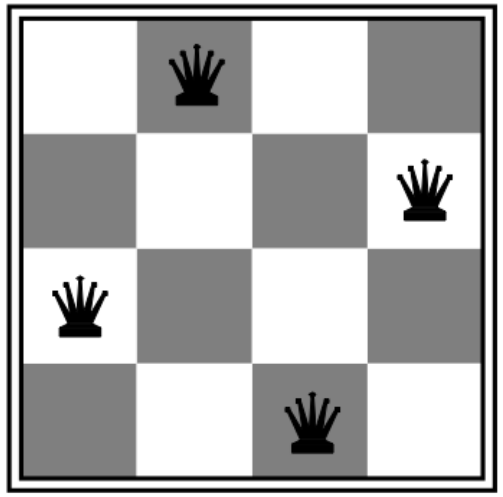
局部搜索（Local Search）

在很多问题中，我们只关心最终返回的状态是否达到目标，而不关心路径



N皇后问题

- 无需搜索从初始状态开始的各条路径
- 局部搜索：从一个状态开始，每次对当前状态领域（neighborhood）内的近邻解进行评价，并移动到其中一个近邻解，以获得越来越好的解

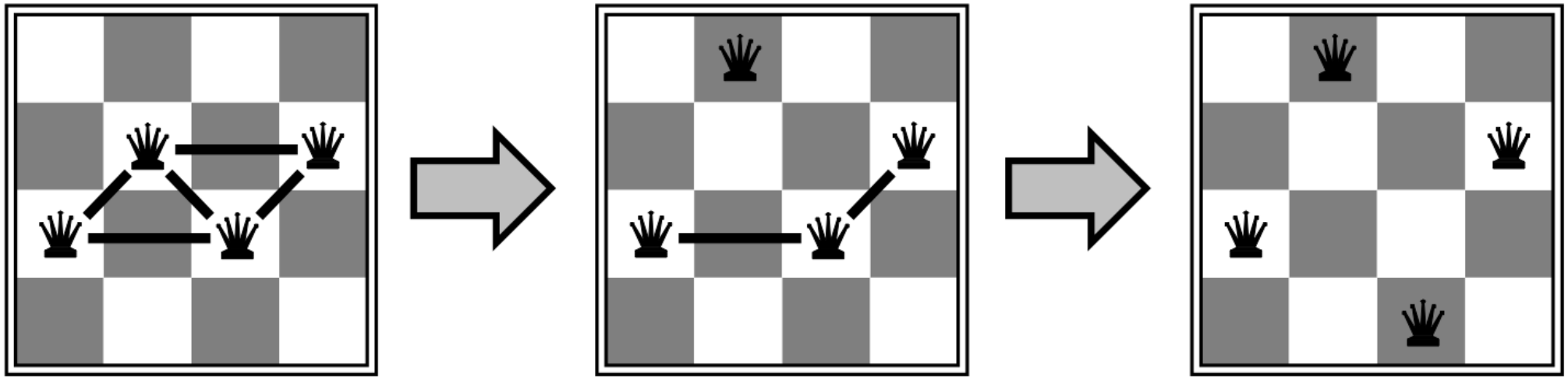
爬山法（Hill Climbing)

1. 从初始状态开始
2. Repeat: 移动到邻域内最好的解
3. 如果没有更好的解，则停止搜索

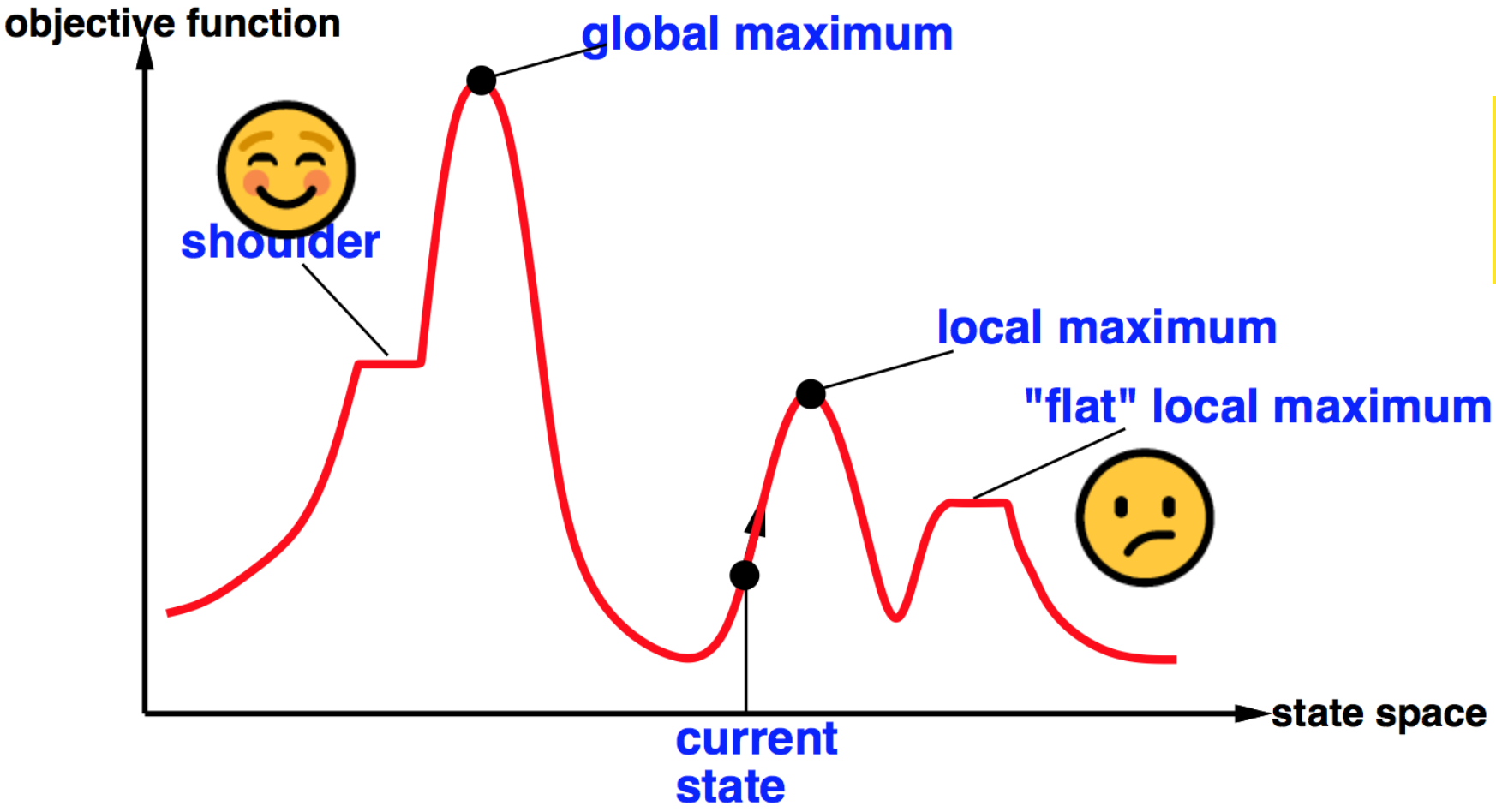
爬山法求解N皇后问题

目标：N个皇后之间没有冲突（不在同一行，同一列，对角线）

评估函数：冲突的个数



局部极大值



- 允许横向移动
- 随机重启(random-start)

