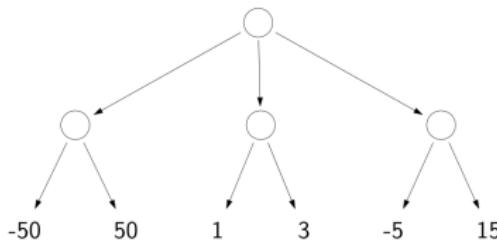


## 期望最大化

- 计算每个黑盒的期望收益
- 然后比较各个黑盒的期望收益，取最大者
- 边上权值是选择该边的概率
- 叶子节点是各种结果最终的收益。

## 博弈树

# 描述博弈：从搜索树到博弈树



## 情况发生改变：两个人（甲乙）玩同一个游戏

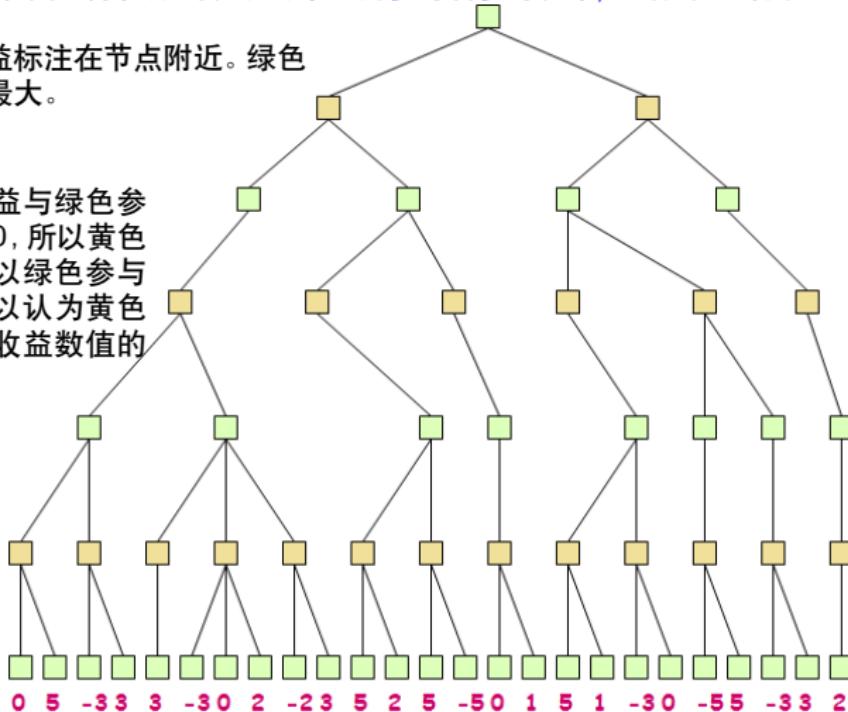
- 每个节点表示参与者的一个决策状态；
- 从跟到叶的一条路径表示一个解，一条搜索路径。
- 对抗性智力游戏中：路径是两个“相互拆台”的参与者共同“生产”出来，不同于华容道等问题，一个参与者“生产”出来。
- 故分别用不同的节点符号（正立的三角形和倒立的三角形）区分两个参与者，如右图。
- 期望最大化  $\Rightarrow$  每个参与者都希望在多个可能性中选择对自己最好的/最有利的。零和博弈：对甲最好，则是对己最差。
- 右上图中，节点的“值”表示参与者甲的收益。甲能控制的是正立三角形（尖角朝上），会选择各分支中的最大值（正立三角形的决策）；乙能控制的是倒立三角形（尖角朝下），乙的收益是负得越多越好（负是指甲付出代价，图中顶点的权值的参照物都是参与人甲），所以甲就会思考“已是一个理性的聪明人，他会做出对他自己最有利的决策，乙会在他所有的决策中选择负得最多的分支”，即倒立三角形的决策是“取各个分支中最小值”

# 博弈树

绿色和黄色方框分别表示两个不同参与者参与决策/控制的“局面”

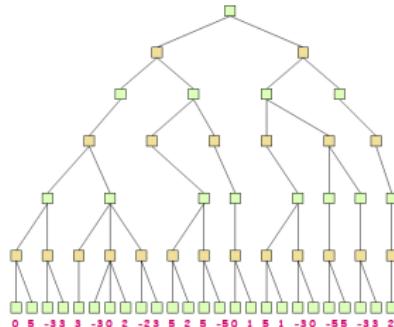
绿色参与者的收益标注在节点附近。绿色  
参与者最求收益最大。

黄色参与者的收益与绿色参  
与者的收益和为0, 所以黄色  
参与者的收益在以绿色参与  
者为参照时, 可以认为黄色  
参与者追求图中收益数值的  
最小。



