搜索求解2

主讲：郭春乐、刘夏雷

南开大学计算机学院

致谢：本课件主要内容来自浙江大学吴飞教授、

南开大学程明明教授

单选题 1分

设置

1.以下关于用搜索算法求解最短路径问题的说法中,不正确的是( )。

A 给定两个状态,可能不存在两个状态之间的路径;也可能存在两个状态之间的路径,但不存在最短路径(如考虑存在负值的回路情况)。

B 假设状态数量有限,当所有单步代价都相同且大于0时,深度优先的图搜索是最优的。

C 假设状态数量有限,当所有单步代价都相同且大于0时,广度优先的图搜索是最优的。

D 图搜索算法通常比树搜索算法的时间效率更高。

提交

设置

2.以下关于启发函数和评价函数的说法中正确的是(　)。

A 启发函数不会过高估计从当前节点到目标结点之间的实际代价。

B 取值恒为0的启发函数必然是可容的。

C 评价函数通常是对当前节点到目标节点距离的估计。

D 如果启发函数满足可容性,那么在树搜索A*算法中节点的评价函数值按照扩展顺序单调非减;启发函数满足一致性时图搜索A*算法也满足该性质。

提交

单选题 1分

设置

3.假如可以对围棋的规则做出如下修改,其中哪个修改方案不影响使用本章介绍的Minimax算法求解该问题?(　)

A 由双方轮流落子,改为黑方连落两子后白方落一子。

B 双方互相不知道对方落子的位置。

C 由两人对弈改为三人对弈。

D 终局时黑方所占的每目(即每个交叉点)计1分,且事先给定了白方在棋盘上每个位置取得一目所获取的分数,假设这些分数各不相同。双方都以取得最高得分为目标。

提交

主观题 10分

设置

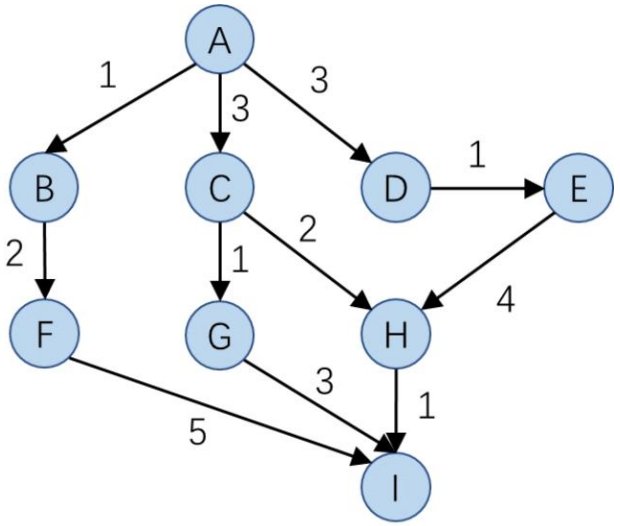


图1 状态转移图

1. 考虑图1中的问题，给定每个状态的启发函数如下表所示。若仍以状态A为初始状态、状态I为终止状态，请分别使用以下算法求解从A到I的路径，请画出算法找到第一条路径时的搜索树，并在搜索书中标出结点的扩展顺序，以及找到的路径。（若有多个节点拥有相同的扩展优先度，则优先扩展对应路径字典序较小的节点）。

(1) 基于树搜索的贪婪最佳优先搜索。

(2) 基于图搜索的A*算法。

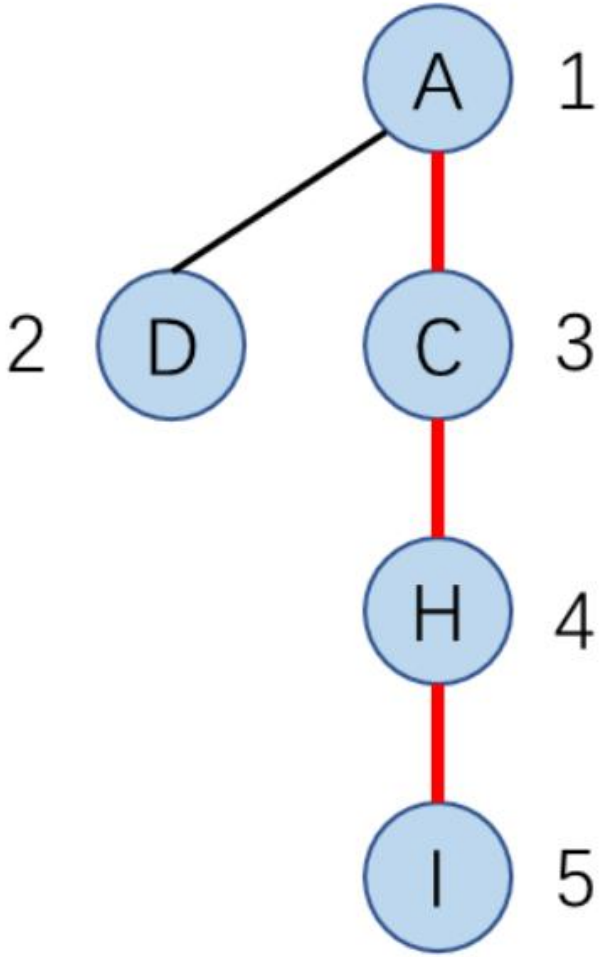
状态	A	B	C	D	E	F	G	H	I
启发函数	5	4	3	2	5	5	2	1	0

正常使用主观题需2.0以上版本雨课堂

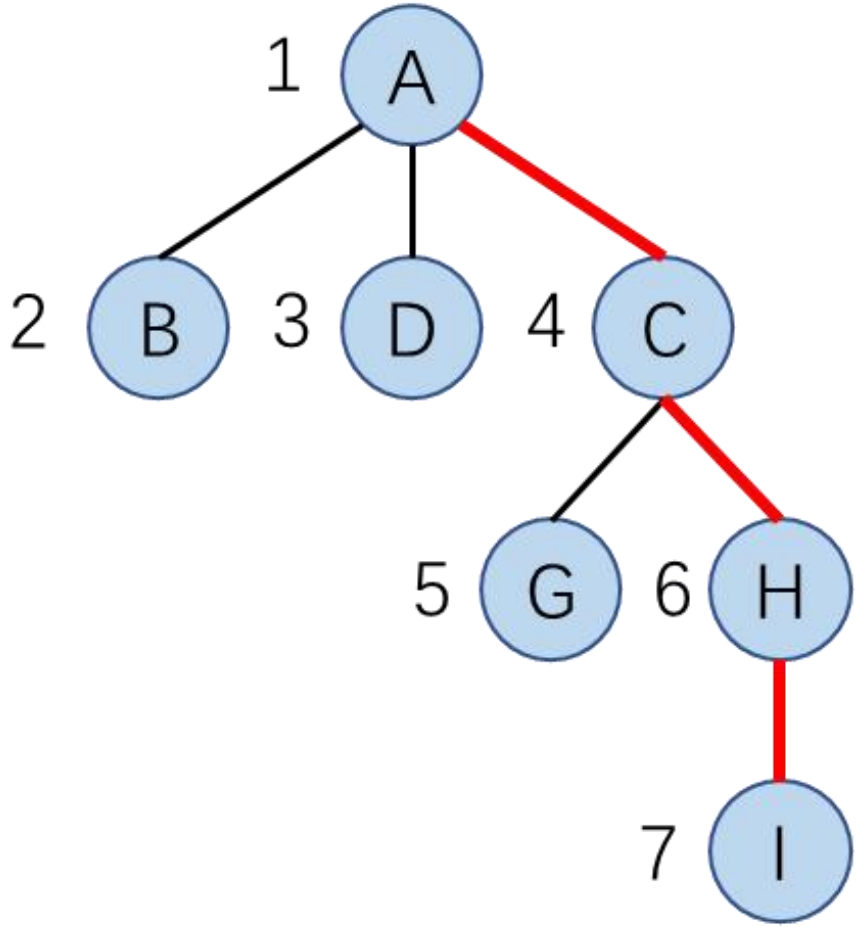
作答

习题答案

(1) 基于树搜索的贪婪最佳优先搜索。



(2) 基于图搜索的A*算法。



提纲

- 搜索算法基础
- 启发式搜索
- 对抗搜索
- 蒙特卡洛树搜索

对抗搜索

- 对抗搜索(Adversarial Search)也称为博弈搜索(Game Search)
- 在一个竞争的环境中，智能体(agents)之间通过竞争实现相反的利益，一方**最大化**这个利益，另外一方**最小化**这个利益。

对抗搜索：主要内容

- **最小最大搜索(Minimax Search)**
  - 最小最大搜索是在对抗搜索中最为基本的一种让玩家来计算最优策略的方法
- **Alpha-Beta剪枝搜索(Pruning Search)**
  - 一种对最小最大搜索进行改进的算法，即在搜索过程中可剪除无需搜索的分支节点，且不影响搜索结果。
- **蒙特卡洛树搜索(Monte-Carlo Tree Search)**
  - 通过采样而非穷举方法来实现搜索。

对抗搜索

- **本课程目前主要讨论在确定的、全局可观察的、竞争对手轮流行动、零和游戏(zero-sum)下的对抗搜索**
  - 所谓零和博弈是博弈论的一个概念，属非合作博弈。指参与博弈的各方，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。与“零和”对应，“双赢博弈”的基本理论就是“利己”不“损人”，通过谈判、合作达到皆大欢喜的结果。

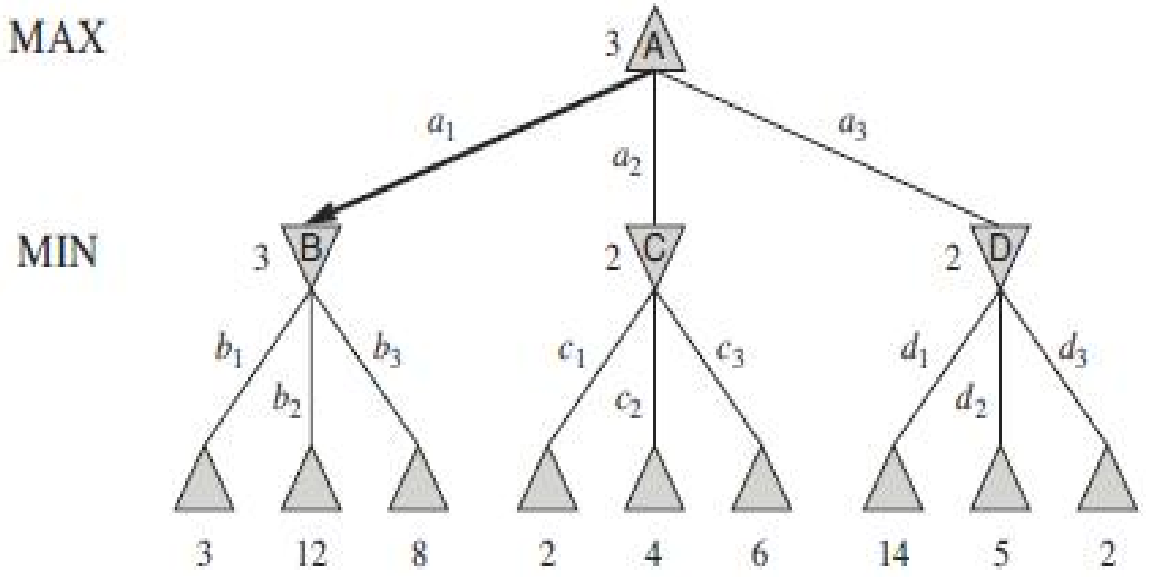
对抗搜索：Alpha-Beta 剪枝搜索

- 在极小化极大算法(minimax算法)中减少所搜索的搜索树节点数。该算法和极小化极大算法所得结论相同，但剪去了不影响最终结果的搜索分枝。

$$MINIMAX(root)$$
$$= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2))$$
$$= \max(3, \min(2,x,y), 2)$$
$$= \max(3,z,2) = 3$$

$$where\ z = \min(2,x,y) \leq 2$$

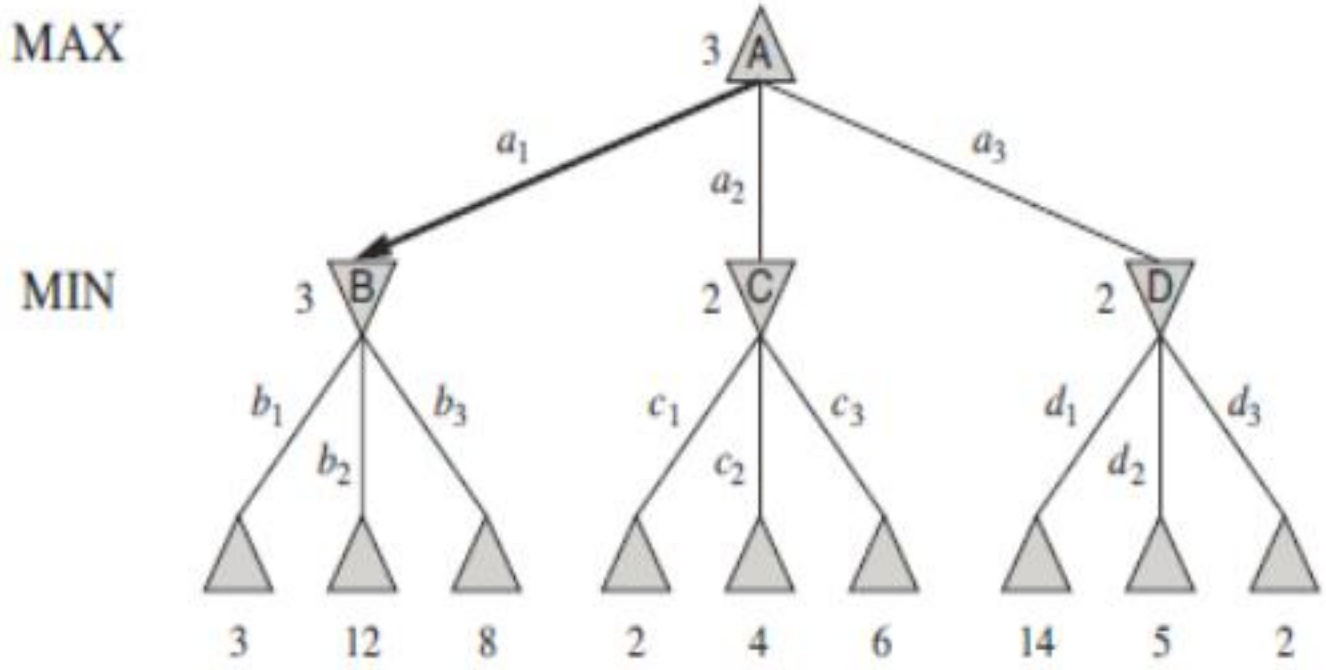
可以看出：根节点(即 MAX选手)的选择与x和y 两个值无关(因此，x和y可以被剪枝去除)