模拟退火（simulated annealing）

- 爬山法：容易陷入最优解
- 随机游走(random walk)：不考虑状态值，随机移动到一个后继状态，最终总能找到全局极大值，但是效率非常低

能否结合一下？

在冶金学中，退火是一种通过将金属加热到高温然后逐渐冷却的方法，是材料到达低能量结晶态以进行硬化的过程

一开始，随机性强一点，后面，随机性弱一点

模拟退火（simulated annealing）

function SIMULATED-ANNEALING(problem, schedule) **returns** a state

current ← initial_state

for t = 1 **to** ∞ **do**

T ←schedule(t)

if T = 0 **then return** current

next ← a randomly selected successor of current

ΔE ← next.value – current.value

if ΔE > 0 **then** current ← next

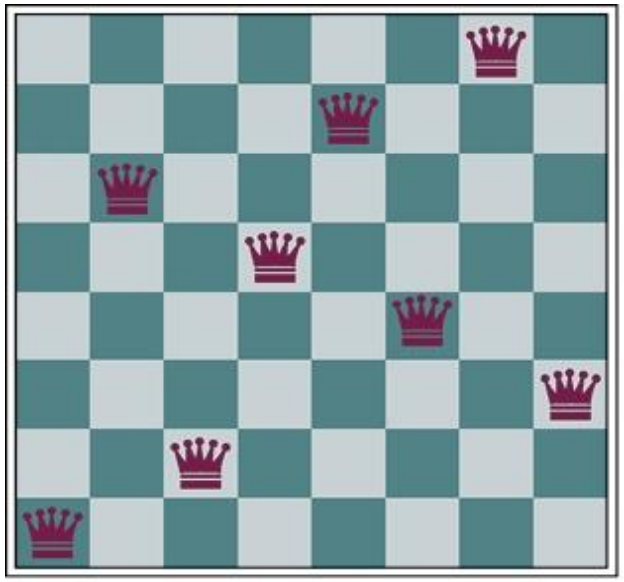
else current ← next only with probability $e^{\Delta E/T}$

演化算法

- 一个由个体（状态）组成的种群，其中最适应环境（值最高）的个体，可以生成后代（后继状态）来繁衍下一代
- 不断对种群进行选择、杂交、突变，一直到最优解
- 关键概念：
  - 种群规模、个体表示、选择、杂交、突变