叶子节点的权值表示从根开始到叶的搜索路径, 参与者获得的收益

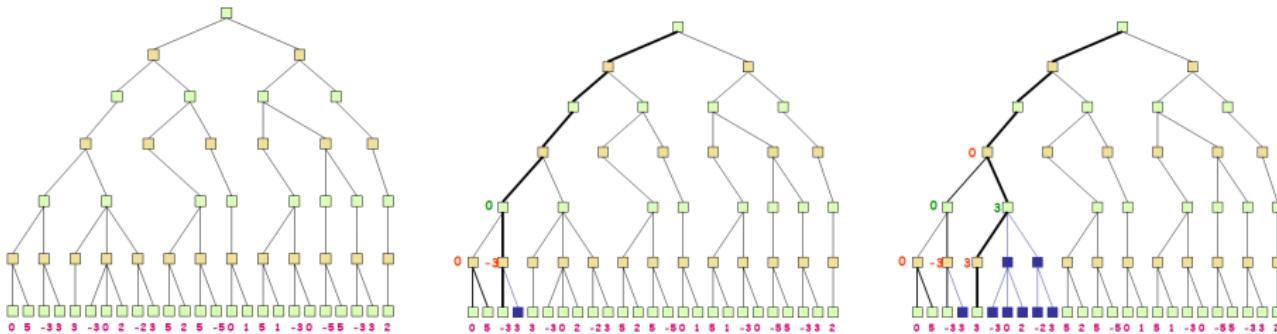
# 博弈树的构造



# 博弈树的构造

- 绿色参与者以他当前面对的棋局为根节点，考虑对手（黄色参与者）的行动；构建博弈树/搜索树；
- 黄色参与者有两种行动，绿色参与者都进行了“进一步”的考虑，成为博弈树的第二层节点；
- 黄色参与者的任何行动，绿色参与者有可以有各自的不同应对；该过程可以循环迭代下去，构成博弈树。

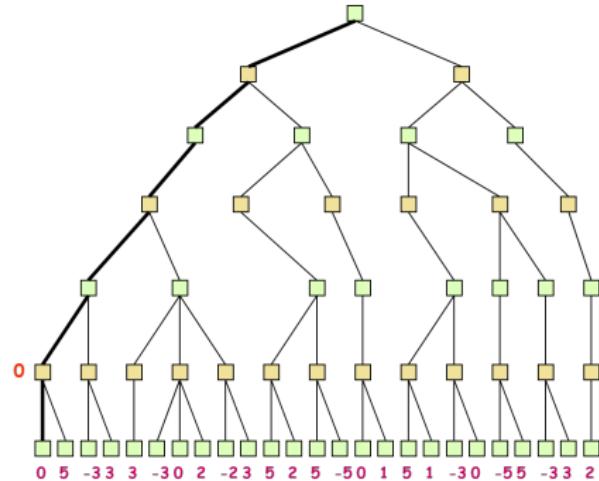
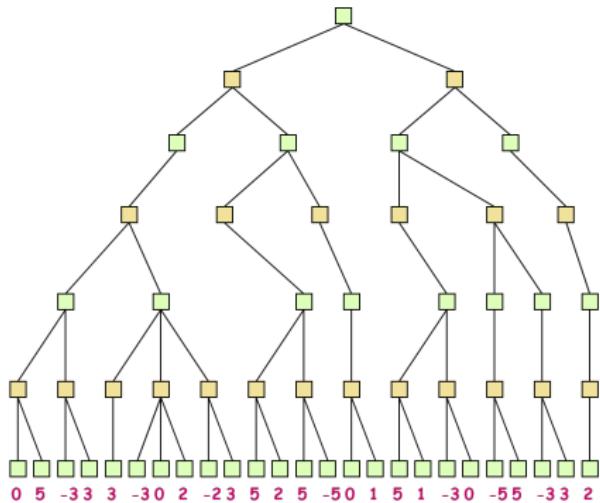
# 博弈树和博弈推理过程



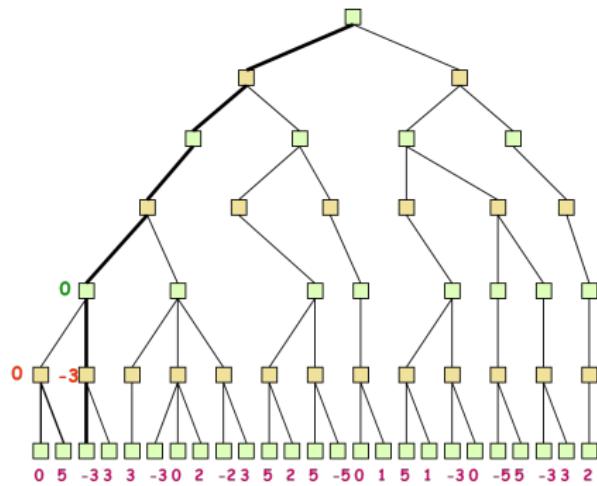
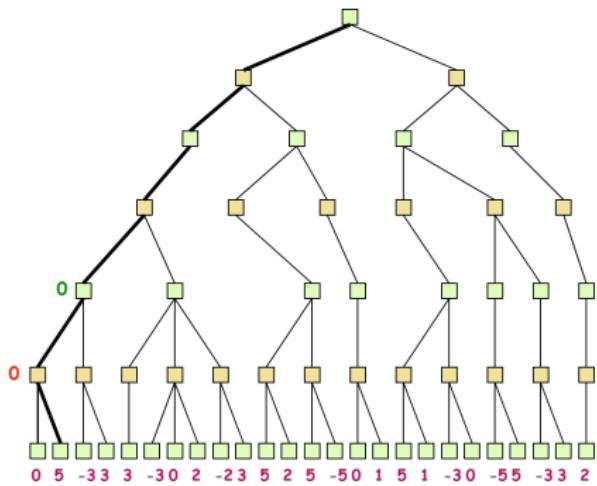
## 基于博弈树的推理