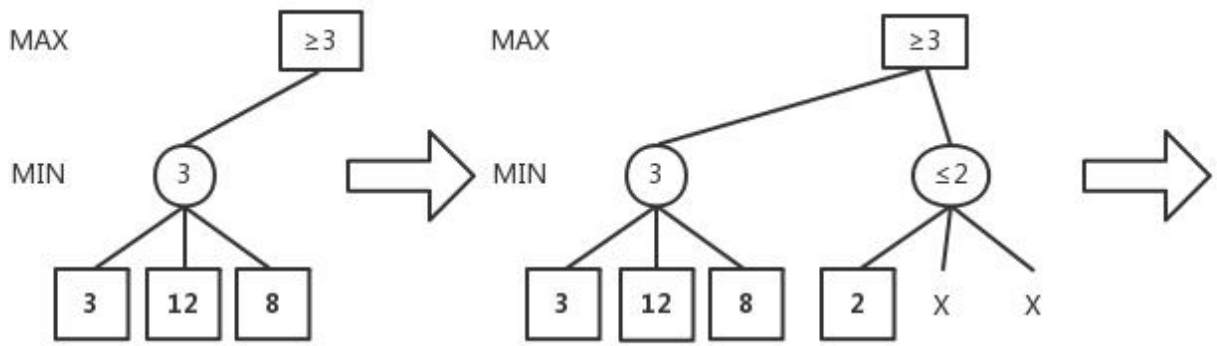
对抗搜索：Alpha-Beta 剪枝搜索

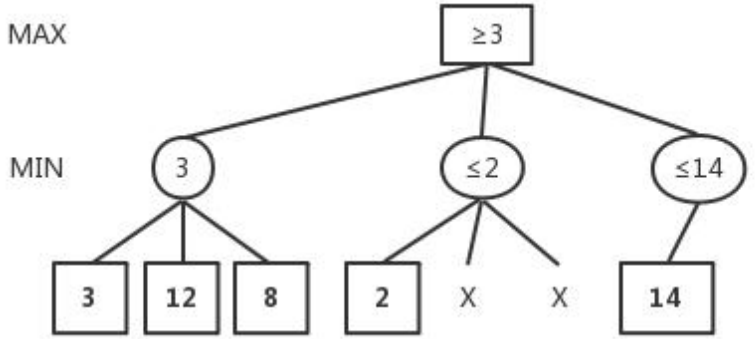
- 在极小化极大算法(minimax算法)中减少所搜索的搜索树节点数。该算法和极小化极大算法所得结论相同，但剪去了不影响最终结果的搜索分枝。

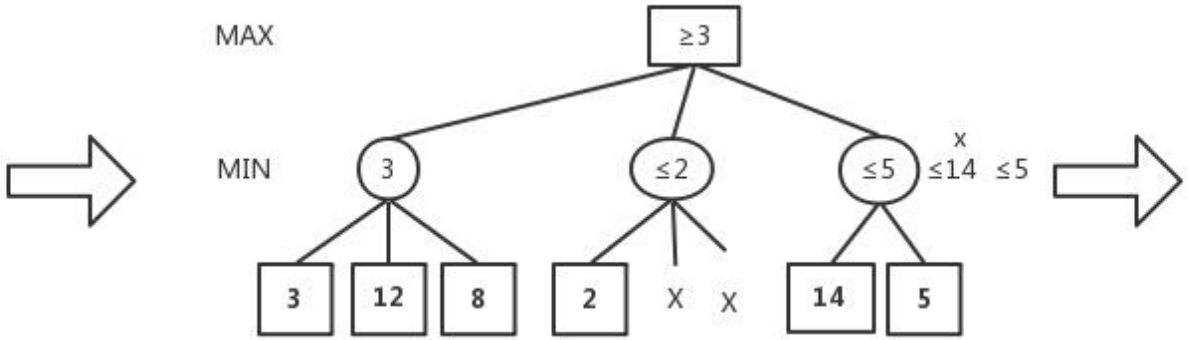
图中MIN选手所在的节点C下属分支4和6与根节点最终优化决策的取值无关，可不被访问。

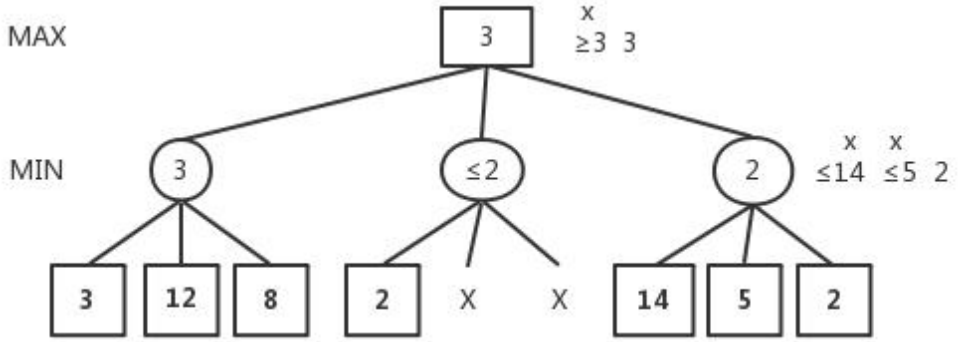


对抗搜索：Alpha-Beta 剪枝搜索





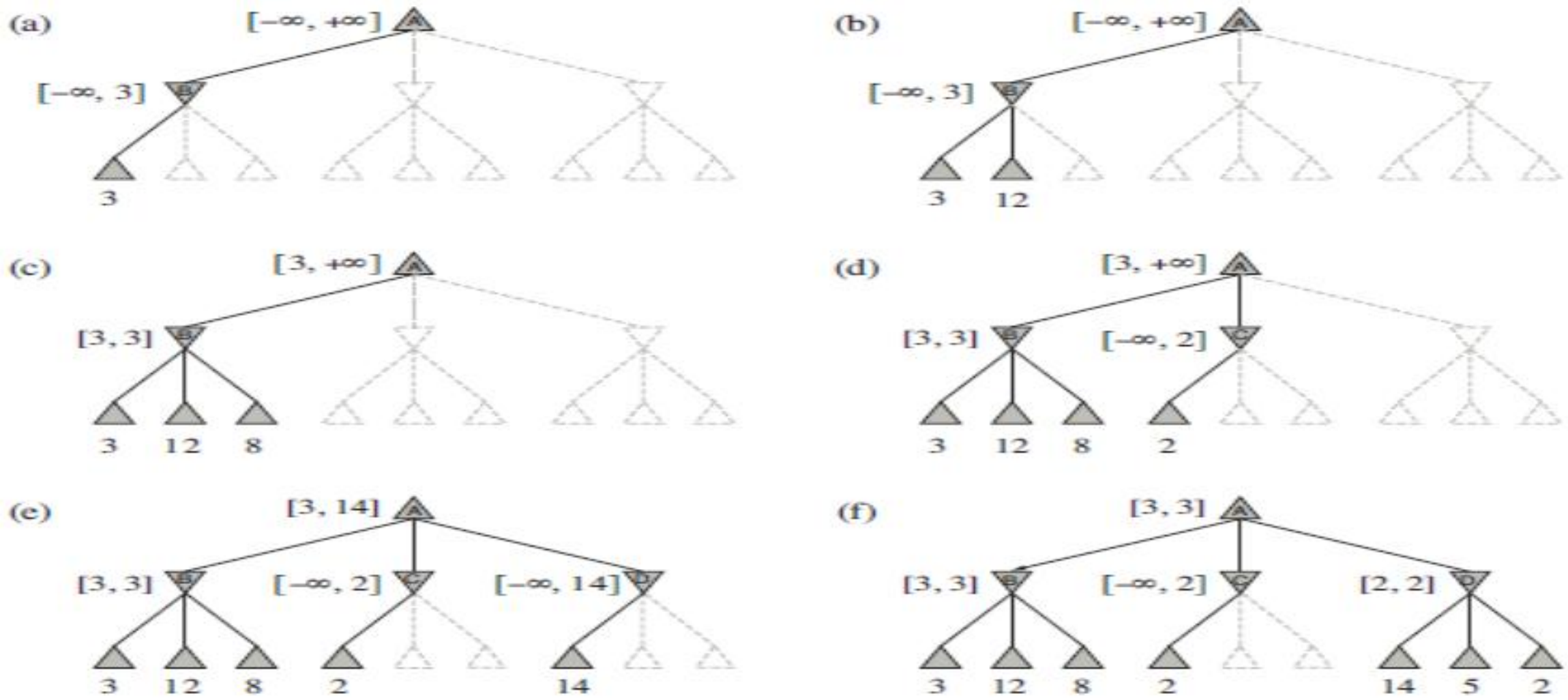




Alpha值(α)	MAX节点目前得到的最高收益
Beta值(β)	MIN节点目前可给对手的最小收益
α和β的值初始化分别设置为$-\infty$和∞	

对抗搜索： Alpha-Beta 剪枝搜索

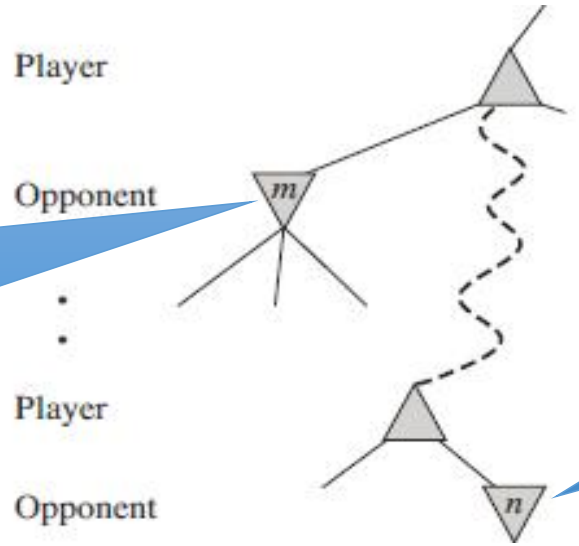
- 从α和β的变化来理解剪枝过程



对抗搜索：如何利用Alpha-Beta 剪枝

Alpha值(α)	玩家MAX(根节点)目前得到的最高收益
	假设n是MIN节点，如果n的一个后续节点可提供的收益小于α，则n及其后续节点可被剪枝
Beta值(β)	玩家MIN目前给对手的最小收益
	假设n是MAX节点，如果n的一个后续节点可获得收益大于β，则n及其后续节点可被剪枝
α和β的值初始化分别设置为$-\infty$和∞	

在图中$m > n$，因此n右边节点及后续节点就被剪枝掉了



对手节点(min节点)在这里可提供的最大收益是$\alpha = m$

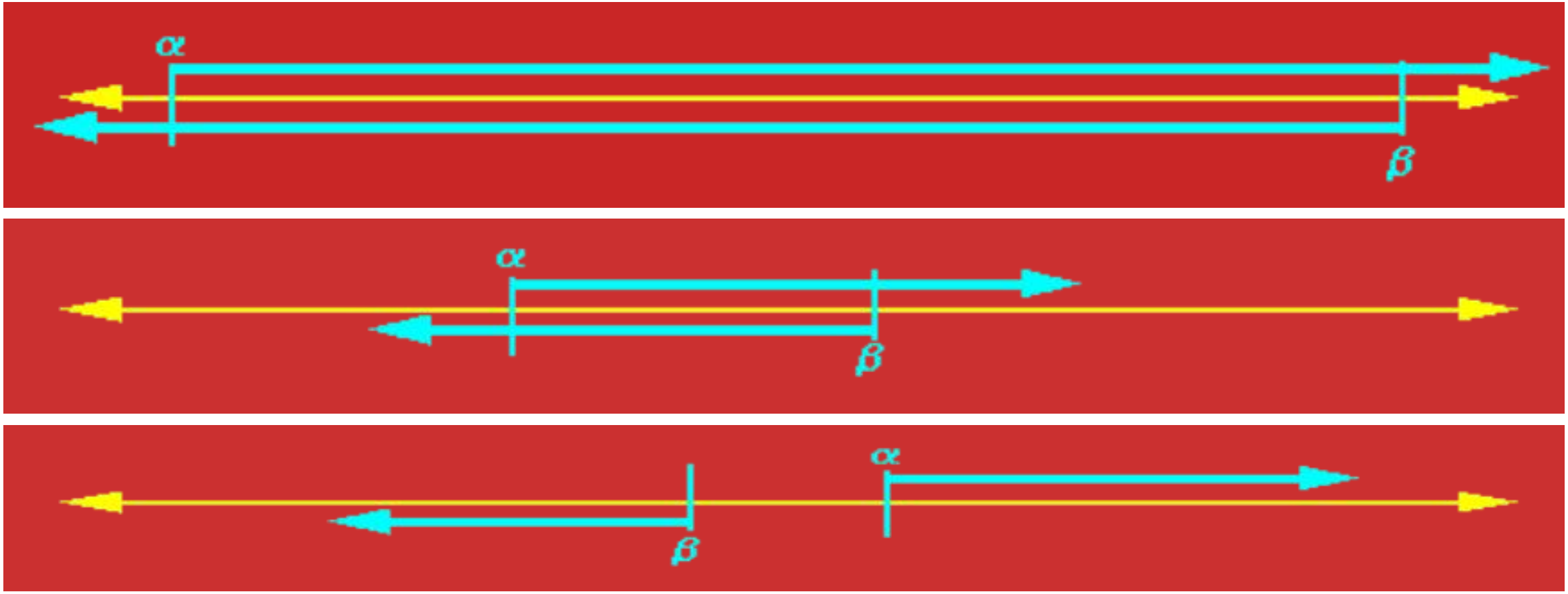
对手节点(min节点)在这里可提供的最大收益是？

对抗搜索：如何利用Alpha-Beta 剪枝

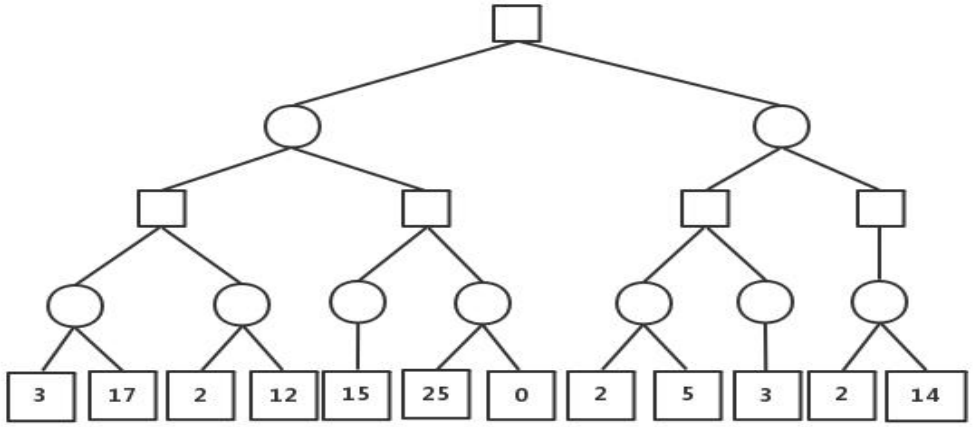
- α为可能解法的最小下界
- β为可能解法的最大上界
  - 如果节点N是可能解法路径中的一个节点，则其产生的收益一定满足如下条件：$\alpha \leq reward(N) \leq \beta$(其中$reward(N)$是节点 产生的收益)
  - 每个节点有两个值，分别是 和 。节点的 和 值在搜索过程中不断变化。其中， 从负无穷大$(-\infty)$逐渐增加、 从正无穷大(∞)逐渐减少。如果一个节点中 > ，则该节点的后续节点可剪枝。