演化算法

人类个体DNA序列可以用ACGT字符串表示，搜索问题中的个体如何表示？



16257483

演化算法

生成多个个体，模拟初始种群

24748552

32752411

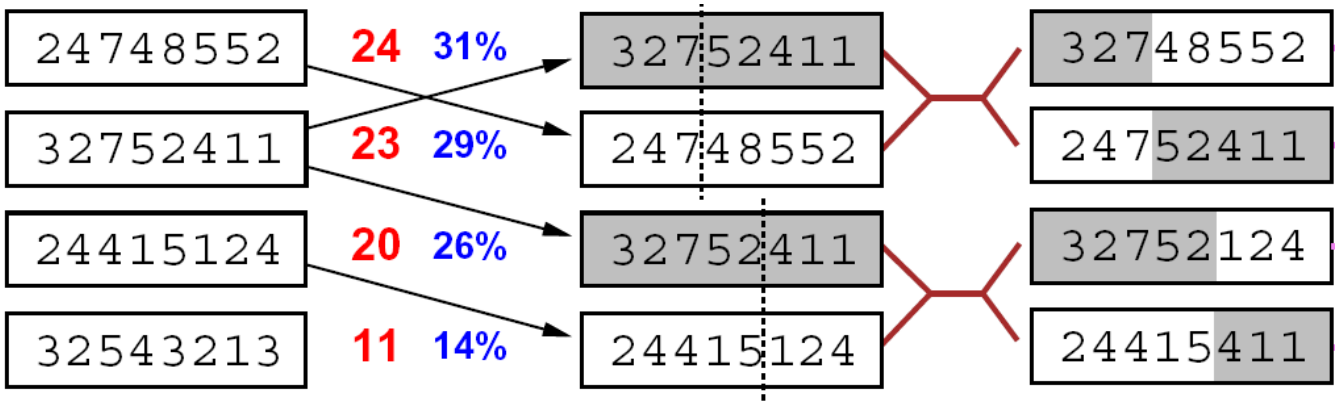
24415124

32543213

演化算法

选择成为下一代亲本的个体，被选中的概率与适应度成正比

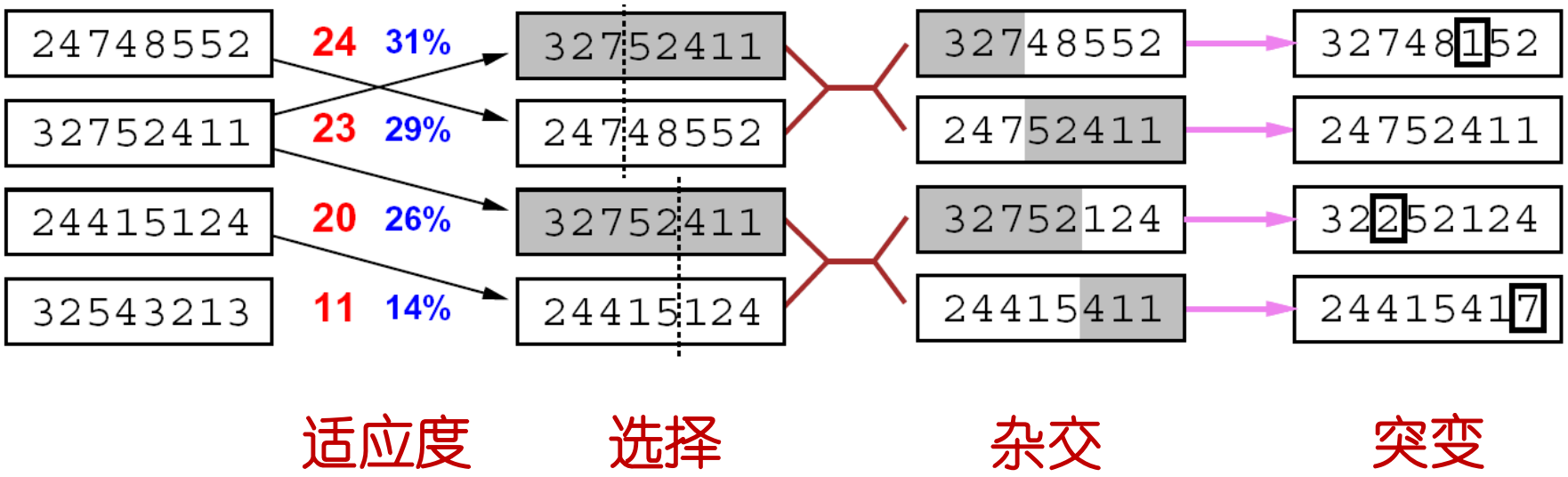
随机选择一个杂交点进行重组



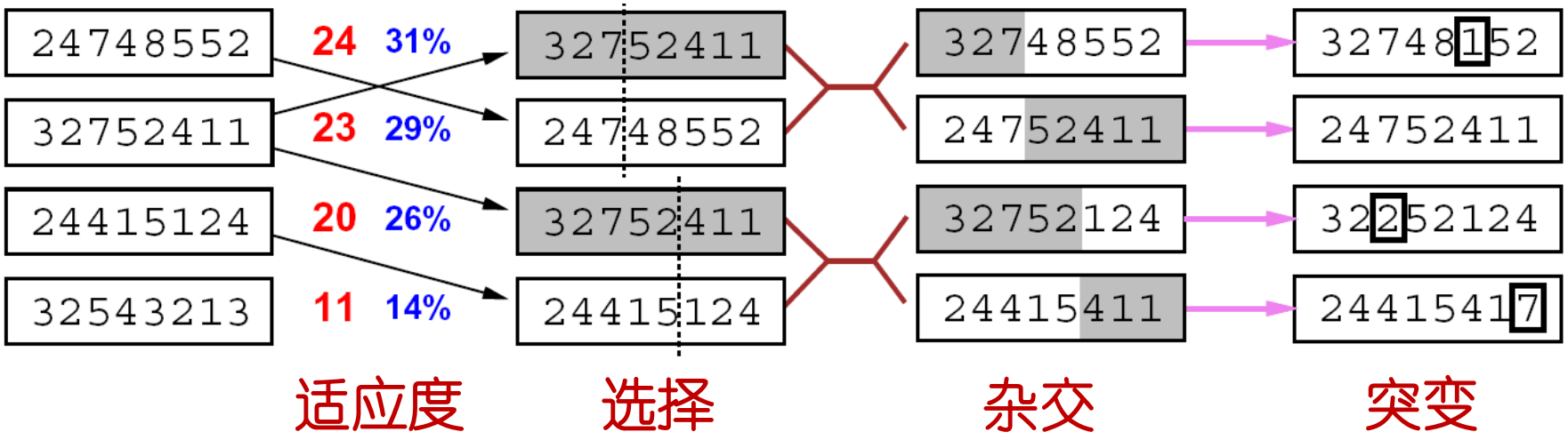
适应度越高的个体，产生后代的概率越大

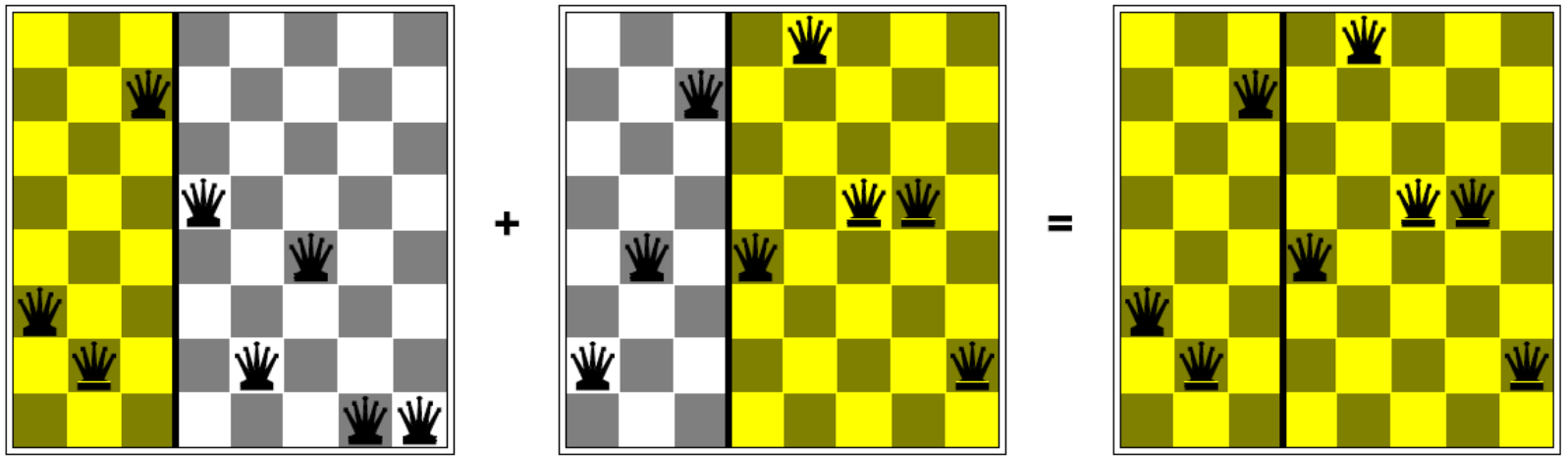
演化算法

后代在其表示上有发生随机突变的概率



演化算法

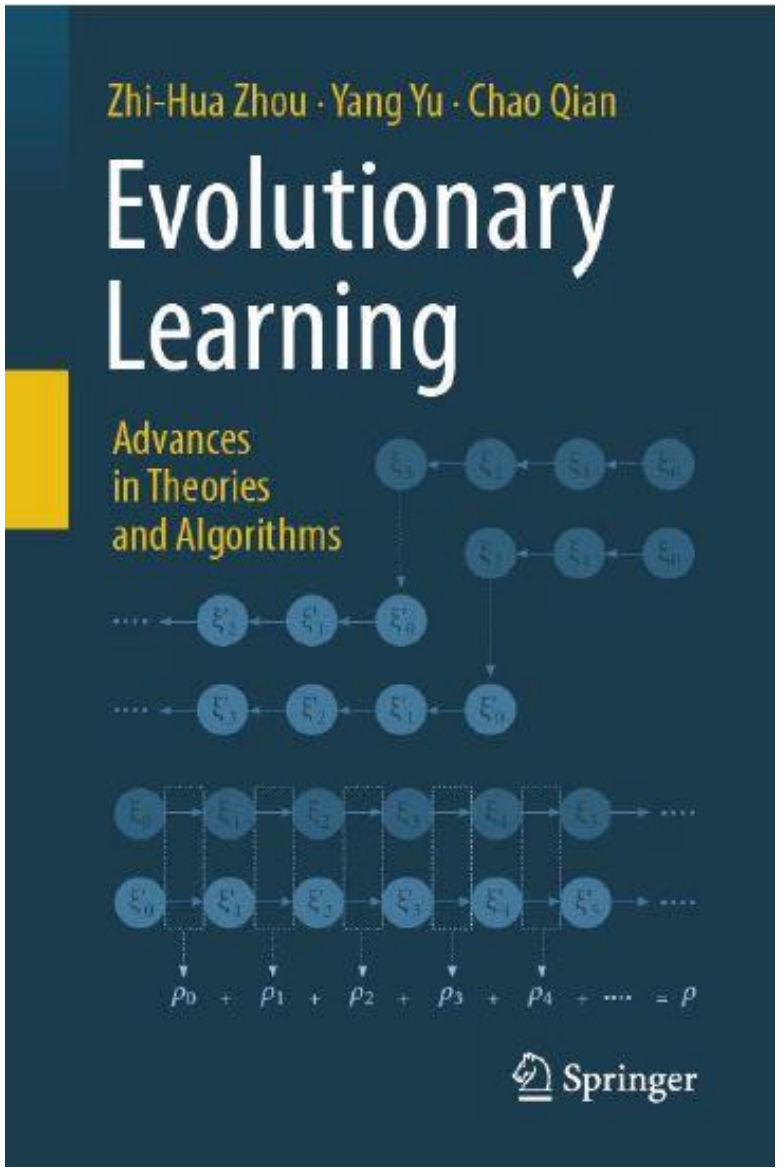

24748552
32752411
24415124
32543213
24 31%
23 29%
20 26%