- 绿色参与者在有限的时间内，假设能思考到前进  $h$  层，可能没有达到博弈结束；
- 棋手水平的高低，可用思考的“深度”  $h$  来衡量；
- 经典的搜索算法：估计将来  $h$  层节点的“收益”，向树根方向执行“最小最大值”算法

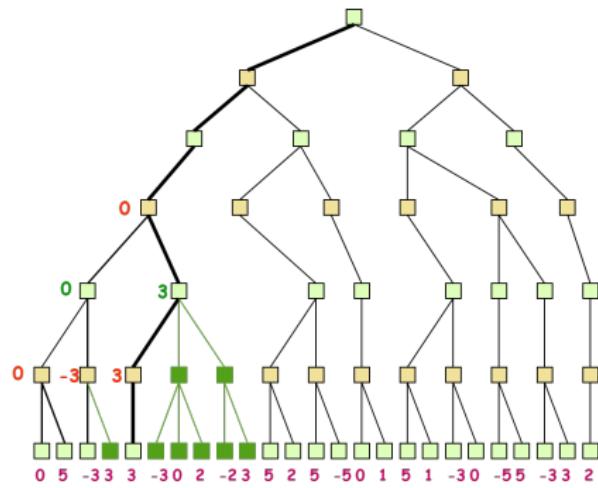
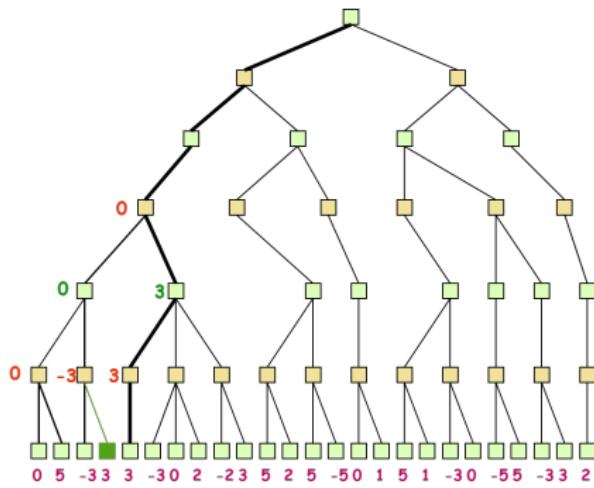
# 最小最大值算法：例子



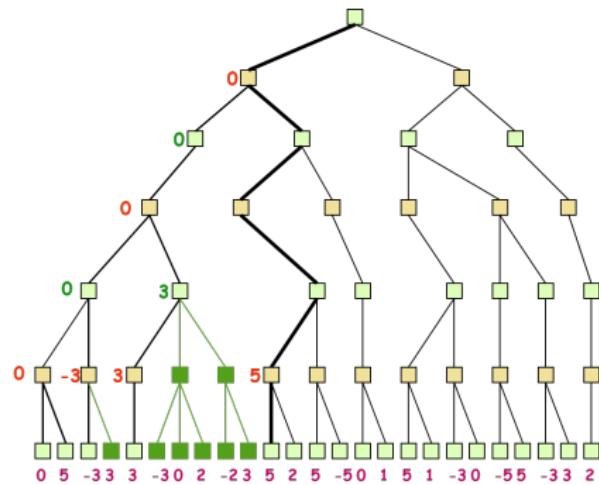
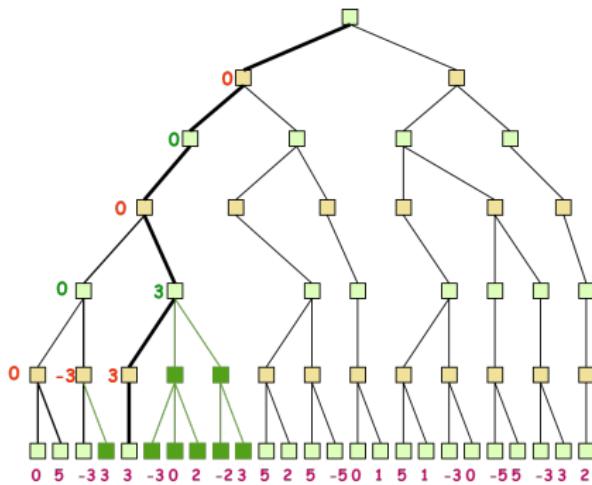
# 最小最大值算法：例子