对抗搜索：如何利用Alpha-Beta 剪枝

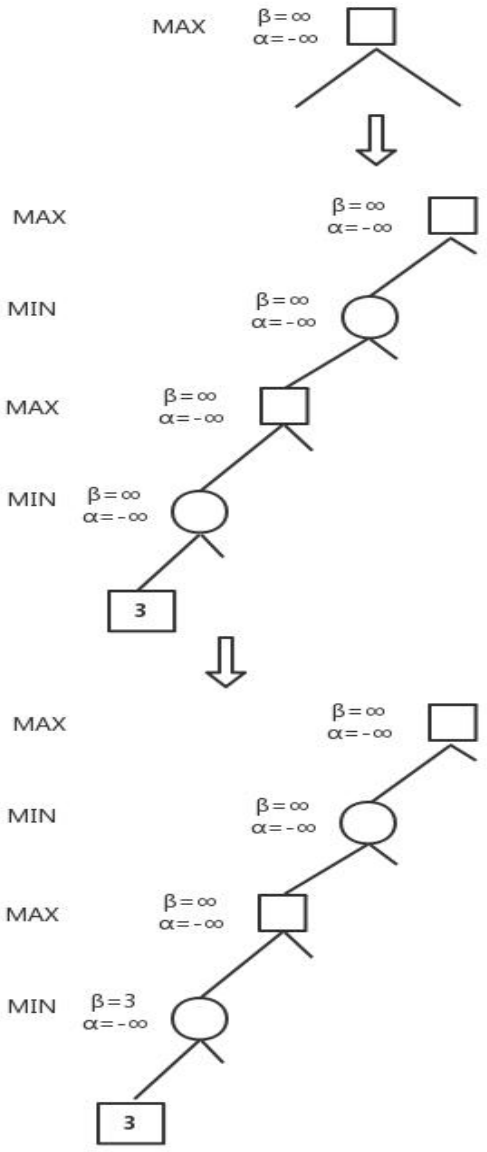
- α为可能解法的最小下界
- β为可能解法的最大上界

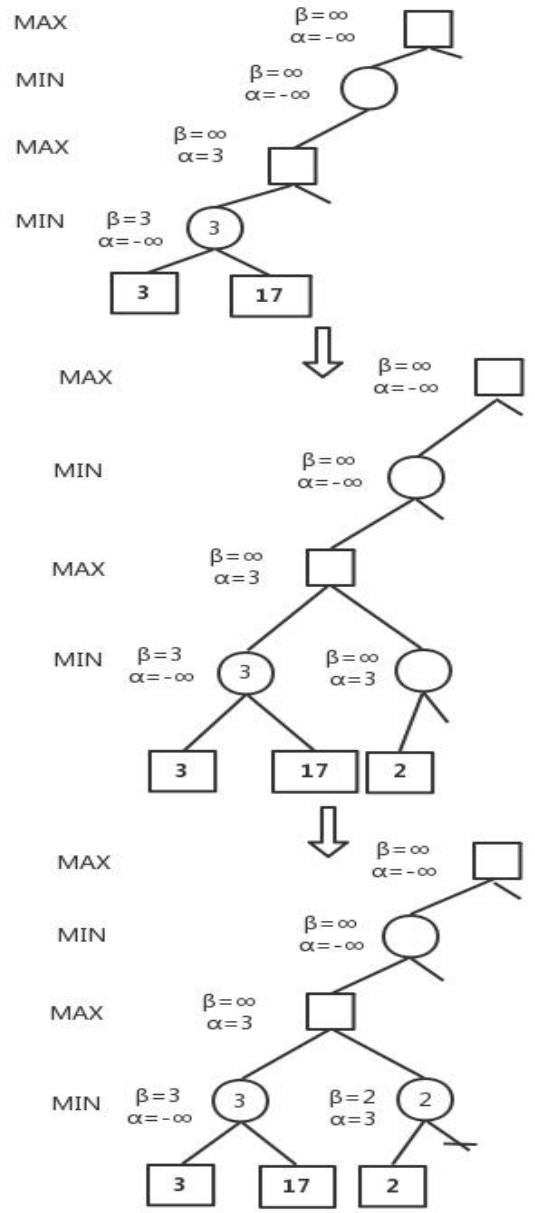


对抗搜索：Alpha-Beta 剪枝搜索示意

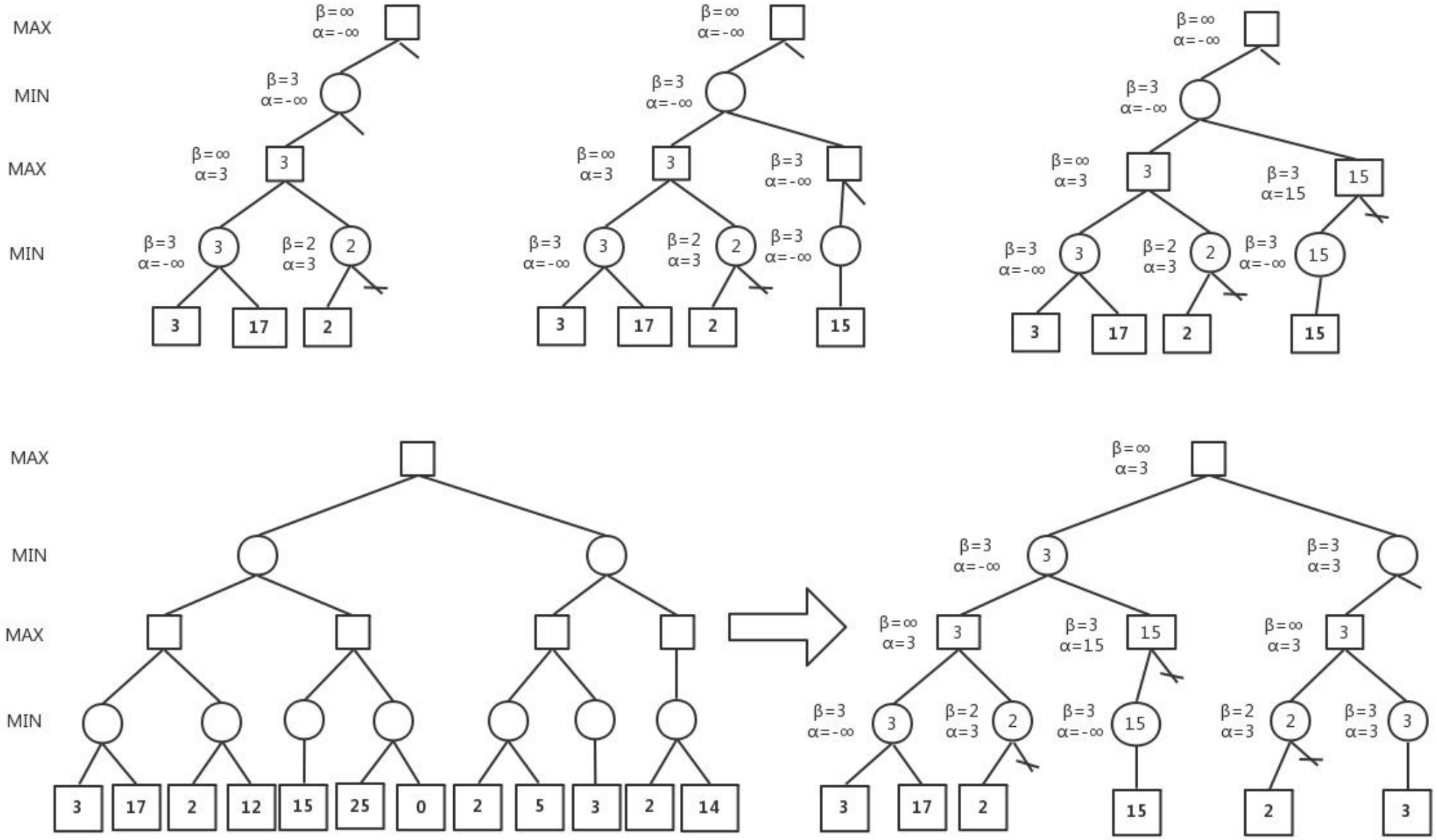


没有必要枚举，可以先继承
父节点的α, β，再按规则更新。





对抗搜索：Alpha-Beta 剪枝搜索示意



对抗搜索：Alpha-Beta 剪枝搜索的算法描述

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

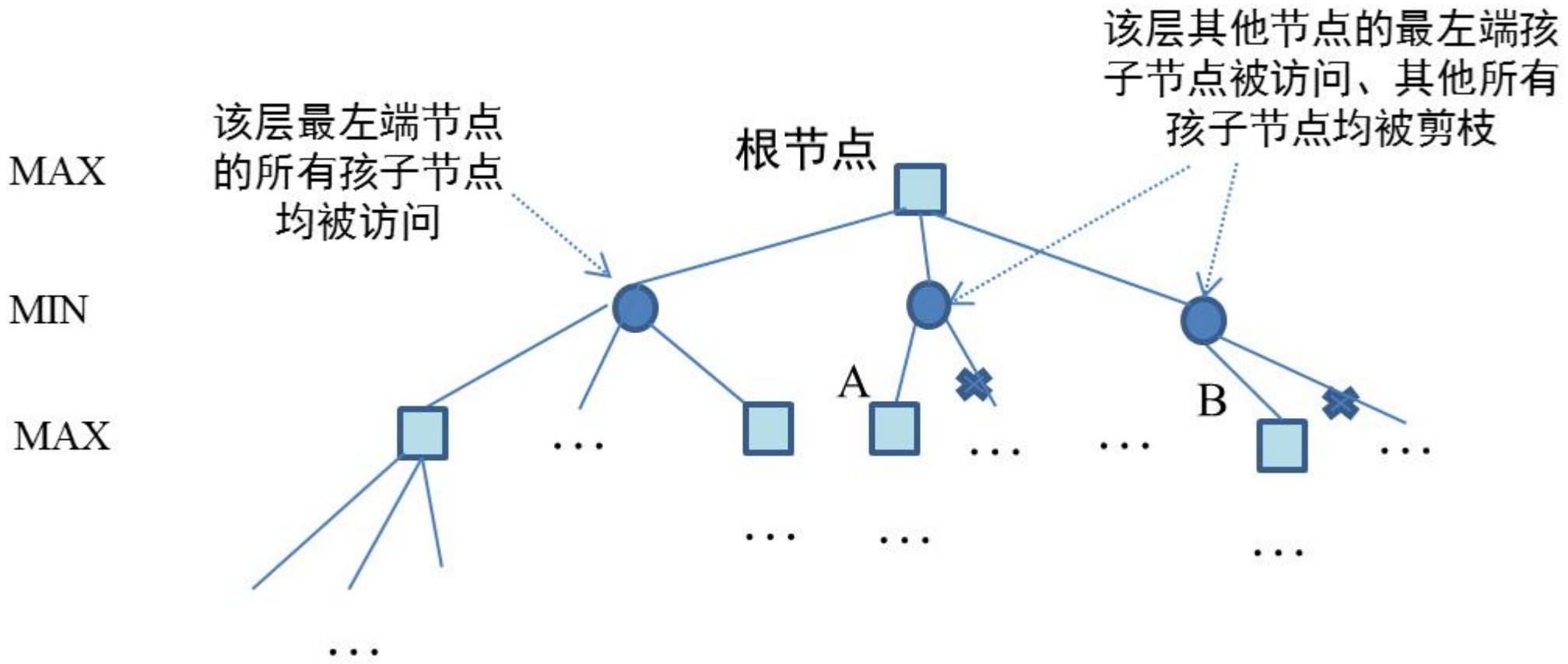
```
function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

对抗搜索：Alpha-Beta 剪枝搜索的性质

- 剪枝本身不影响算法输出结果
- 节点先后次序会影响剪枝效率
- 如果节点次序“恰到好处”，Alpha-Beta剪枝的时间复杂度为$O(b^{\frac{m}{2}})$，最小最大搜索的时间复杂度为$O(b^m)$

b	分支因子，即搜索树中每个节点最大的分支数目
m	搜索树中路径的最大可能长度

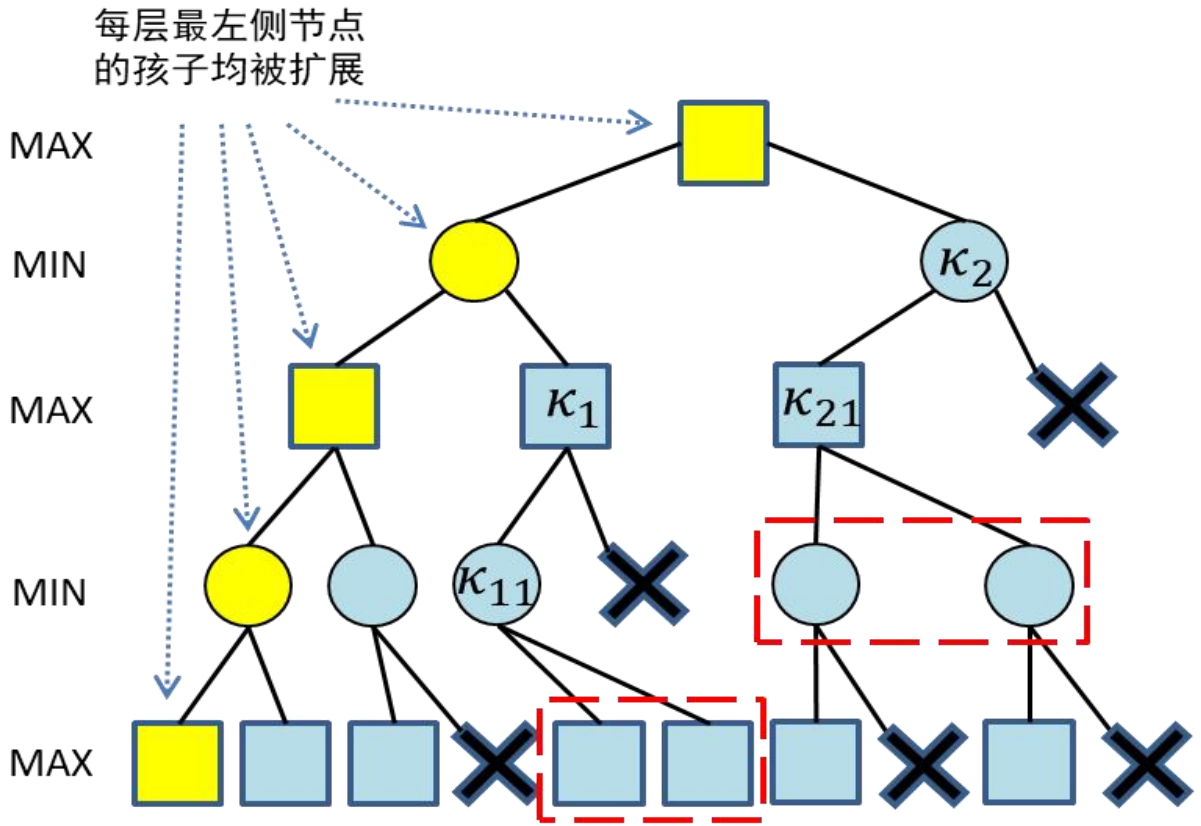
对抗搜索：Alpha-Beta 剪枝算法性能分析



最优状况下的剪枝结果示意图，每一层最左端结点的所有孩子结点均被访问，其他节点仅有最左端孩子结点被访问、其他孩子结点被剪枝。

对抗搜索：Alpha-Beta 剪枝算法性能分析

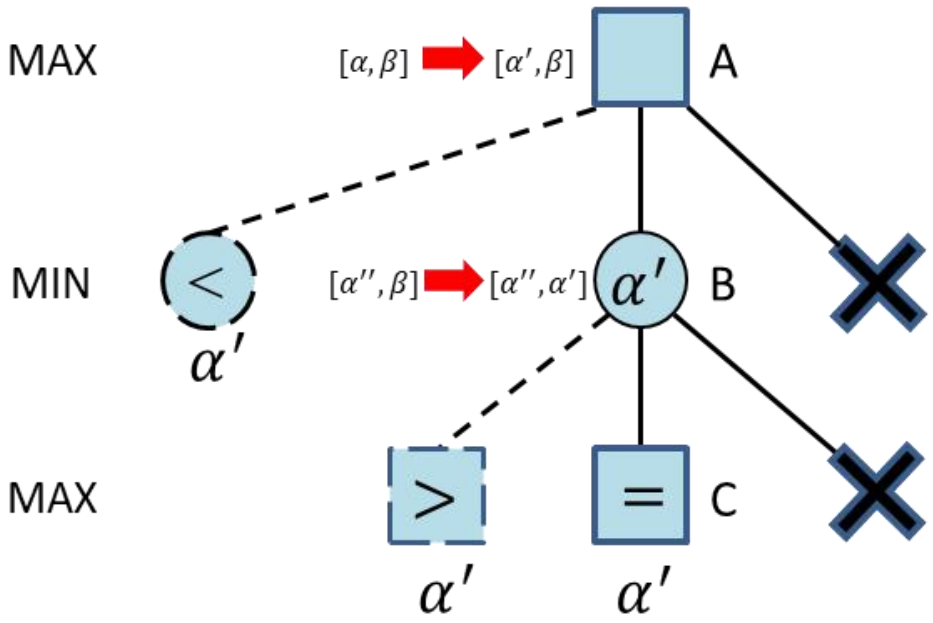
- 观察：如果一个节点导致了其兄弟节点被剪枝，可知其孩子节点必然被扩展



κ_{11}和κ_{21}两个节点分别导致了其右端的兄弟被剪枝，于是可以肯定κ_{11}和κ_{21}全部孩子节点必然被扩展。

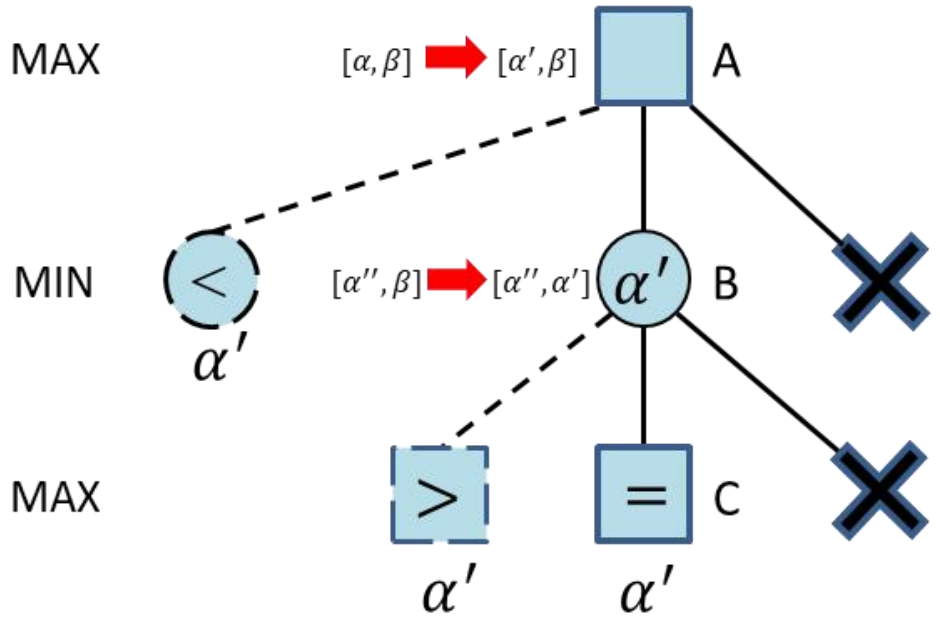
对抗搜索：Alpha-Beta 剪枝算法性能分析

- **不妨假设A结点为MAX节点。**
  - 假设节点A对应的初值是 $[\alpha, \beta]$
  - 如果A的孩子结点B导致了B右端的兄弟结点被剪枝。由于剪枝直到B的收益分数计算完后才发生，可知结点B必然是A已扩展孩子结点中收益分数最大的结点；
  - 同时根据剪枝的发生条件，可知B的收益分数α'必然满足 α' > α 。



对抗搜索：Alpha-Beta 剪枝算法性能分析

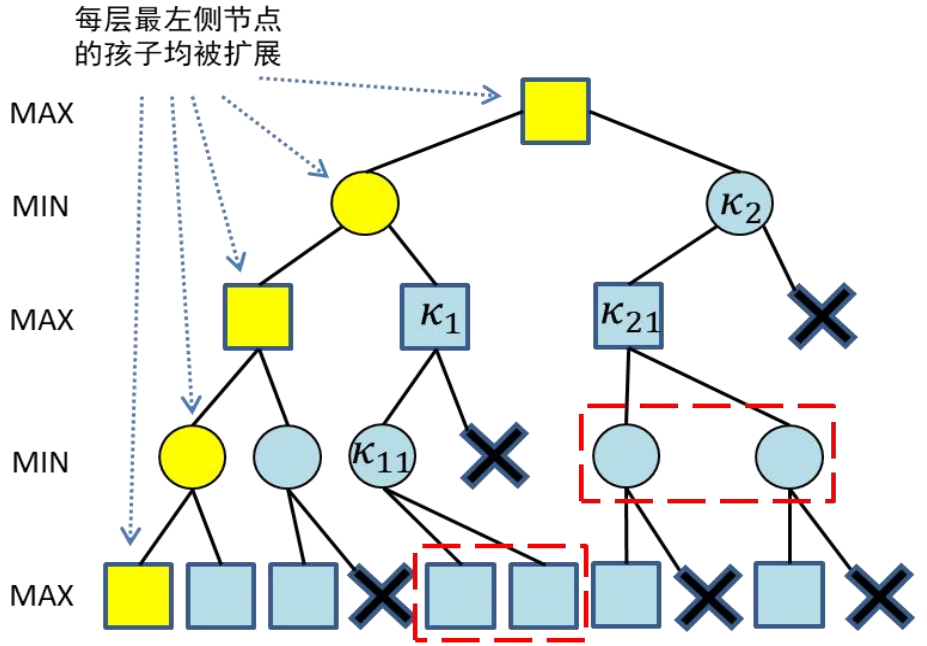
- 不妨假设A结点为MAX节点。
  - 假设扩展B时的上下界为$[\alpha'', \beta]$，其中α''来自B被扩展时A的下界
  - B至少有一个子结点C被扩展，因此$\alpha'' \leq \beta < \alpha'$，即$\alpha'' < \alpha'$
  - 如果C导致B的其余子结点被剪枝，则C是B已扩展子结点中收益最小的，B和C的收益同为α'，且必然满足$\alpha' < \alpha''$，该式与前面的不等式矛盾。



由此不难归纳出结论：如果某个结点导致了其右侧兄弟结点被剪枝，那么该节点的所有孩子结点必然已被全部扩展

对抗搜索：Alpha-Beta 剪枝算法性能分析

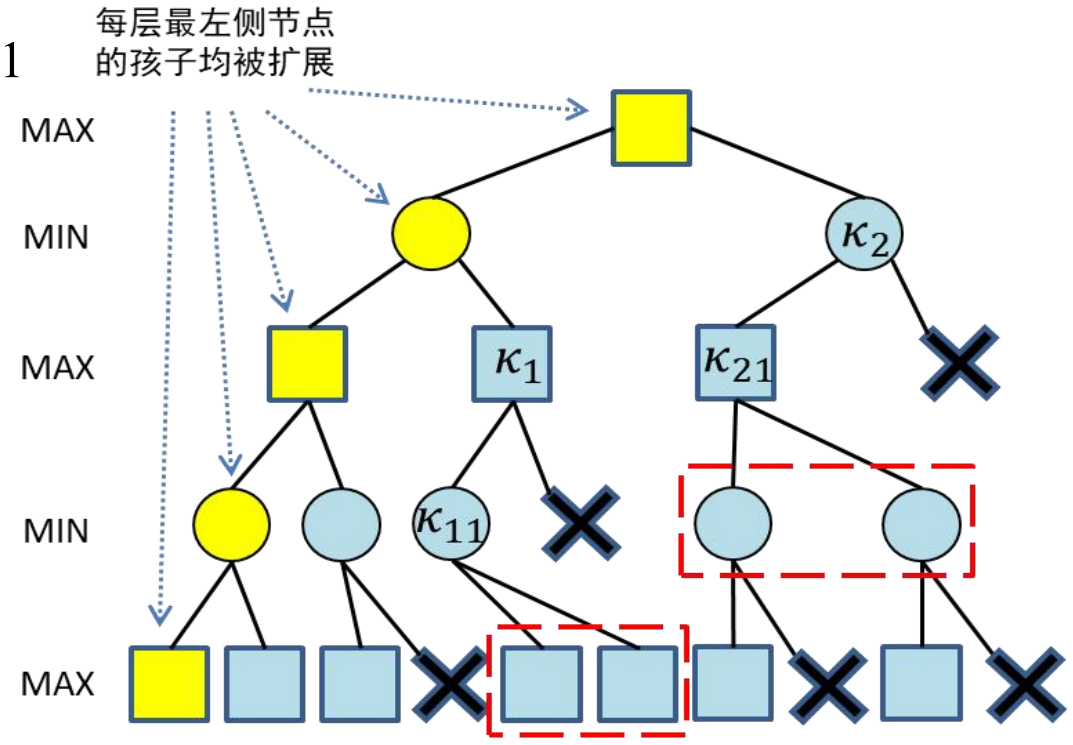
- 下图展示了alpha-beta剪枝算法效率最优的情况
  - 为了方便说明，假设图中每个结点恰好有 b 个子节点
  - 搜索树每层最左端结点的孩子结点必然全部被扩展。其他结点的孩子结点被扩展情况分为如下两类
    - 1)只扩展其最左端孩子结点；
    - 2)它是其父亲结点唯一被扩展的子结点，且它的所有子结点(如果存在)一定被扩展。



对抗搜索：Alpha-Beta 剪枝算法性能分析

- 最左侧子节点外其他子节点
  - 1)只扩展其最左端孩子结点，如κ_1和κ_2；
  - 2)它是其父亲结点唯一被扩展的子结点，且它的所有子结点(如果存在)一定被扩展，如κ_{11}和κ_{21}

Alpha-Beta剪枝在最优效率下扩展的结点数量为$O(b^{\frac{m}{2}})$，比起最小最大搜索的$O(b^m)$有显著提升。



提纲

- 搜索算法基础
- 启发式搜索
- 对抗搜索
- 蒙特卡洛树搜索

蒙特卡洛树搜索

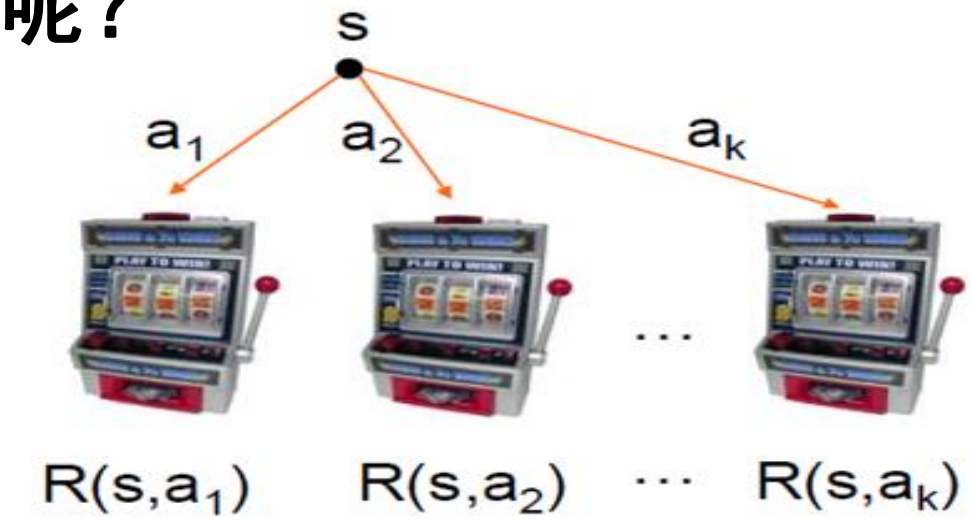
- **推荐阅读材料**
  - D. Silver, et.al., Mastering the game of Go with Deep Neural Networks and Tree Search, Nature, 529:484-490,2016
  - C. Browne, et.al., Survey of Monte Carlo Tree Search Methods, IEEE Transactions on Computational Intelligence and AI in Games, 4(1):1-49,2012
  - S. Gelly, et al., The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions, Communications of the ACM, 55(3):106-113,2012