11 14%
32752411
24748552
32752411
24415124
32748552
24752411
32752124
24415411
32748152
24752411
32252124
24415417
适应度
选择
杂交
突变


+
=

演化算法

演化算法的应用

- 电路布图
- 作业车间调度
- 神经网络架构搜索
- …



南京大学
NANJING UNIVERSITY

人工智能导论

对抗搜索

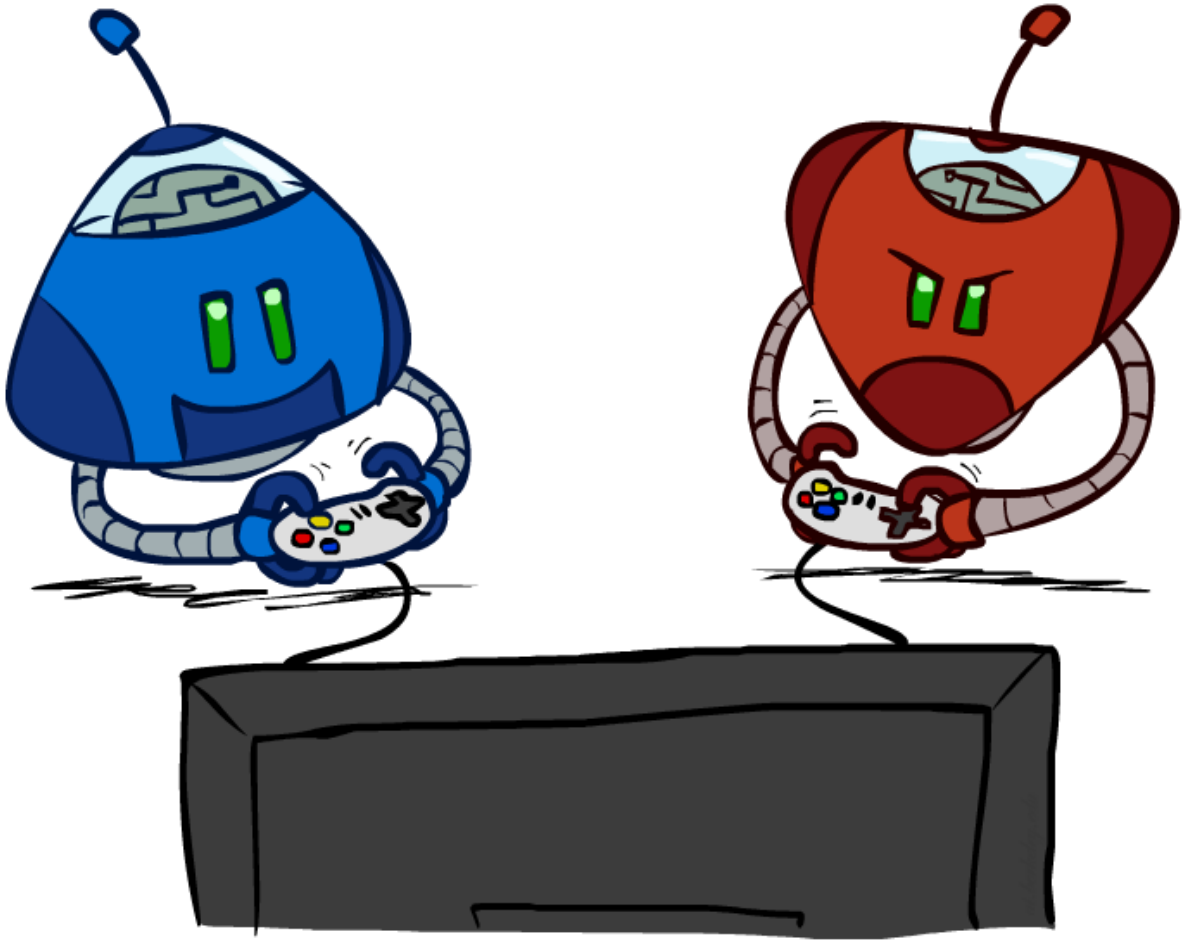
郭兰哲

南京大学 智能科学与技术学院

Homepage: www.lamda.nju.edu.cn/guolz

Email: guolz@nju.edu.cn

对抗搜索



提纲

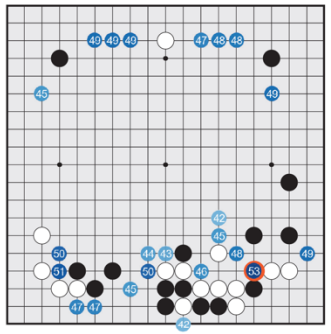
- 对抗博弈
  - 双人零和博弈
- 确定性搜索
  - 最大最小搜索
  - Alpha-beta 剪枝
- 基于模拟的搜索
  - 蒙特卡洛树搜索