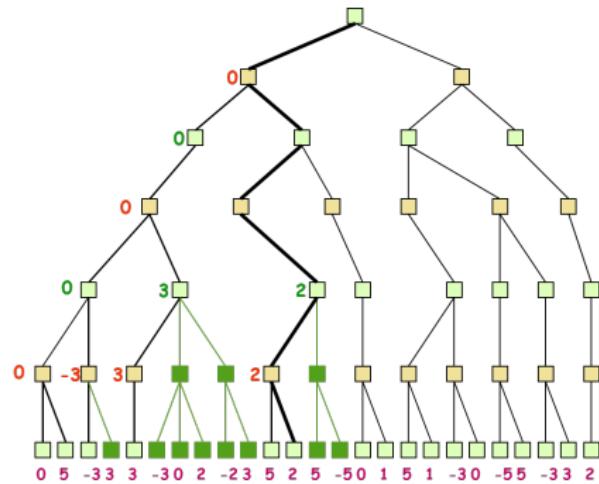
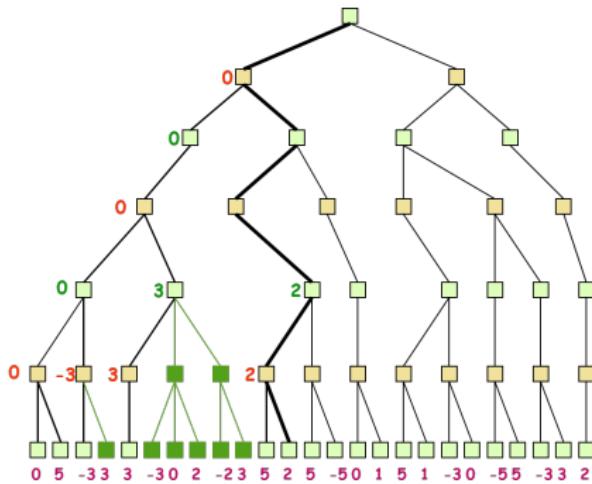
# 最小最大值算法：例子



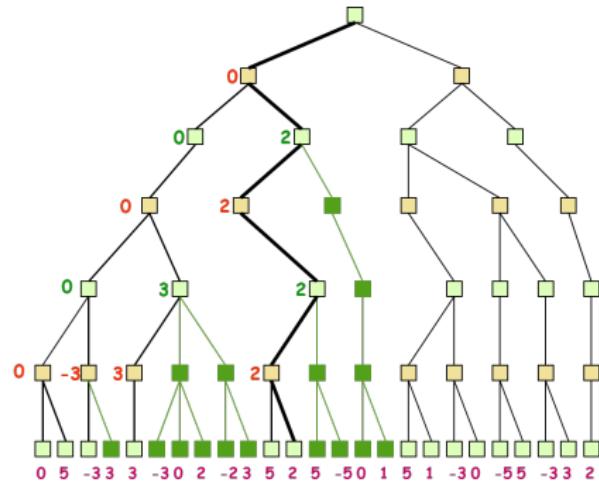
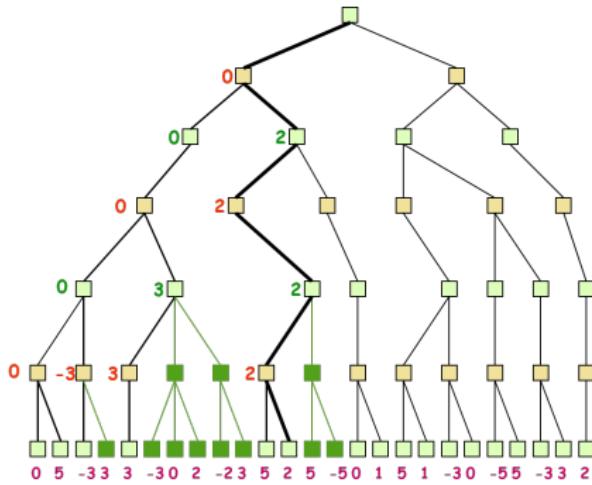
# 最小最大值算法：例子