  - L. Kocsis and C. Szepesvari, Bandit Based Monte-Carlo Planning, ECML 2006
  - P. Auer, et. al., Finite-time analysis of the multi-armed bandit problem, Machine learning, 47(2), 235-256, 2002

蒙特卡洛规划 (Monte-Carlo Planning)

- **单一状态蒙特卡洛规划： 多臂赌博机 (multi-armed bandits)**
- **上限置信区间策略 (Upper Confidence Bound Strategies, UCB)**
- **蒙特卡洛树搜索 (Monte-Carlo Tree Search)**
  - UCT (Upper Confidence Bounds on Trees)

单一状态蒙特卡洛规划： 多臂赌博机

- 单一状态， 种行动(即有 个摇臂)
- 在摇臂赌博机问题中，每次以随机采样形式采取一种行动 ，好比随机拉动第 个赌博机的臂膀，得到$R(s, a_k)$ 的回报。
- 问题：下一次需要拉动那个赌博机的臂膀，才能获得最大回报呢？



多臂赌博机

- **多臂赌博机问题是一种序列决策问题，这种问题需要在利用(exploitation)和探索(exploration) 之间保持平衡。**
  - 利用(exploitation)：保证在过去决策中得到最佳回报
  - 探索(exploration)：寄希望在未来能够得到更大回报
- **如果有k个赌博机，这k个赌博机产生的操作序列为$X_{i,1}, X_{i,2}, \ldots$ $(\mathrm{i} = 1, \ldots, \mathrm{K})$。在时刻$t = 1,2\ldots$，选择第$I_t$个赌博机后，可得到奖赏$X_{I_t,t}$，则在$n$次操作$I_1, \ldots, I_n$后，可如下定义悔值函数：**

$$R_n = \max_{i=1,\ldots,k} \sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}$$

多臂赌博机

$$R_n = \max_{i=1,\ldots,k} \sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}$$

- 悔值函数表示了在第t次对赌博机操作时，假设知道哪个赌博机能够给出最大奖赏(虽然现实中不存在)，则将得到的最大奖赏减去实际操作第I_t个赌博机所得到的奖赏。将n次操作的差值累加起来，就是悔值函数的结果。
- 很显然，一个良好的多臂赌博机操作的策略是在不同人进行了多次玩法后，能够让悔值函数的方差最小。

上限置信区间 (Upper Confidence Bound, UCB)

- 在多臂赌博机的研究过程中，上限置信区间成为一种较为成功的策略学习方法，因为其在探索-利用之间取得平衡。
- 在UCB方法中，使$X_{i,T_i\,(t-1)}$来记录第i个赌博机在过去$t-1$时刻内的平均奖赏，则在第t时刻，选择使如下具有最佳上限置区间的赌博机：

$$I_t = \max_{i\in\{1,\ldots,k\}} \{\overline{X_{i,T_i(t-1)}} + c_{t-1,T_i(t-1)}\}$$

- 其中$c_{t,s}$取值定义如下：$c_{t,s} = \sqrt{\frac{2\ln \boldsymbol{n}}{s}}$
- $T_i(t) = \sum_{s=1}^{t} \prod (I_s = i)$为在过去时刻(初始时刻到$t$时刻)过程中选择第$i$个赌博机的次数总和。

上限置信区间 (Upper Confidence Bound, UCB)

- 也就是说，在第 时刻，UCB算法一般会选择具有如下最大值的第 个赌博机：

$$UCB = \overline{X_j} + \sqrt{\frac{2\ln n}{n_j}} \text{ 或者 } UCB = \overline{X_j} + C \times \sqrt{\frac{2\ln n}{n_j}}$$

$\overline{X_j}$是第j个赌博机在过去时间内所获得的平均奖赏值，n_j是在过去时间内拉动第j个赌博机臂膀的总次数，n是过去时间内拉动所有赌博机臂膀的总次数。C是一个平衡因子，其决定着在选择时偏重探索还是利用。

上限置信区间 (Upper Confidence Bound, UCB)

- 也就是说，在第 时刻，UCB算法一般会选择具有如下最大值的第 个赌博机：

$$UCB = \overline{X_j} + \sqrt{\frac{2\ln n}{n_j}} \text{ 或者 } UCB = \overline{X_j} + C \times \sqrt{\frac{2\ln n}{n_j}}$$

从这里可看出UCB算法如何在探索-利用之间寻找平衡：既需要拉动在过去时间内获得最大平均奖赏的赌博机，又希望去选择那些拉动臂膀次数最少的赌博机。

上限置信区间

- **UCB算法描述**

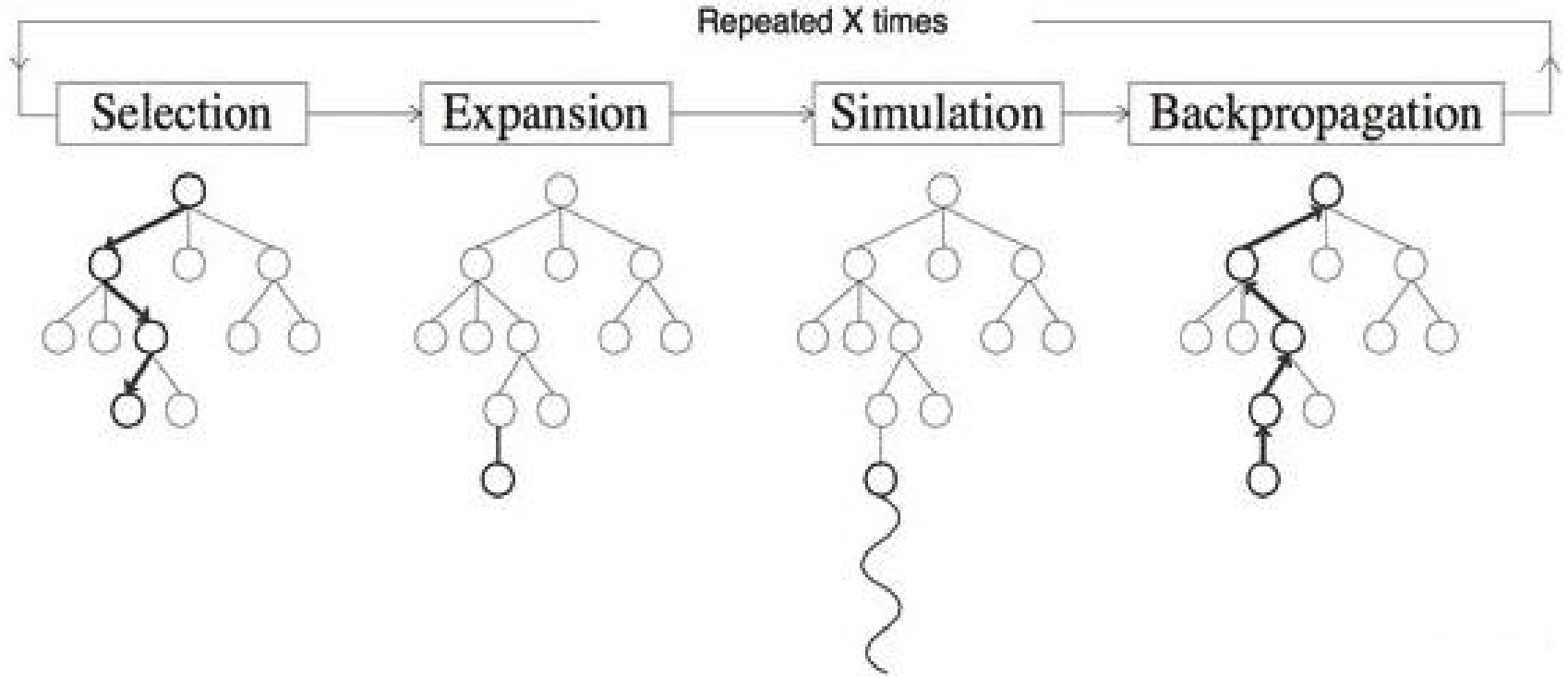
Deterministic policy: UCB1.

Initialization: Play each machine once.