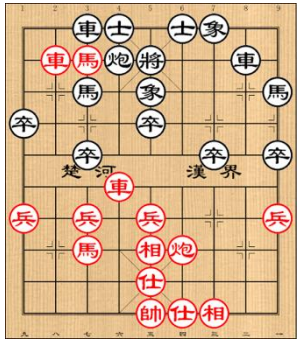
提纲

- 对抗博弈
  - 双人零和博弈
- 确定性搜索
  - 最大最小搜索
  - Alpha-beta 剪枝
- 基于模拟的搜索
  - 蒙特卡洛树搜索

博弈

对抗搜索（Adversarial Search）也称为博弈搜索（Game Search）

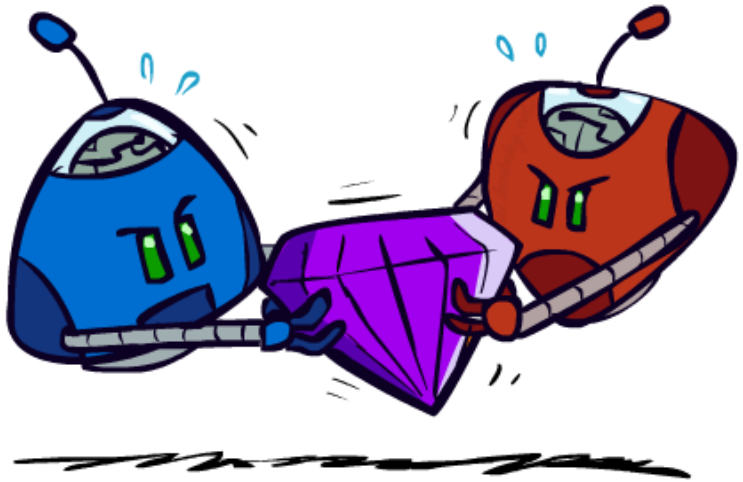




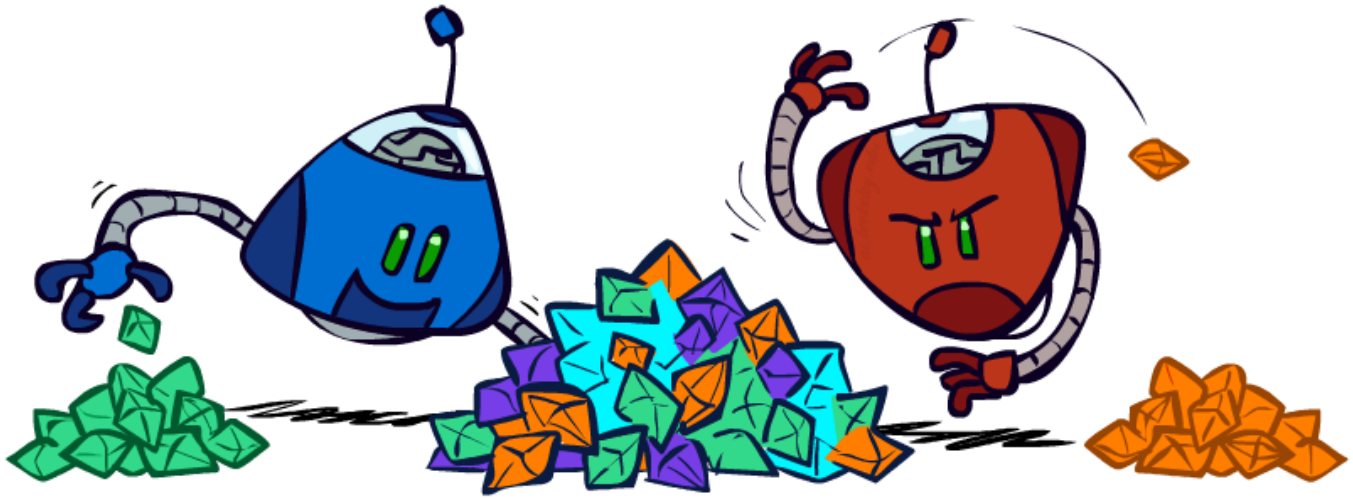
博弈的种类：

- ✓ 确定的、有随机性的
- ✓ 是否有完整信息？
- ✓ 几个玩家？
- ✓ 是不是零和博弈？

零和博弈



一个玩家赢了，则对手一定输了



你可能赚了，但我也不亏

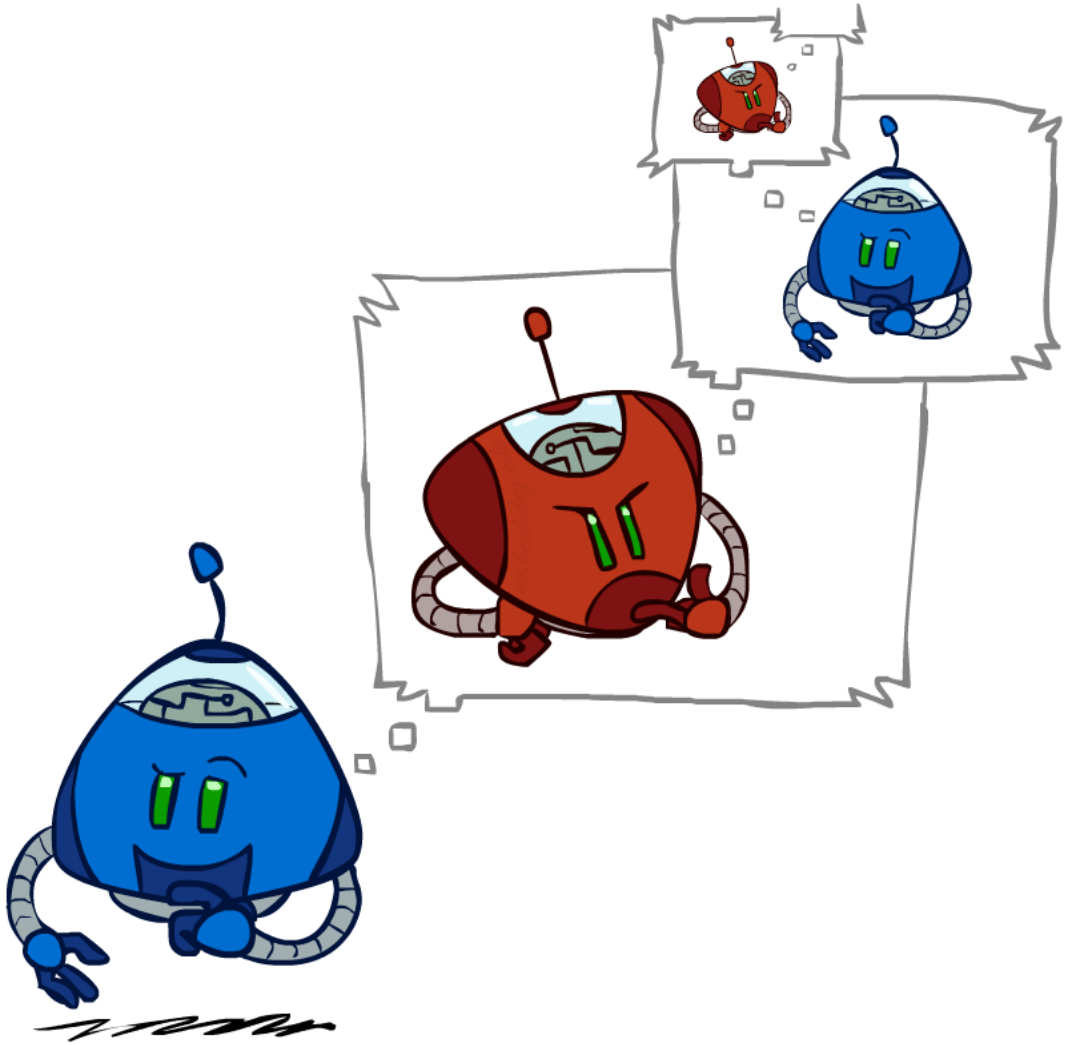
双人零和博弈

我们考虑信息确定、全局可观察、竞争对手轮流行动、输赢

收益零和假设下的双人博弈问题

VS.