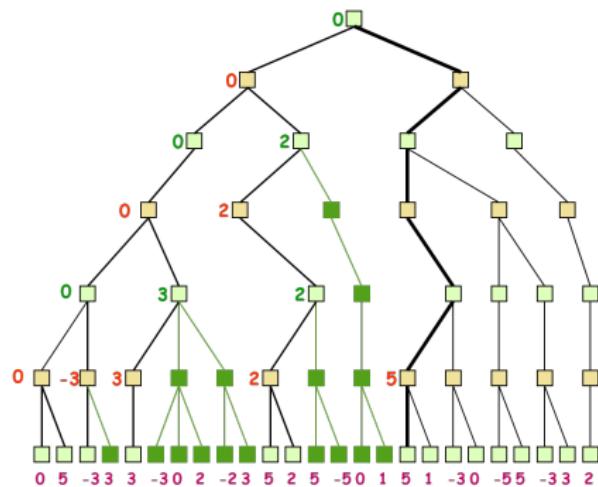
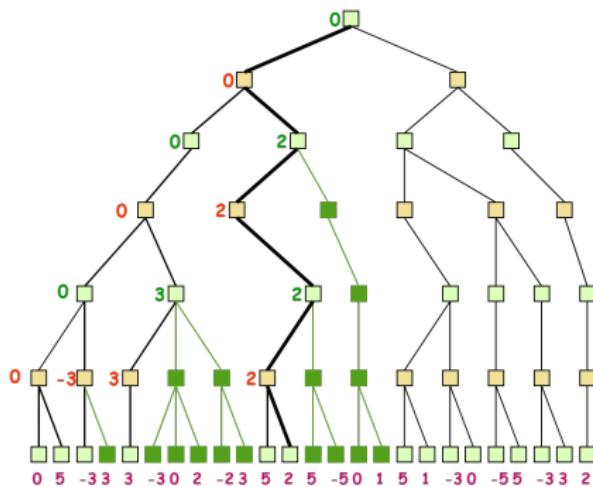
# 最小最大值算法：例子



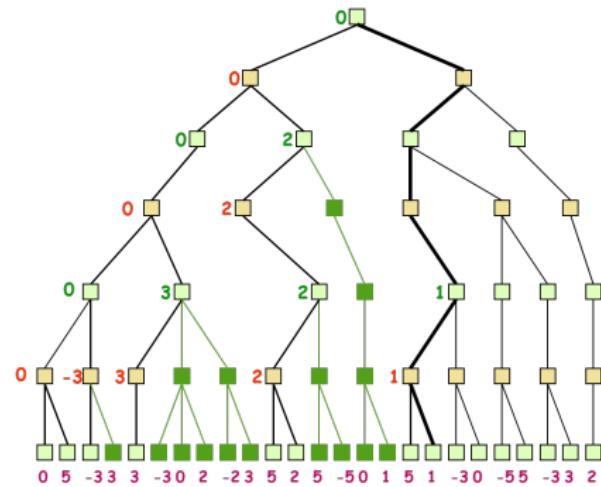
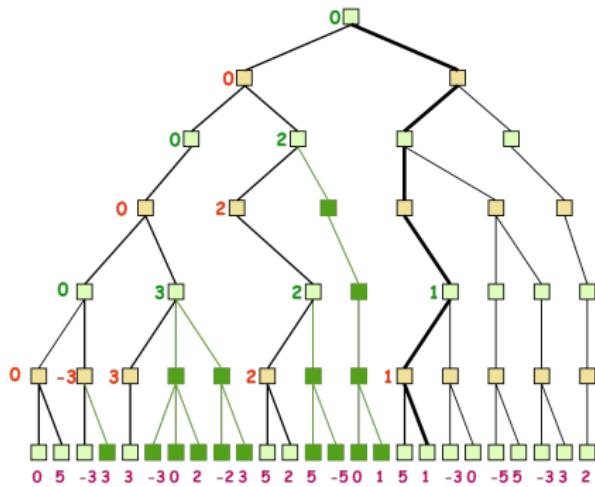
# 最小最大值算法：例子



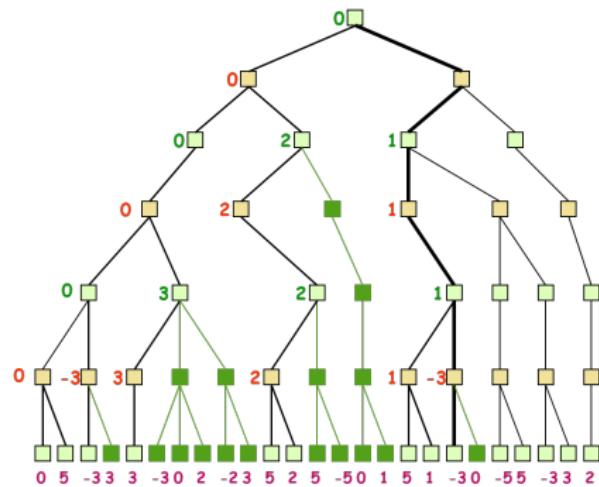
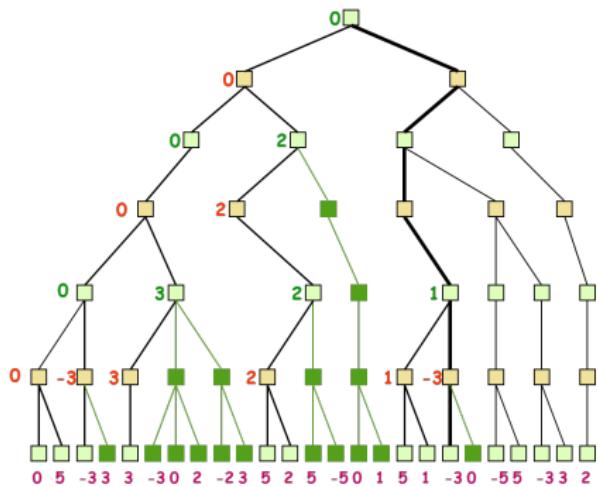
# 最小最大值算法：例子