Loop: Play machine j that maximizes $\bar{x}_j + \sqrt{\frac{2\ln n}{n_j}}$, where $\bar{x}_j$ is the average reward obtained from machine j, n_j is the number of times machine j has been explored so far, and n is the overall number of plays done so far.

蒙特卡洛树搜索(Monte-Carlo Tree Search, MCTS)

- 将上限置信区间算法UCB应用于游戏树的搜索方法
  - 包括了四个步骤：选择(selection)，扩展(expansion)，模拟(simulation)，反向传播(Back-Propagation)

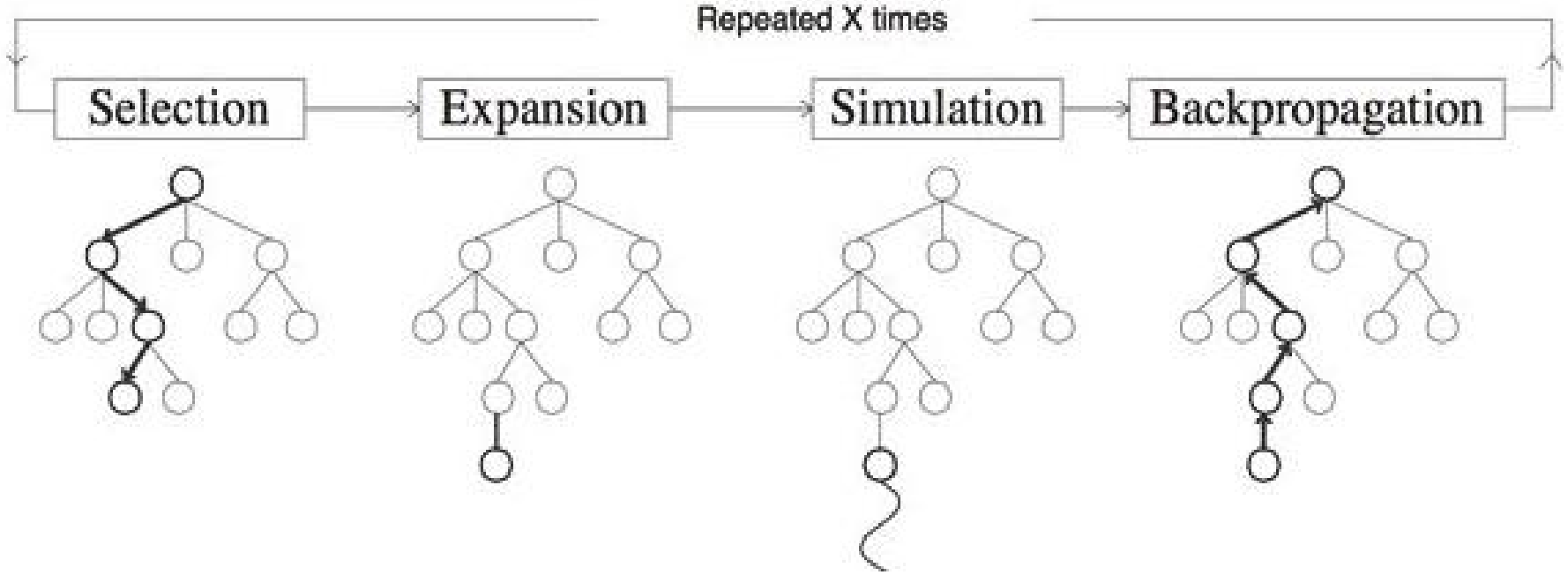


L. Kocsis and C. Szepesvari, Bandit based Monte-Carlo Planning, ***ECML***, 2006:282–293

蒙特卡洛树搜索

- **选择： 从根节点 R 开始，递归选择子节点，直至到达叶节点或到达具有还未被扩展过的子节点的节点L。**
  - 具体来说，通常用UCB1 (Upper Confidence Bound，上限置信区间)选择最具“潜力”的后续节点

$$UCB = \overline{X_j} + \sqrt{\frac{2\ln n}{n_j}}$$

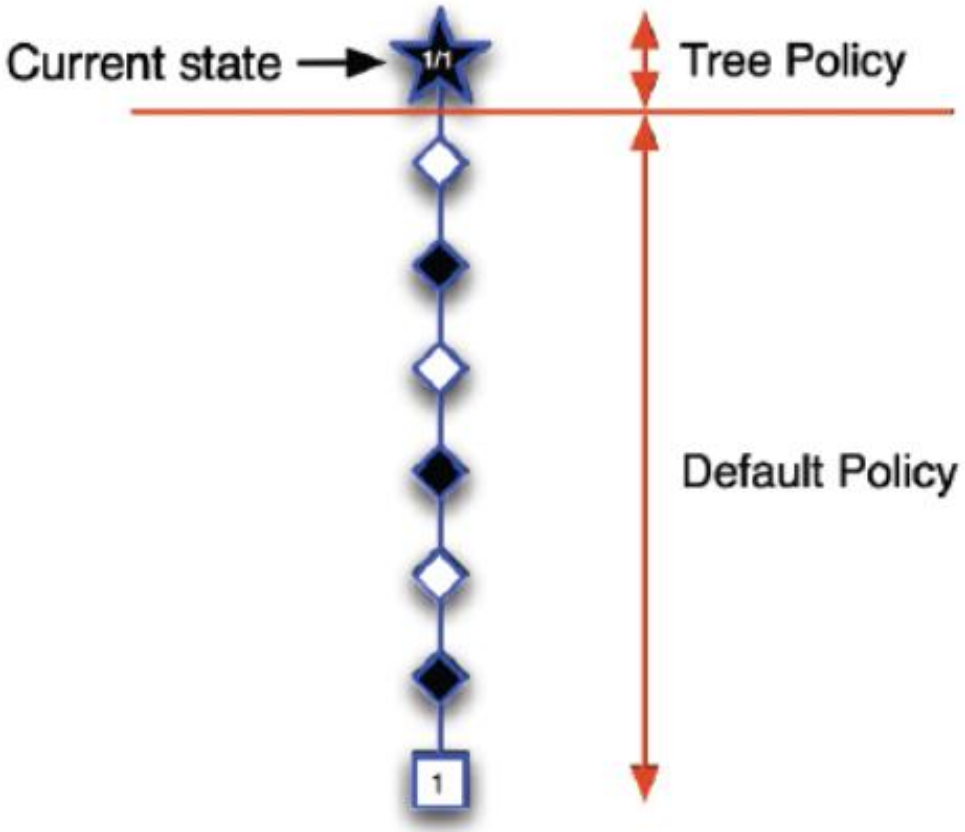


蒙特卡洛树搜索

- **扩展：**
  - 如果L不是一个终止节点，则随机创建其后的一个未被访问节点，选择该节点作为后续子节点C。
- **模拟：**
  - 从节点C出发，对游戏进行模拟，直到博弈游戏结束。
- **反向传播**
  - 用模拟所得结果来回溯更新导致这个结果的每个节点中获胜次数和访问次数。

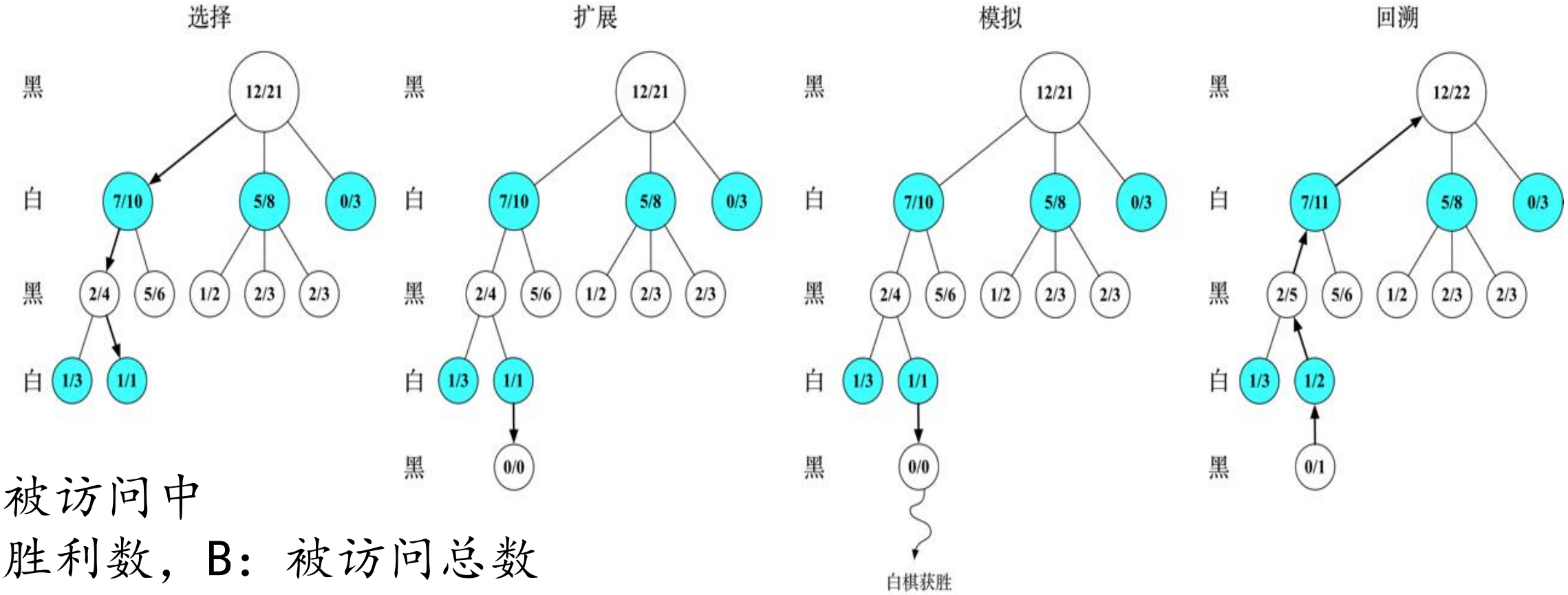
蒙特卡洛树搜索

- **两种策略学习机制：**
  - **搜索树策略**：从已有的搜索树中选择或创建一个叶子结点(即**选择**和**拓展**)。搜索树策略需要在利用和探索之间保持平衡。
  - **模拟策略**：从非叶子结点出发模拟游戏，得到游戏仿真结果。



蒙特卡洛树搜索：例子

- 以围棋为例，假设根节点是执黑棋方。
- 图中每一个节点都代表一个局面，每一个局面记录两个值A/B：



该局面被访问中

A:黑棋胜利数，B：被访问总数

蒙特卡洛树搜索：例子

- 假设由黑棋行棋，取一个UCB1值最大的节点作为后续节点：

左1：7/10对应

$$\frac{7}{10}+\sqrt{\frac{\log(21)}{10}}=1.252$$

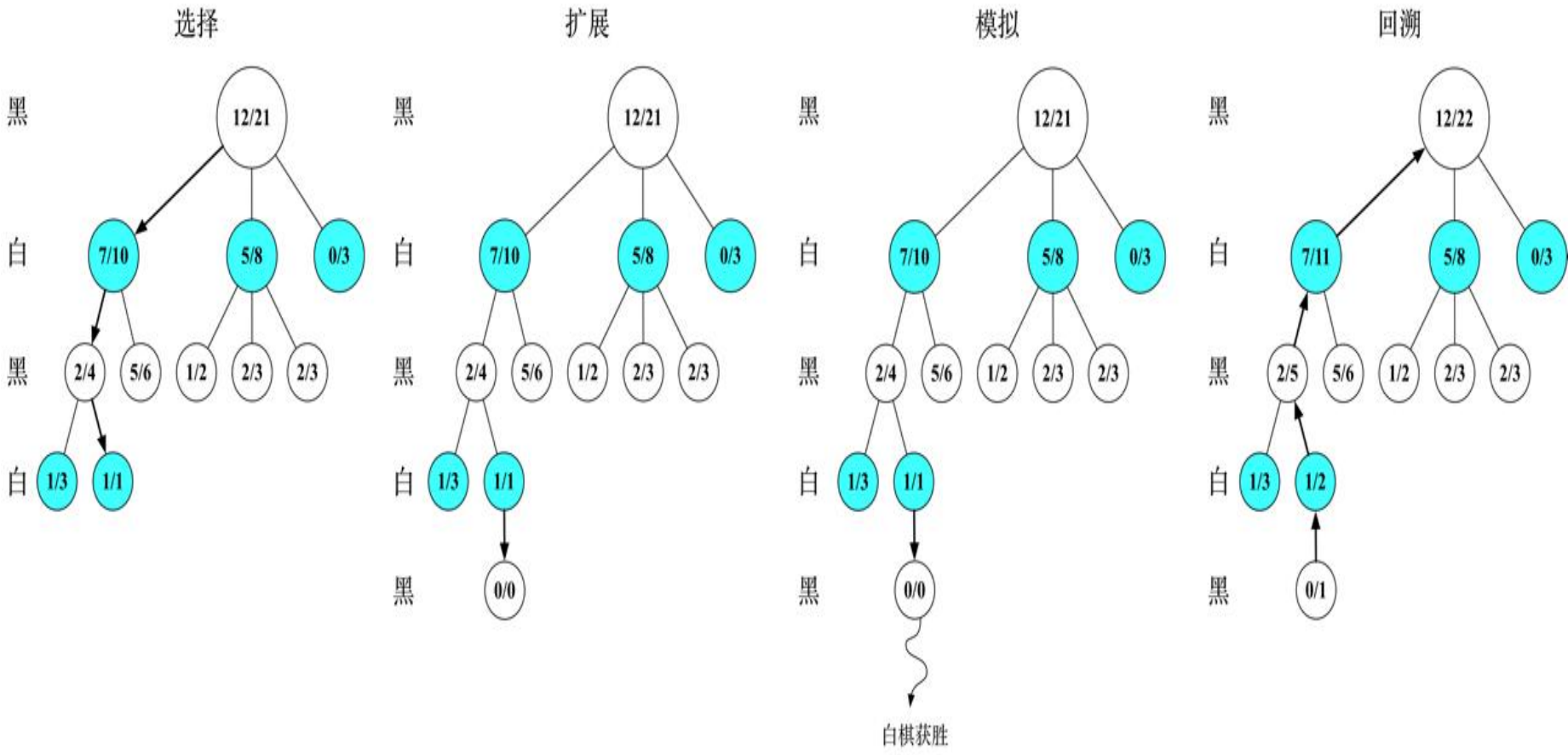
左2，5/8对应

$$\frac{5}{8}+\sqrt{\frac{\log(21)}{8}}=1.243$$

左3，0/3对应

$$\frac{0}{3}+\sqrt{\frac{\log(21)}{3}}=1.007$$

黑棋会选择局面7/10



蒙特卡洛树搜索：例子

- 在节点7/10，由白棋行棋(第一项为1-A/B)，由UCB1公式可得

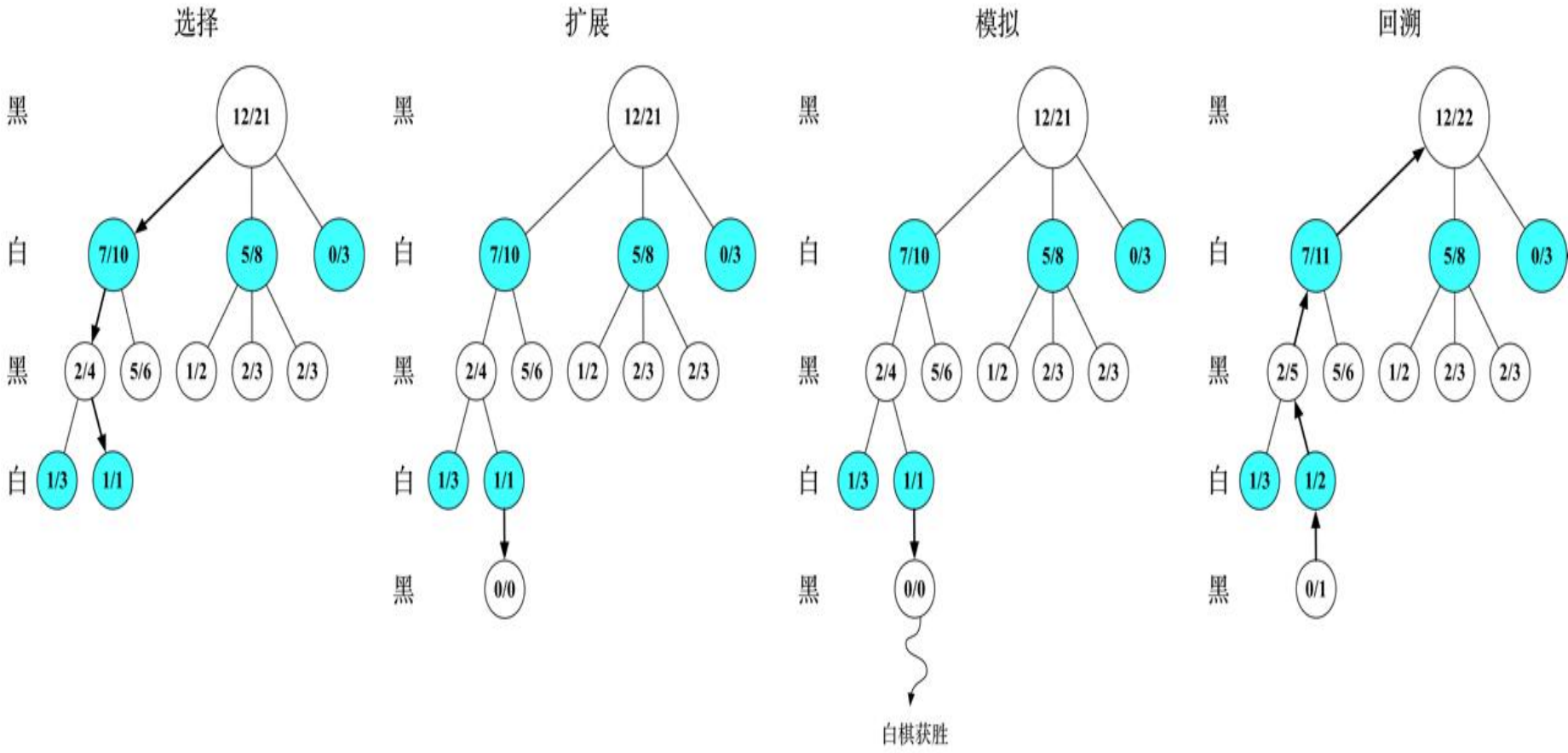
左1，2/4对应

$$\frac{2}{4}+\sqrt{\frac{\log(10)}{4}}=1.26$$

左2，5/6对应

$$\frac{1}{6}+\sqrt{\frac{\log(10)}{6}}=0.786$$

白棋会选择局面2/4



蒙特卡洛树搜索：例子

- 在节点2/4，黑棋评估下面的两个局面，由UCB1公式可得

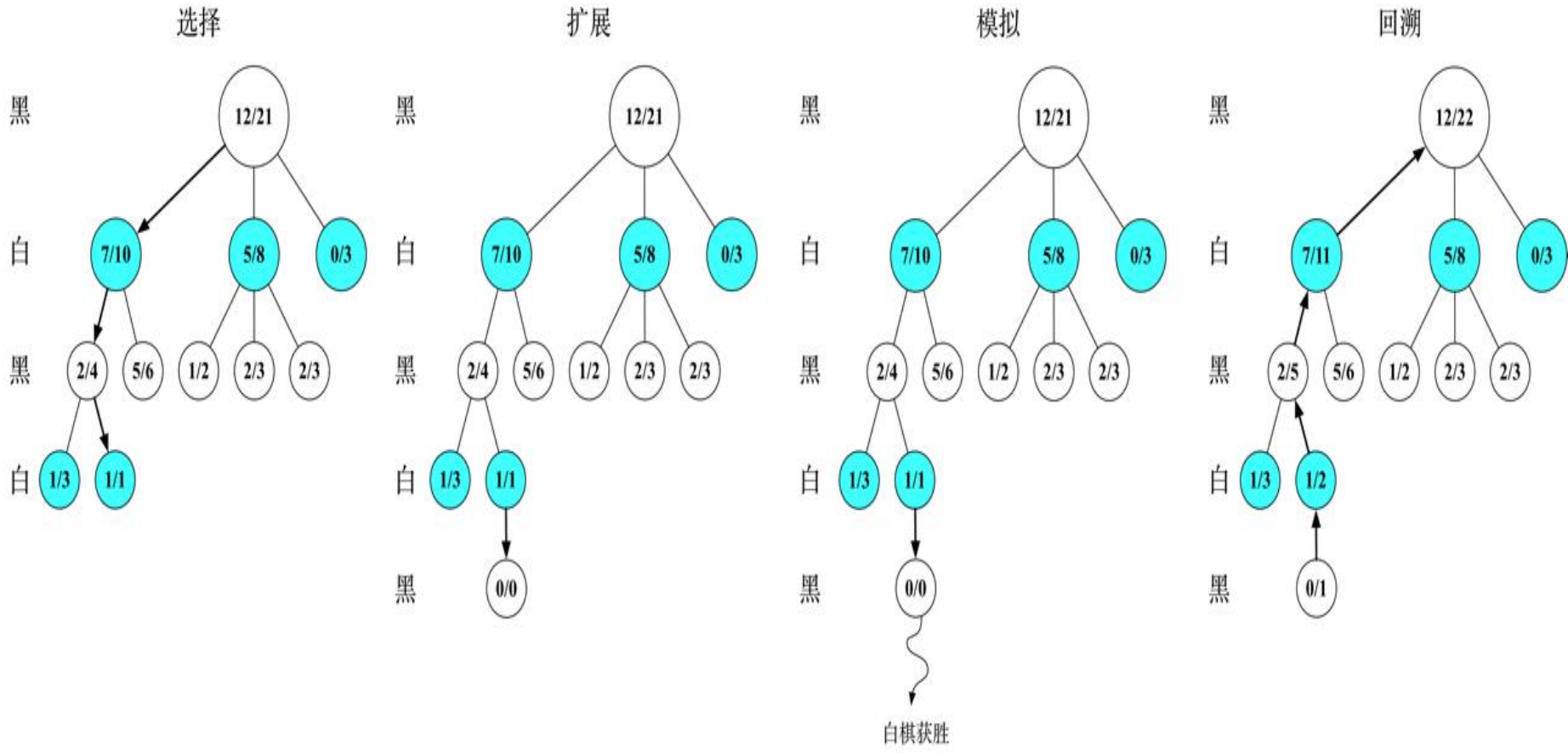
左1，1/3对应

$$\frac{1}{3}+\sqrt{\frac{\log(4)}{3}}=1.01$$

左1，1/1对应

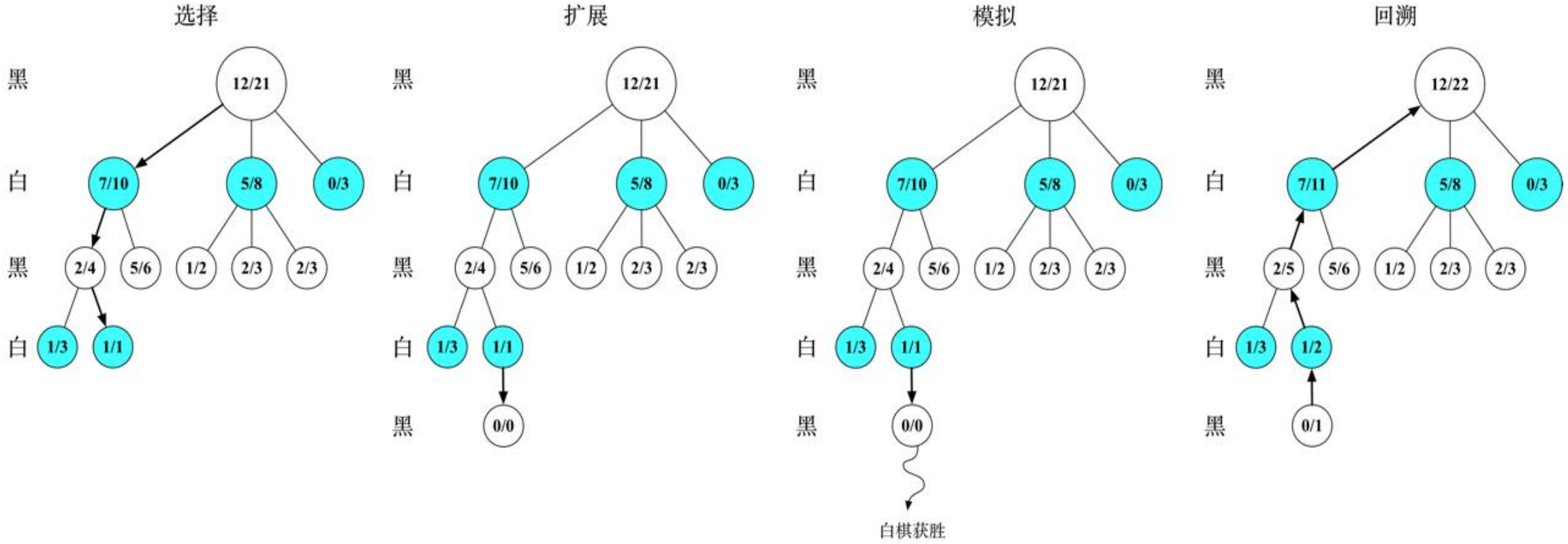
$$\frac{1}{1}+\sqrt{\frac{\log(4)}{1}}=2.18$$

黑棋会选择局面1/1。已达叶结点，需要扩展



蒙特卡洛树搜索：例子

- 随机扩展一个新节点，初始化为0/0，在该节点下进行模拟
- 假设经过一系列仿真行棋后，最终白棋获胜。该新节点的A/B值被更新为0/1，并向上回溯，所有父辈节点的A不变，B加1



使用蒙特卡洛树搜索的原因

- 蒙特卡洛树搜索基于采样来得到结果、而非穷尽式枚举 (虽然在枚举过程中也可剪掉若干不影响结果的分支)。

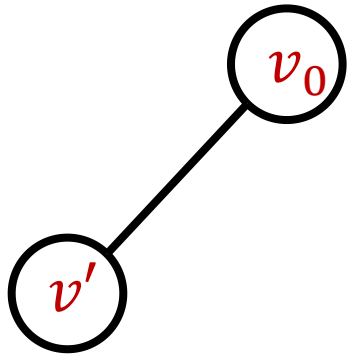
蒙特卡洛树搜索算法

$\boldsymbol{S}$	状态集
$\boldsymbol{A(s)}$	在状态s能够采取的有效行动的集合
$\boldsymbol{s(v)}$	节点v所代表的状态
$\boldsymbol{a(v)}$	所采取的行动导致到达节点v
$\boldsymbol{f}: S \times \mathrm{A} \rightarrow S$	状态转移函数
$\boldsymbol{N(v)}$	节点v被访问的次数
$\boldsymbol{Q(V)}$	节点v所获得的奖赏值
$\boldsymbol{\Delta(v,p)}$	玩家p选择节点v所得到的奖赏值

蒙特卡洛树搜索算法(UCT)

S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

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