对抗搜索



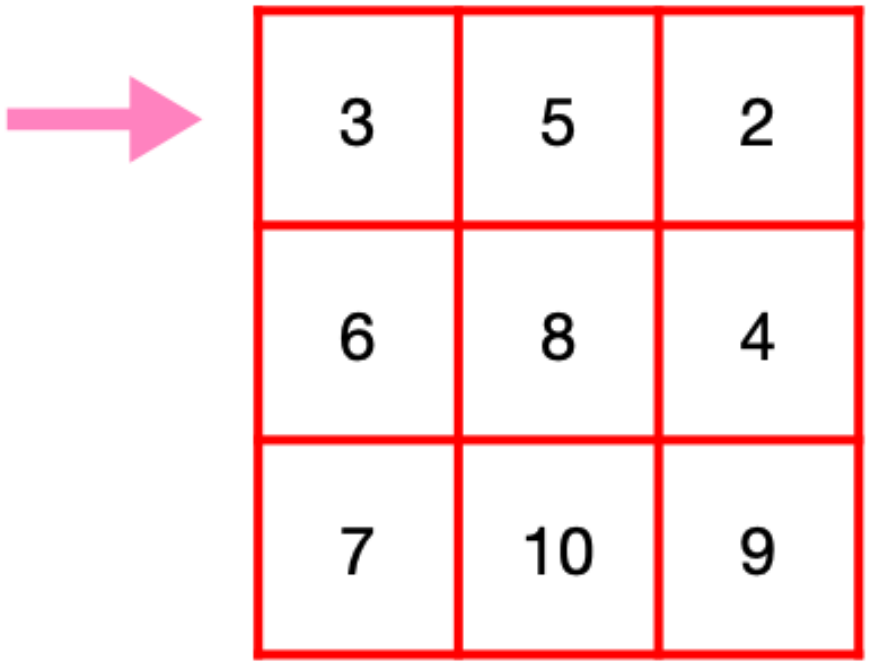
Two-Step Game

- Two step game: 首先，Alice选择第 i 行，然后，Bob选择第 j 列
- 结果：Alice输（Bob赢）第 i 行第 j 列的元素值

3	5	2
6	8	4
7	10	9

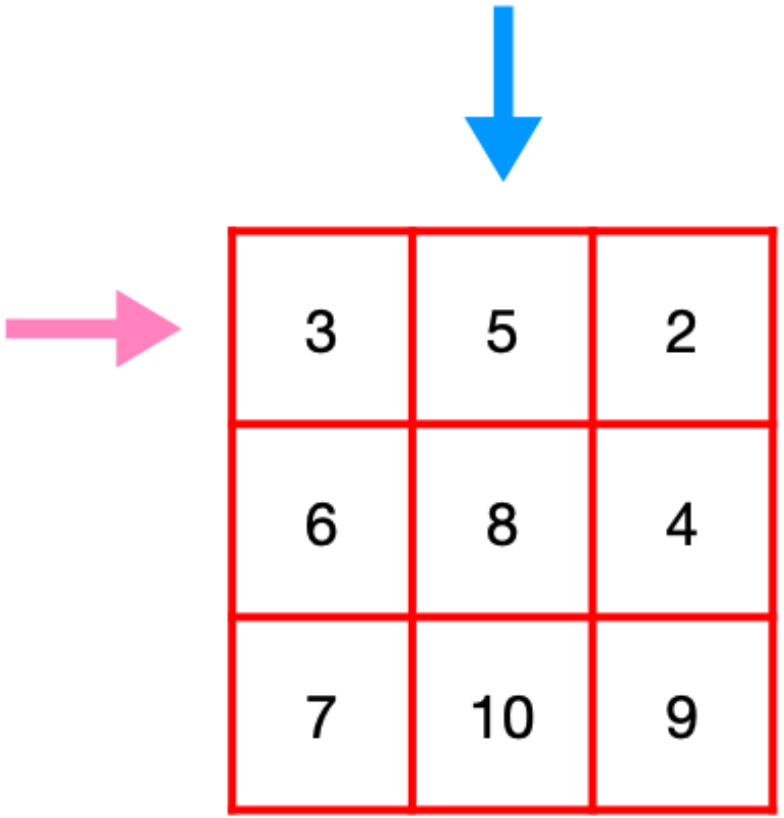
Two-Step Game

- Two step game: 首先，Alice选择第 i 行，然后，Bob选择第 j 列
- 结果：Alice输（Bob赢）第 i 行第 j 列的元素值



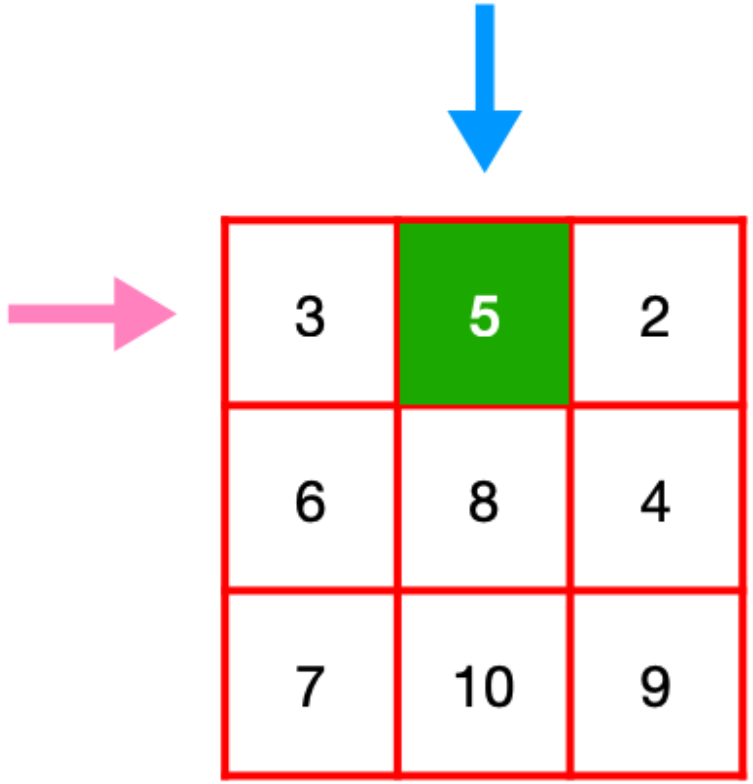
Two-Step Game

- Two step game: 首先，Alice选择第 i 行，然后，Bob选择第 j 列
- 结果：Alice输（Bob赢）第 i 行第 j 列的元素值



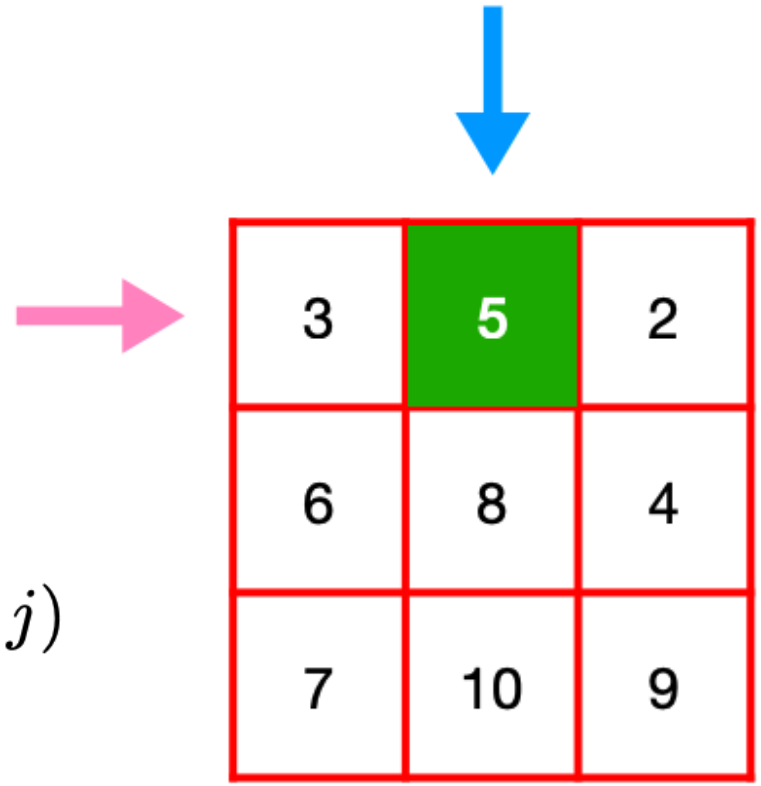
Two-Step Game

- Two step game: 首先，Alice选择第 i 行，然后，Bob选择第 j 列