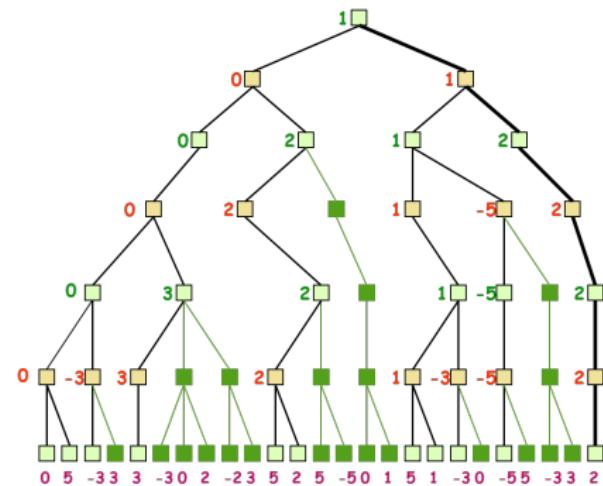
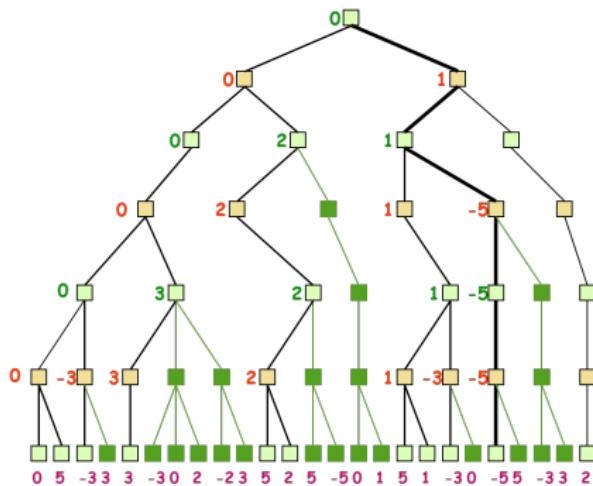
# 最小最大值算法：例子



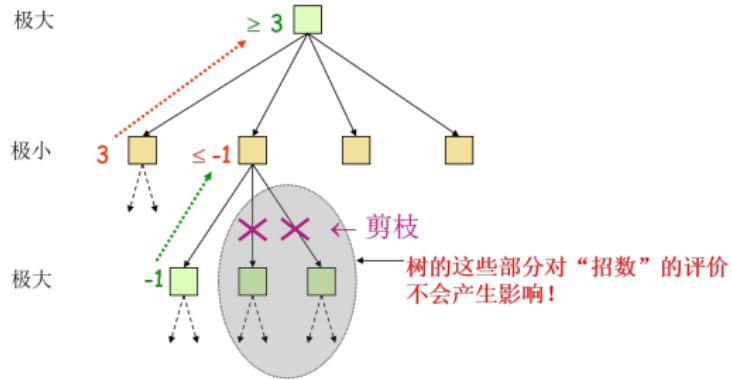
## 最小最大值算法：例子



# 最小最大值算法：例子



# $\alpha - \beta$ 剪枝



## $\alpha - \beta$ 剪枝原理：减少需要检验的分枝数量

- $\alpha$  值：任何一个需极大化的节点，以其为根的子树中，目前找到的极大值
- $\beta$  值：任何一个需极“小”化的节点，以其为根的子树中，目前找到的极小值
- 存在问题：如何确定每层节点的检验次序？对搜索效率至关重要。

# 围棋与 AlphaGo/AlphaZero

## 基本原理

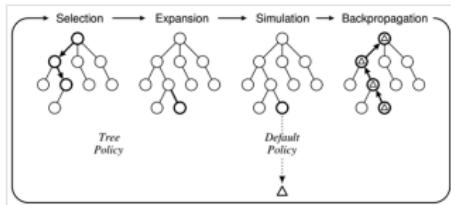
- 算法的主体是蒙特卡洛树搜索，但是加入两个深度网络来改进：
  - policy network：卷积神经网络，嵌入强化学习思想，收集已有棋谱，估计对手落子的区域，即随机路径产生的过程中，对手“机器人”不是均匀随机的；对当前棋局的每个对手落子给出一个条件概率分布；
  - value network：卷积神经网络，给出自己每个落子收益；（部分采用随机落子）
- 强化学习
- 是否需要先验知识（AlphaGo or AlphaZero）