function TREEPOLICY(v)
while v is nonterminal **do**
if v not fully expanded **then**
return EXPAND(v)
else
$v \leftarrow$ BESTCHILD(v, Cp)
return 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 BESTCHILD(v, c)
return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$

function BACKUP(v, Δ)
while v is not null **do**
$N(v) \leftarrow N(v) + 1$
$Q(v) \leftarrow Q(v) + \Delta(v, p)$
$v \leftarrow$ parent of 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'

蒙特卡洛树搜索算法(UCT)

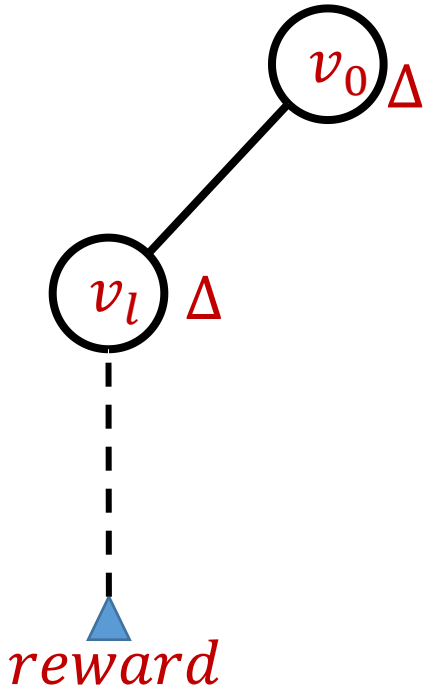
S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

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

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return 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 BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2\ln N(v)}{N(v')}}$



function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of 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'

蒙特卡洛树搜索算法(UCT)

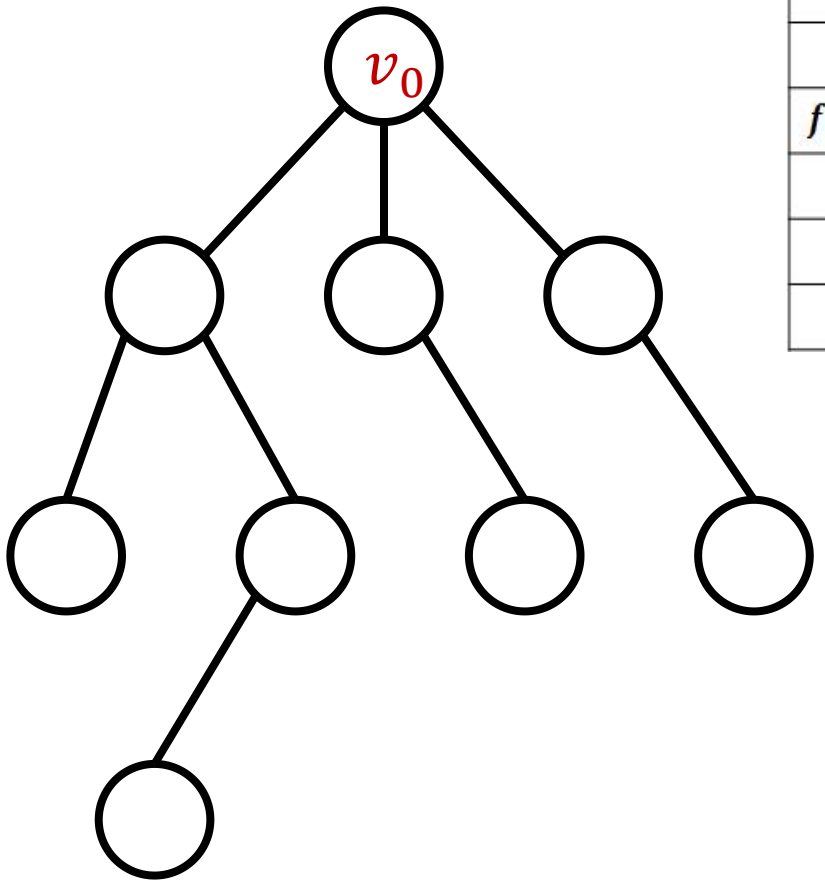
S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

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

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return 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 BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$



function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of 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'

蒙特卡洛树搜索算法(UCT)

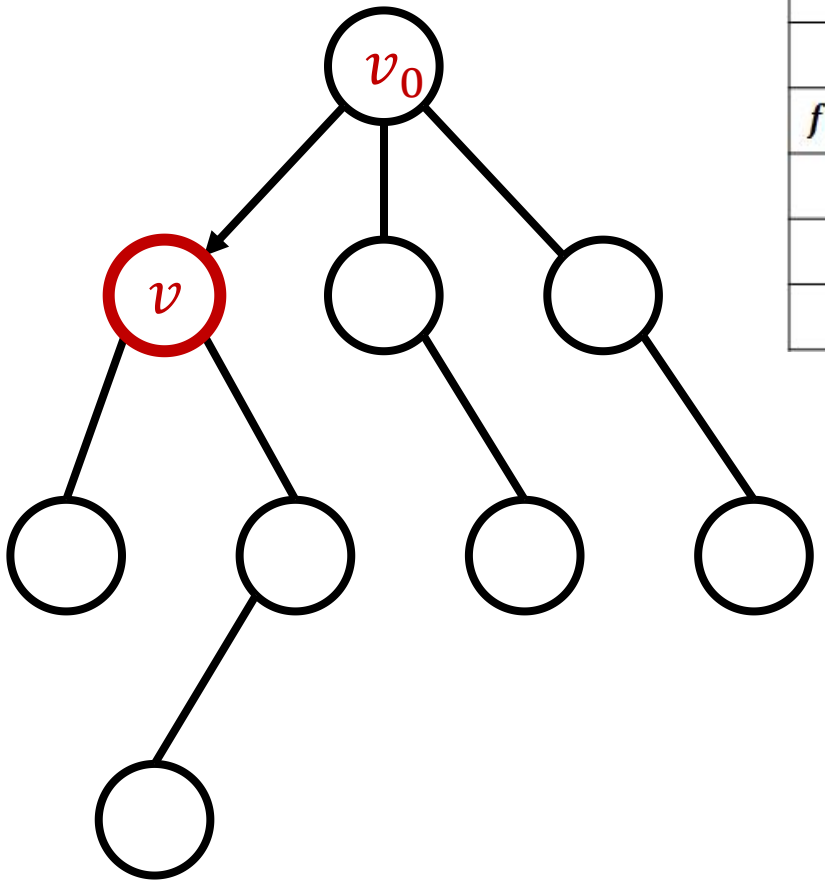
S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

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

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return 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 BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2\ln N(v)}{N(v')}}$



function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of 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'

蒙特卡洛树搜索算法(UCT)

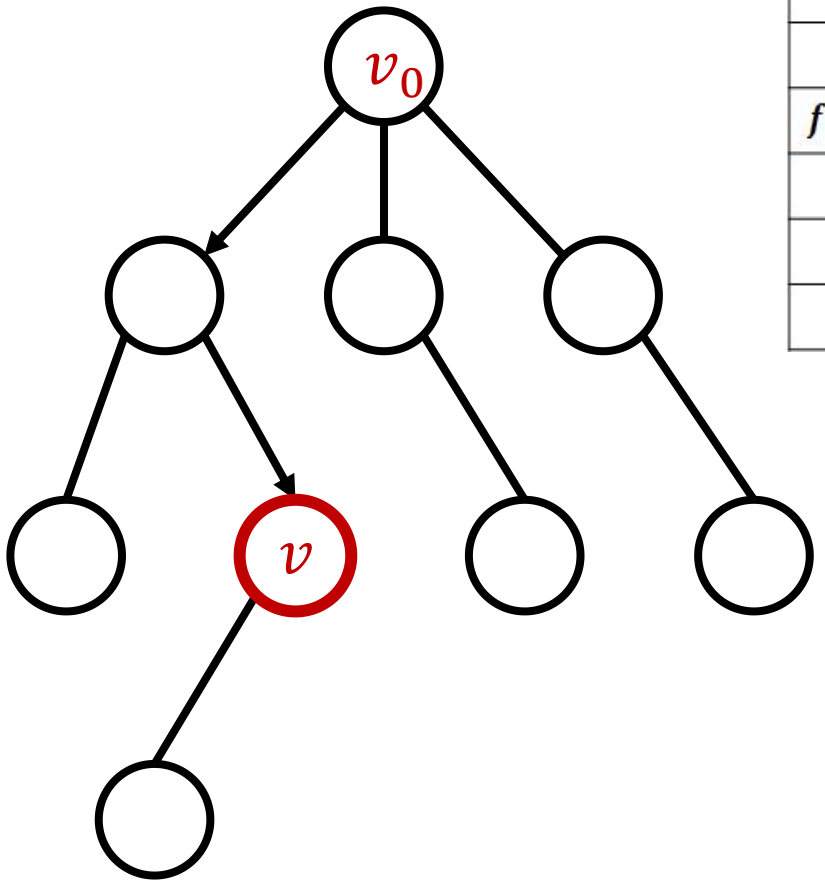
S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

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

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return 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 BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$



function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of 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'

蒙特卡洛树搜索算法(UCT)

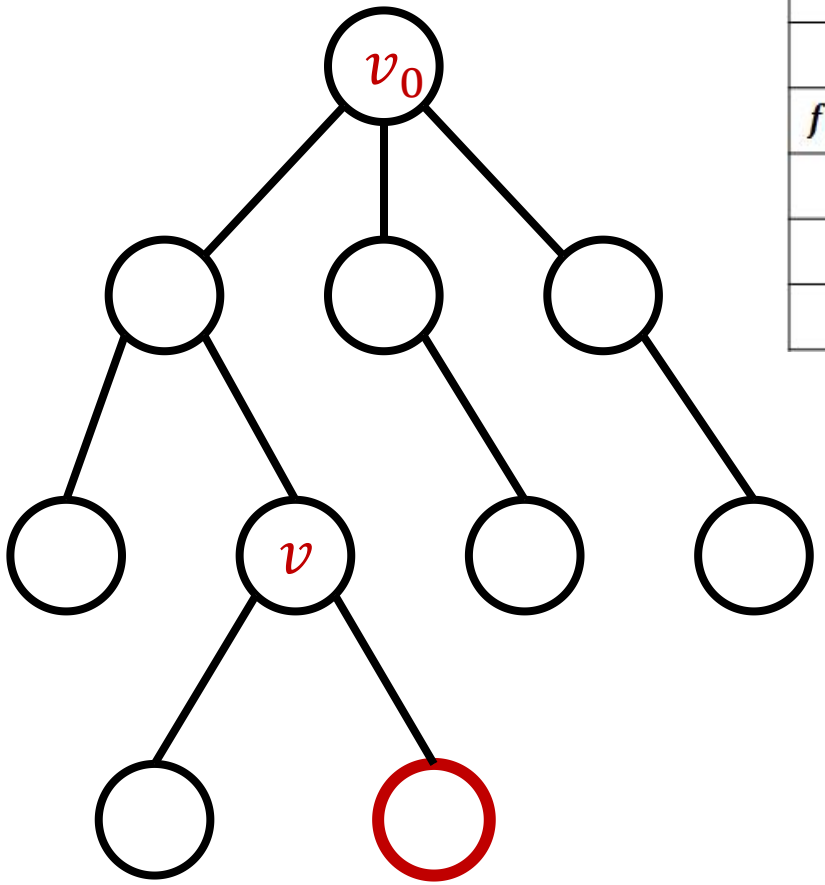
S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