- 结果：Alice输（Bob赢）第 i 行第 j 列的元素值



Two-Step Game

- Two step game: 首先，Alice选择第 i 行，然后，Bob选择第 j 列
- 结果：Alice输（Bob赢）第 i 行第 j 列的元素值
- A MinMax Game



$$U(i^*, j^*) = \min_i \max_j U(i, j)$$

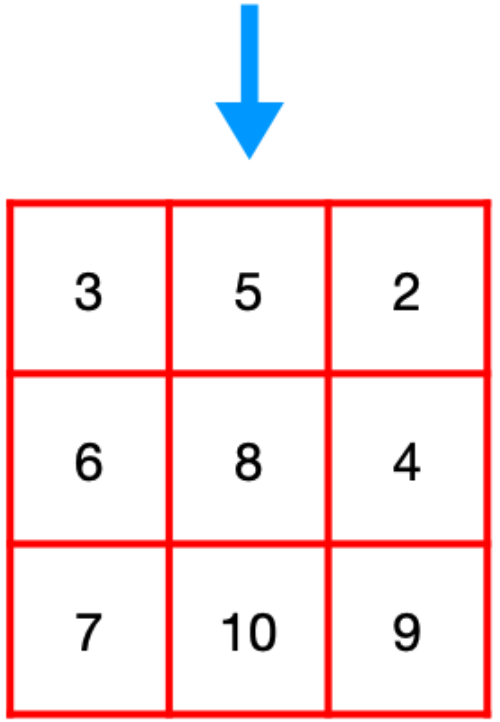
Two-Step Game

- Two step game: 首先，Bob选择第 j 列，然后，Alice选择第 i 行
- 结果：Alice输（Bob赢）第 i 行第 j 列的元素值

3	5	2
6	8	4
7	10	9

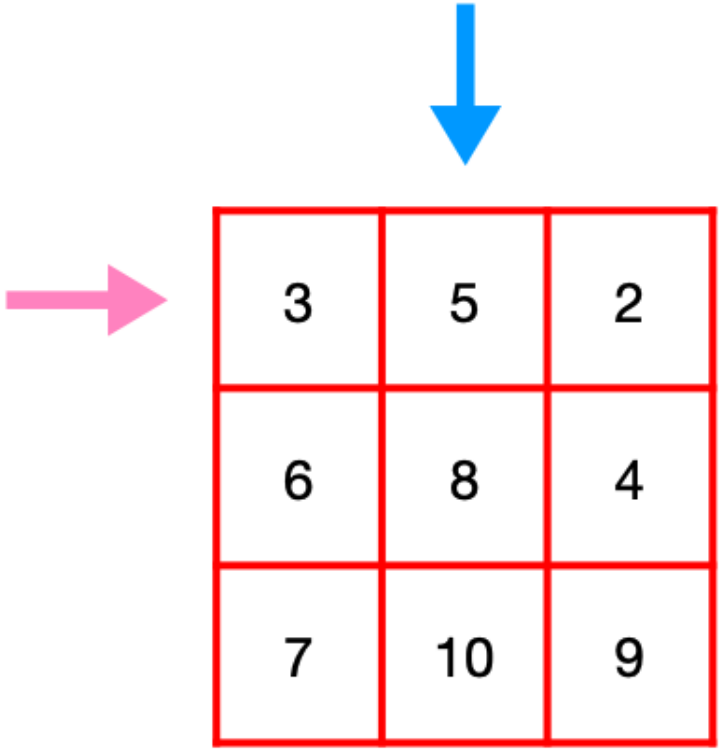
Two-Step Game

- Two step game: 首先，**Bob**选择第 j 列，然后，**Alice**选择第 i 行
- 结果：**Alice**输（**Bob**赢）第 i 行第 j 列的元素值