# 蒙特卡洛树搜索简介

问题：指数增加的树如何构造、存储和评价分枝？

- 博弈树/搜索树随着深度增加，节点数目指数膨胀，即使是最小最大值方法，也无法解决该问题；
- 随机模拟来估计每个分枝的“收益” / “转移概率”
- 给定每一步的时间限制（例如每步最多 5min），假设两个“随机棋手”从当前棋局开始下棋，二者都是随机落子；二人多次以给定的初始棋局开始，重复随机博弈，统计实际路径中每个节点的收益。随机路径的数目由时间限制来约束。
- 每次随机博弈的胜利方收益为 1，多次（大量）重复随机博弈，得到了博弈树的部分随机路径（子树），对该子树执行最小最大值算法或者估计经过节点的收益估计值和转移概率，并以之为实际值的一种估计。
- 类比 MDP 和强化学习。

# 蒙特卡洛树搜索思想



**Algorithm 1** General MCTS approach.

```
function MCTSSEARCH( $s_0$ )
    create root node  $v_0$  with state  $s_0$ 
    while within computational budget do
         $v_l \leftarrow \text{TREEPOLICY}(v_0)$ 
         $\Delta \leftarrow \text{DEFAULTPOLICY}(s(v_l))$ 
        BACKUP( $v_l, \Delta$ )
    return  $a(\text{BESTCHILD}(v_0))$ 
```

## 算法思想

- 选择：使用树策略从根节点开始采取行动，直到选择到一个最紧急的可扩展节点（叶子节点或是含有部分未探索的行动的中间节点）
- 扩展：在该节点上通过树策略选择一个可行的动作，并将其指向的节点添加到搜索树中。
- 模拟：从新节点开始采用默认策略进行模拟，直到达到一个终端状态并得出奖励。
- 回溯：模拟得出的奖励回传到从新节点到根节点的每一个经由节点上，回传的奖励将会影响树策略。

# 蒙特卡洛树搜索算法的具体实现

## Algorithm 2 The UCT algorithm.

```
function UCTSEARCH( $s_0$ )
    create root node  $v_0$  with state  $s_0$ 
    while within computational budget do
         $v_l \leftarrow \text{TREEPOLICY}(v_0)$ 
         $\Delta \leftarrow \text{DEFAULTPOLICY}(s(v_l))$ 
        BACKUP( $v_l, \Delta$ )
    return  $a(\text{BESTCHILD}(v_0, 0))$ 
```



```
function TREEPOLICY( $v$ )
    while  $v$  is nonterminal do
        if  $v$  not fully expanded then
            return EXPAND( $v$ )
        else
             $v \leftarrow \text{BESTCHILD}(v, Cp)$ 
    return  $v$ 
```

```
function EXPAND( $v$ )
    choose  $a \in$  untried actions from  $A(s(v))$ 
    add a new child  $v'$  to  $v$ 
        with  $s(v') = f(s(v), a)$ 
        and  $a(v') = a$ 
    return  $v'$ 
```

```
function BESTCHILD( $v, c$ )
    return  $\arg \max_{v' \in \text{children of } v} \frac{Q(v')}{N(v')} + c \sqrt{\frac{2 \ln N(v)}{N(v')}}$ 
```

```
function DEFAULTPOLICY( $s$ )
    while  $s$  is non-terminal do
        choose  $a \in A(s)$  uniformly at random
         $s \leftarrow f(s, a)$ 
    return reward for state  $s$ 
```

```
function BACKUP( $v, \Delta$ )
    while  $v$  is not null do
         $N(v) \leftarrow N(v) + 1$ 
         $Q(v) \leftarrow Q(v) + \Delta(v, p)$ 
         $v \leftarrow \text{parent of } v$ 
```

# 上置信界/Upper Confidence Bounds, UCB

## 上置信界和树的上置信界

- $UCB = \bar{X}_j + \sqrt{\frac{2 \ln n}{n_j}}$ , 其中  $\bar{X}_j$  是节点  $j$  获得的平均奖励,  $n_j$  表示节点  $j$  被探索的次数,  $n$  为探索的总次数
- $UCT = \bar{X}_j + 2C_p \sqrt{\frac{2 \ln n}{n_j}}$

## 作用

- 平衡搜索的强度和广度
- 平均奖励越高, 即  $\bar{X}_j$  越大, 越倾向于对节点  $j$  进行搜索,  $n_j$  也越大, 第二项也就越接近 0;
- 平均奖励小, 第二项  $n_j$  也会小, 但是总搜索次数  $n$  在增加, 第二项的作用越来越大, 值有可能成为 UCB 和 UCT 的主要部分, 推动对节点  $j$  的搜索。

# 蒙特卡洛树搜索中的树策略

```
function TREEPOLICY( $v$ )
    while  $v$  is nonterminal do
        if  $v$  not fully expanded then
            return EXPAND( $v$ )
        else
             $v \leftarrow \text{BESTCHILD}(v, Cp)$ 
    return  $v$ 
```

```
function EXPAND( $v$ )
    choose  $a \in$  untried actions from  $A(s(v))$ 
    add a new child  $v'$  to  $v$ 
        with  $s(v') = f(s(v), a)$ 
        and  $a(v') = a$ 
    return  $v'$ 
```

```
function BESTCHILD( $v, c$ )
    return  $\arg \max_{v' \in \text{children of } v} \frac{Q(v')}{N(v')} + c \sqrt{\frac{2 \ln N(v)}{N(v')}}$ 
```