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

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return 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 BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$



function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of 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'

蒙特卡洛树搜索算法(UCT)

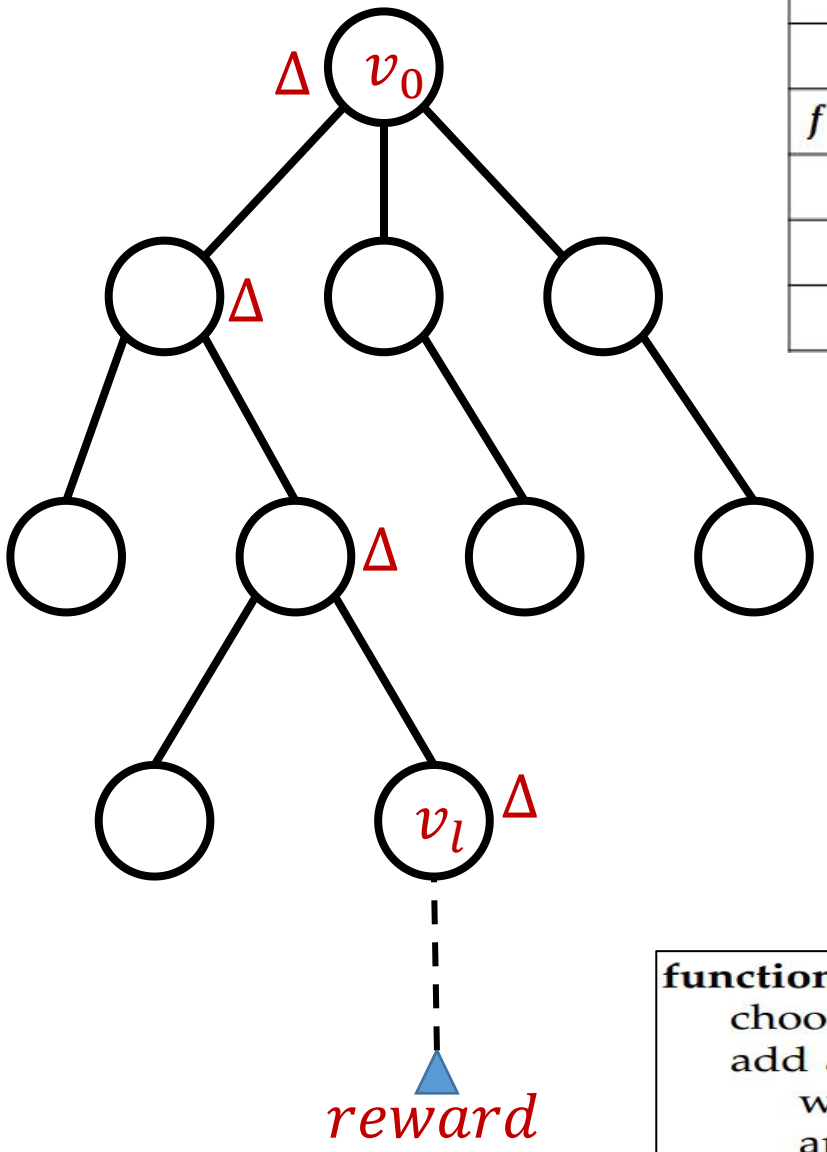
S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

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

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return 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 BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$



function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of 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'

蒙特卡洛树搜索算法(UCT)

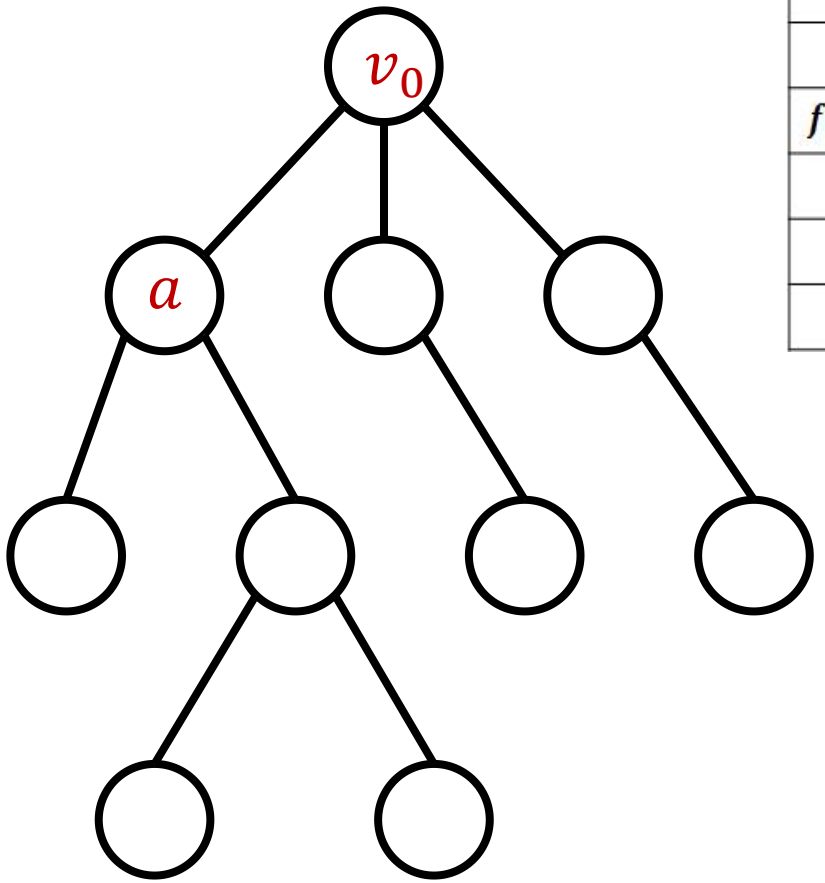
S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

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

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return 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 BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2\ln N(v)}{N(v')}}$

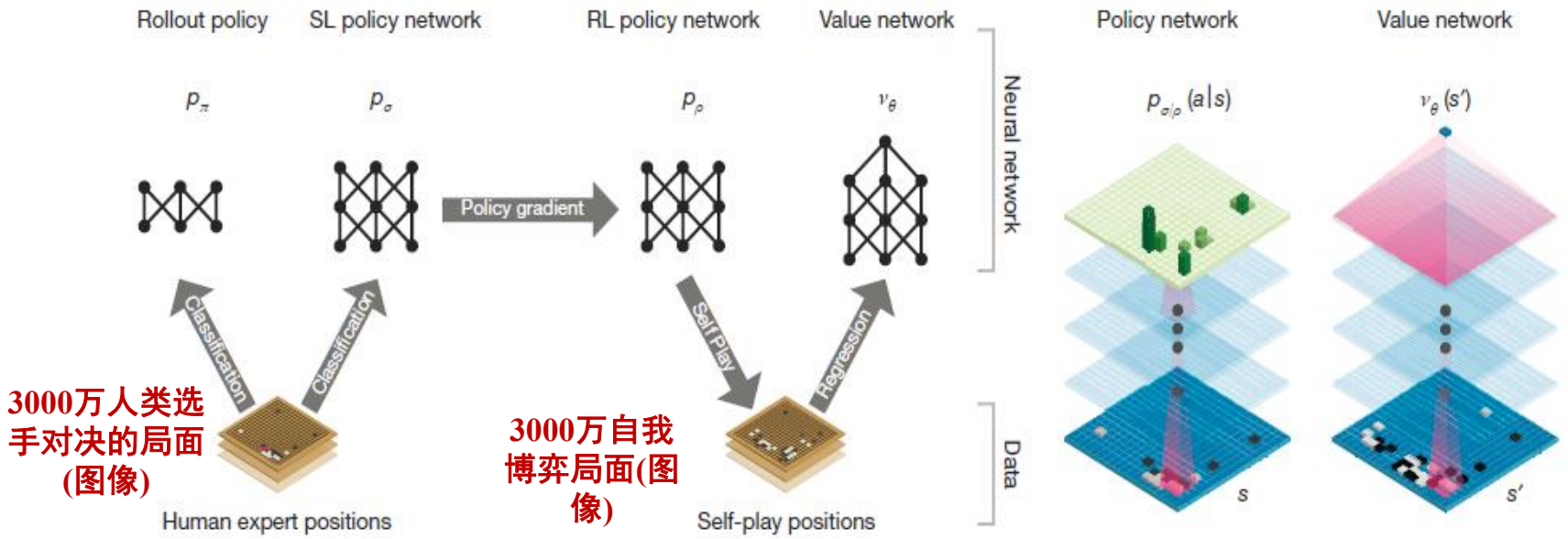


function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v

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'

AlphaGo算法解读

- 将每个状态(局面)均视为一幅图像
  - 训练策略(policy)网络和 价值(value)网络
  - $p_{\sigma,p}(a|s)$表示当前状态为s(局面)时，采取行动a后所得到的概率；$v_\theta(s')$表示当前状态为s'时，整盘棋获胜的概率。



D. Silver, et.al., Mastering the game of Go with Deep Neural Networks and Tree Search, *Nature*, 529:484-490,2016

AlphaGo算法解读：策略网络的训练

- **基于监督学习来先训练策略网络**
  - Idea: perform supervised learning (SL) to predict human moves
  - Given state , predict probability distribution over moves , $p_{\sigma,p}(a|s)$
  - Trained on 30M positions, 57% accuracy on predicting human moves
  - Also train a smaller, faster rollout policy network (24% accurate)
- **再基于强化学习来训练策略网络**
  - Idea: fine-tune policy network using reinforcement learning (RL)
  - Initialize RL network to SL network
  - Play two snapshots of the network against each other, update parameters to maximize expected final outcome
  - RL network wins against SL network 80% of the time, wins against open-source Pachi Go program 85% of the time

AlphaGo算法解读：价值网络的训练

- **Value network**
  - Idea: train network for position evaluation
  - Given state s', estimate $v_\theta(s')$, expected outcome of play starting with position s and following the learned policy for both players