## 树策略

- 只要当前节点不是游戏终端状态/叶节点，如果当前节点的所有动作未被完全扩展，则扩展本节点的其他动作并返回下一个新节点，否则选择一个最佳子节点。若
- 到达了终端节点，则返回终端节点。
- 扩展节点  $v$ : 选择节点  $v$  没有被访问的一个动作，并把该动作指向的节点  $v'$  添加到树中，同时记录下指向节点  $v'$  的动作，返回节点  $v'$ 。
- 选择最佳子节点  $v'$ : 应用 UCT，在树策略中寻找下一个最佳子节点。

# 蒙特卡洛树搜索的默认策略与回溯

```
function UCTSEARCH( $s_0$ )
    create root node  $v_0$  with state  $s_0$ 
    while within computational budget do
         $v_l \leftarrow \text{TREEPOLICY}(v_0)$ 
         $\Delta \leftarrow \text{DEFAULTPOLICY}(s(v_l))$ 
        BACKUP( $v_l, \Delta$ )
    return  $a(\text{BESTCHILD}(v_0, 0))$ 
```

```
function DEFAULTPOLICY( $s$ )
    while  $s$  is non-terminal do
        choose  $a \in A(s)$  uniformly at random
         $s \leftarrow f(s, a)$ 
    return reward for state  $s$ 

function BACKUP( $v, \Delta$ )
    while  $v$  is not null do
         $N(v) \leftarrow N(v) + 1$ 
         $Q(v) \leftarrow Q(v) + \Delta(v, p)$ 
         $v \leftarrow \text{parent of } v$ 
```

## 默认策略

- 用于模拟时使用，随机选取当前节点的一个动作，直到模拟到最终的节点，并返回其终端状态对应的奖励。

## 奖励回溯

- 从当前扩展节点到根节点的每一个节点的访问次数 +1，奖励增加。

# 蒙特卡洛树搜索评价

## 优点：

- 不需要特定领域的启发式算法
- 但是如果采用了启发式算法会极大地降低模拟次数、减少分支，就是说可以结合启发式方法，提高效率
- 结合启发式方法提高效率，带来的问题是会减少探索的空间，容易陷入局部最优

## 蒙特卡洛树搜索的改进和变型

### Bandit Algorithm for Smooth Trees (BAST)

- 使用了奖励是平滑的的假设，可以识别出是否是最优分支，并且会不断扩展最优分支，并假设如果时间无限则只有最优分支会被不断扩展上去，而普通的 UCT 则会扩展全部的分支

### Temporal Difference Learning (TDL)

- TDL 和 MCTS 都是根据状态对应的价值或者状态行动对来选取行动的。
- 差异在于 TDL 是提前运行好全部的过程并保存下来，而 MCTS 是一边运行一边计算

### Single-Player MCTS (SP-MCTS)

- $UCB = \bar{X}_j + \sqrt{\frac{2 \ln n}{n_j}} + \sqrt{\sigma^2 + \frac{D}{n_j}}$
- 增加第三项，代表单个玩家可能带来的偏差

# 蒙特卡洛树搜索的改进和变型

## Multi-player MCTS

- 在 Minimax 方法中，每名玩家都希望自己奖励最大化，敌人奖励最小化。而多玩家对应于每个节点都存储一个奖励向量，在计算时整体考虑本联盟的奖励，从字面上理解为使得我方  $n$  个队友的奖励和最大，这有可能对于本玩家并不是最大奖励。其将会考虑减少不利于友方玩家的影响，就好像都是自己的一部分一样。对于多个合作的对手，可以看成单一有效的对手。对于多玩家的不同联盟情况有三种决策方法：
  - Paranoid UCT：认为所有人都是敌对的
  - UCT with Alliances：有明确的联盟成员名单
  - Confident UCT：对每一个玩家进行搜索比较，确定是否可以形成联盟

## Multi-agent MCTS 和 Ensemble UCT

- 有些研究认为在默认策略的模拟阶段，采用并行化的多个智能体单独模拟并结合分析结果，产生的结果要优于只有一个智能体模拟的结果。

# 蒙特卡洛树搜索研究的几个方向点

## 结合领域知识

- 不管是否是通用方法，领域知识对特定问题总是能带来提高

## UCB 公式的改进

- 用于平衡搜索的强度和广度

## 树策略的改进

- 超参  $C_p$  选择，行动分组，保存模拟搜索数据，对每个行动预评分/利用领域知识

## 模拟结果的利用方法

- 用在同层次的其它节点，压缩存储模拟结果等

## 提升模拟能力

- 利用领域知识，利用历史，加快模拟

## 改进回溯

- 奖励加权，后期奖励更有效；非 +1 和 -1 的奖励，允许 -1 到 +1 之间的实数；奖励沿路径衰减机制等

## 并行化

- 由于 MCTS 的每次模拟都具有独立性，因此可以采用并行化来加快运行速度。