  - Train network by minimizing mean squared error between actual and predicted outcome
  - Trained on 30M positions sampled from different self-play games

AlphaGo算法解读：策略网络和价值网络

- 在通过深度学习得到的策略网络和价值网络帮助之下，如下完成棋局局面的选择和搜索。给定节点v_0，将具有如下最大值的节点v选择作为v_0的后续节点

$$\frac{Q(v)}{N(v)}+\frac{P(v|v_0)}{1+N(v)}$$

  - 这里$P(v|v_0)$ 的值由策略网络计算得到。
  - 在模拟策略阶段(default policy)，AlphaGo不仅考虑仿真结果，而且考虑价值网络计算结果。
  - 策略网络和价值网络是离线训练得到的。

AlphaGo算法解读：策略网络和价值网络

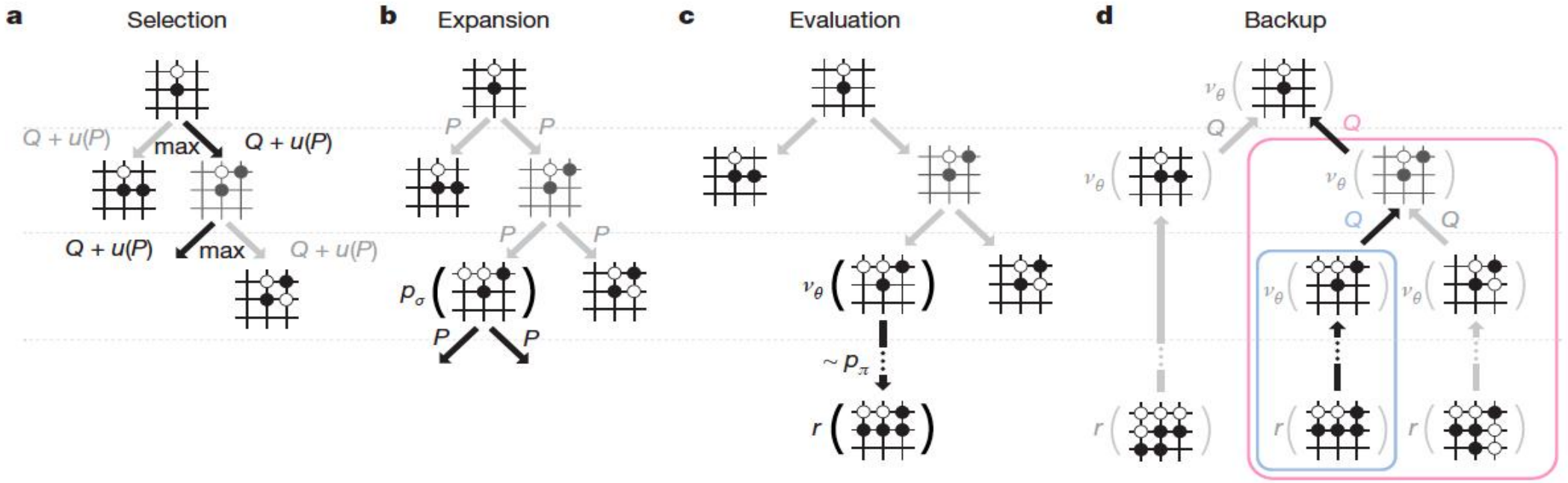


Figure 3 | Monte Carlo tree search in AlphaGo. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b**, The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c**, At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d**, Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t,a) + u(s_t,a))$$

$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i)$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) V(s_L^i)$$

价值网络计算

策略网络计算

$$u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

AlphaGo算法解读

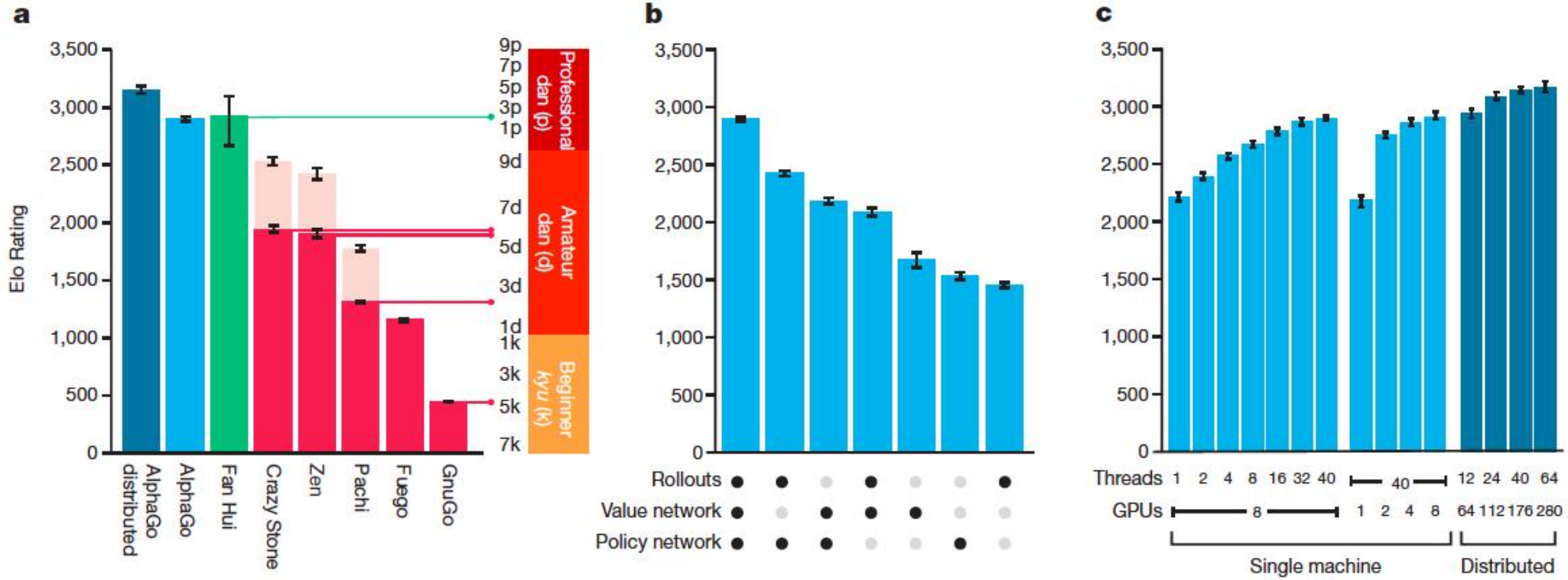
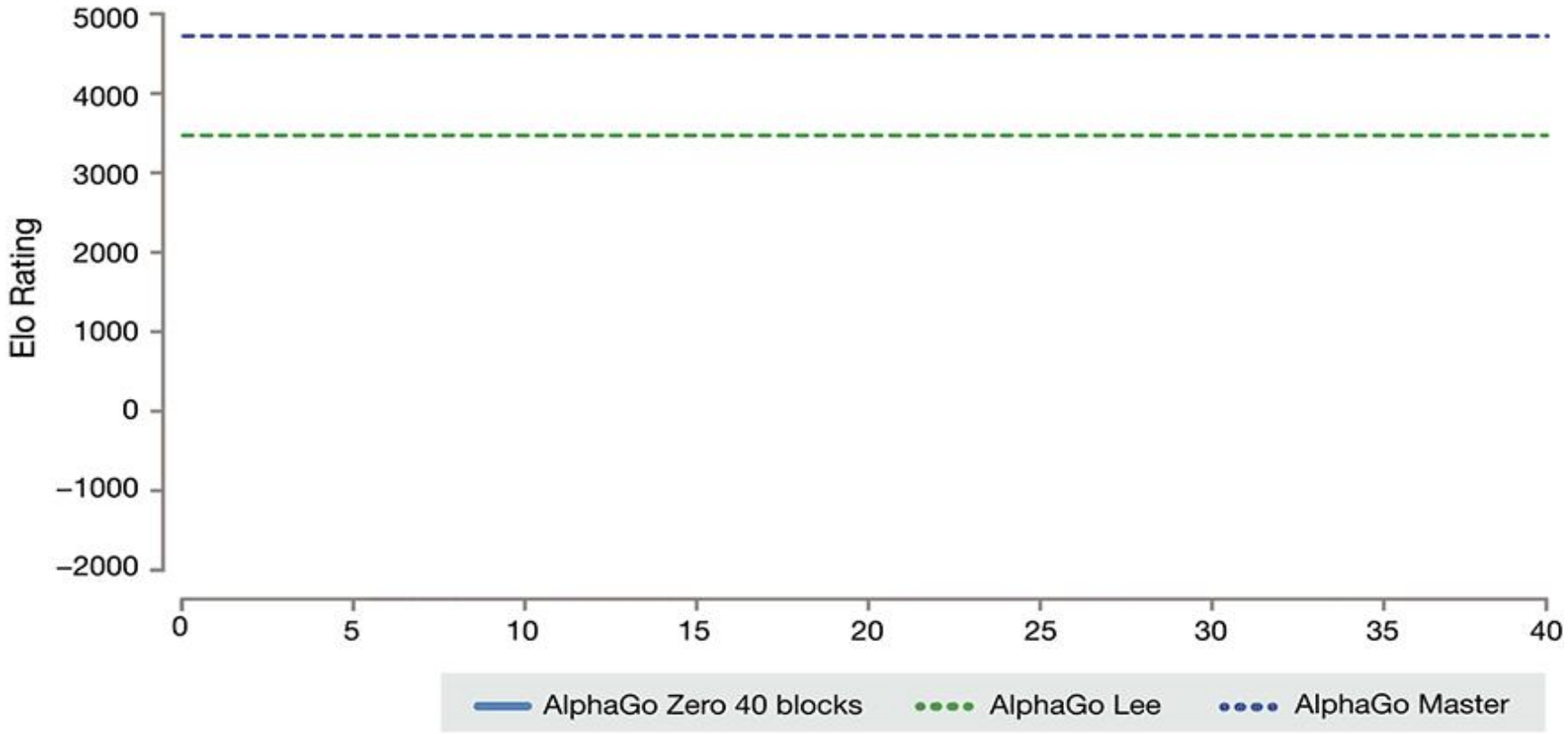


Figure 4 | Tournament evaluation of AlphaGo. a, Results of a tournament between different Go programs (see Extended Data Tables 6–11). Each program used approximately 5 s computation time per move. To provide a greater challenge to AlphaGo, some programs (pale upper bars) were given four handicap stones (that is, free moves at the start of every game) against all opponents. Programs were evaluated on an Elo scale[37]: a 230 point gap corresponds to a 79% probability of winning, which roughly corresponds to one amateur *dan* rank advantage on KGS[38]; an approximate correspondence to human ranks is also shown, horizontal lines show KGS ranks achieved online by that program. Games against the human European champion Fan Hui were also included; these games used longer time controls. 95% confidence intervals are shown. **b**, Performance of AlphaGo, on a single machine, for different combinations of components. The version solely using the policy network does not perform any search. **c**, Scalability study of MCTS in AlphaGo with search threads and GPUs, using asynchronous search (light blue) or distributed search (dark blue), for 2 s per move.

AlphaGo算法解读：AlphaGo Zero (一张白纸绘蓝图)

- 经过40天训练后，Zero总计运行约2900万次自我对弈，得以击败AlphaGo Master，比分为89比11



Mastering the game of Go without human knowledge, *Nature*, volume 550, pages 354–359, 2017

谢谢!