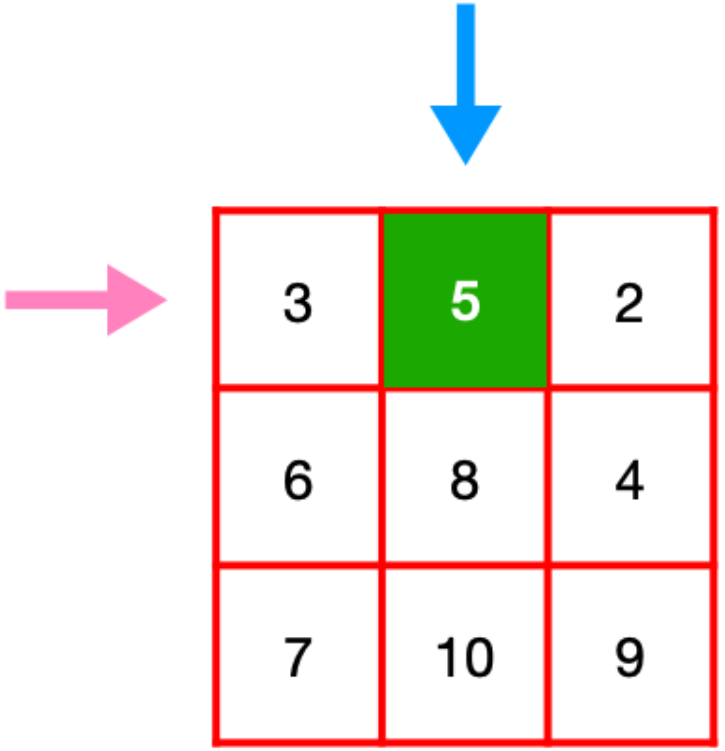
Two-Step Game

- Two step game: 首先，**Bob**选择第 j 列，然后，**Alice**选择第 i 行
- 结果：**Alice**输（**Bob**赢）第 i 行第 j 列的元素值



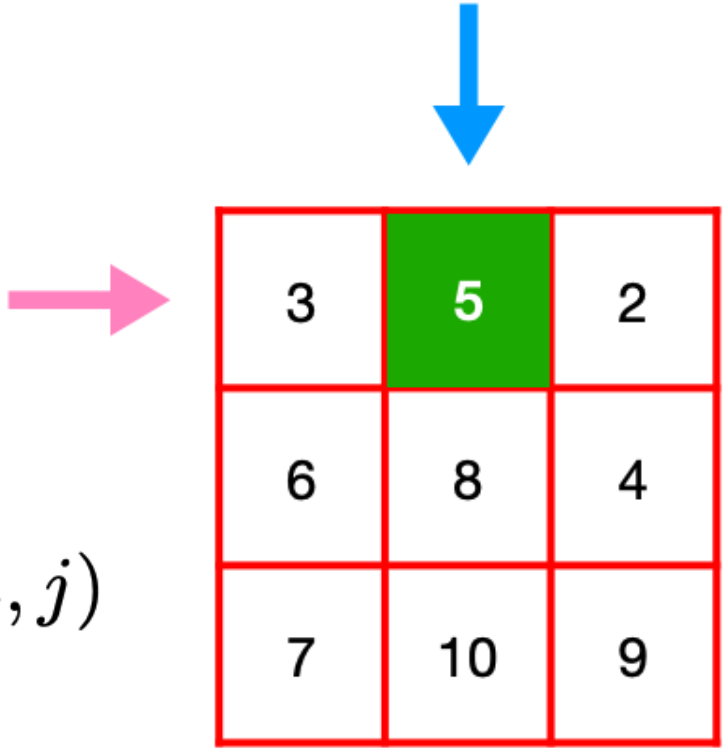
Two-Step Game

- Two step game: 首先，Bob选择第 j 列，然后，Alice选择第 i 行
- 结果：Alice输（Bob赢）第 i 行第 j 列的元素值



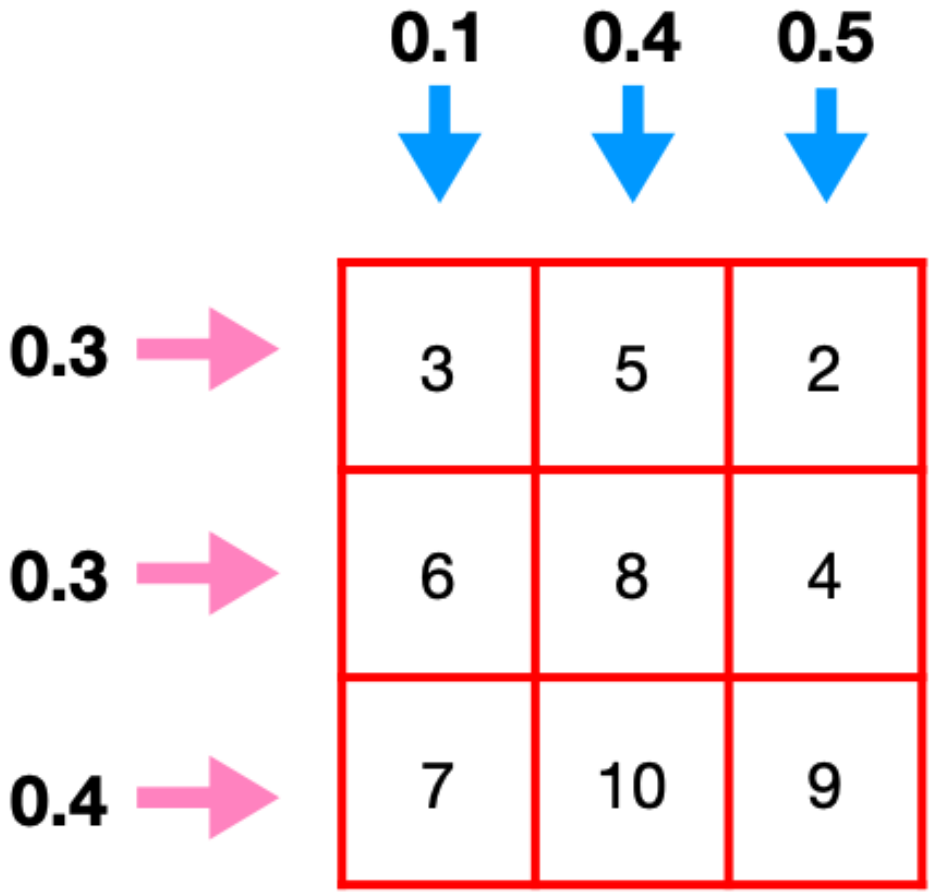
Two-Step Game

- Two step game: 首先，Bob选择第 j 列，然后， Alice选择第 i 行
- 结果：Alice输（Bob赢）第 i 行第 j 列的元素值
- A MaxMin Game



$$U(i^*, j^*) = \max_j \min_i U(i, j)$$

确定性策略和非确定性策略



提纲

- 对抗博弈
  - 双人零和博弈
- 确定性搜索
  - 最大最小搜索
  - Alpha-beta 剪枝
- 基于模拟的搜索
  - 蒙特卡洛树搜索

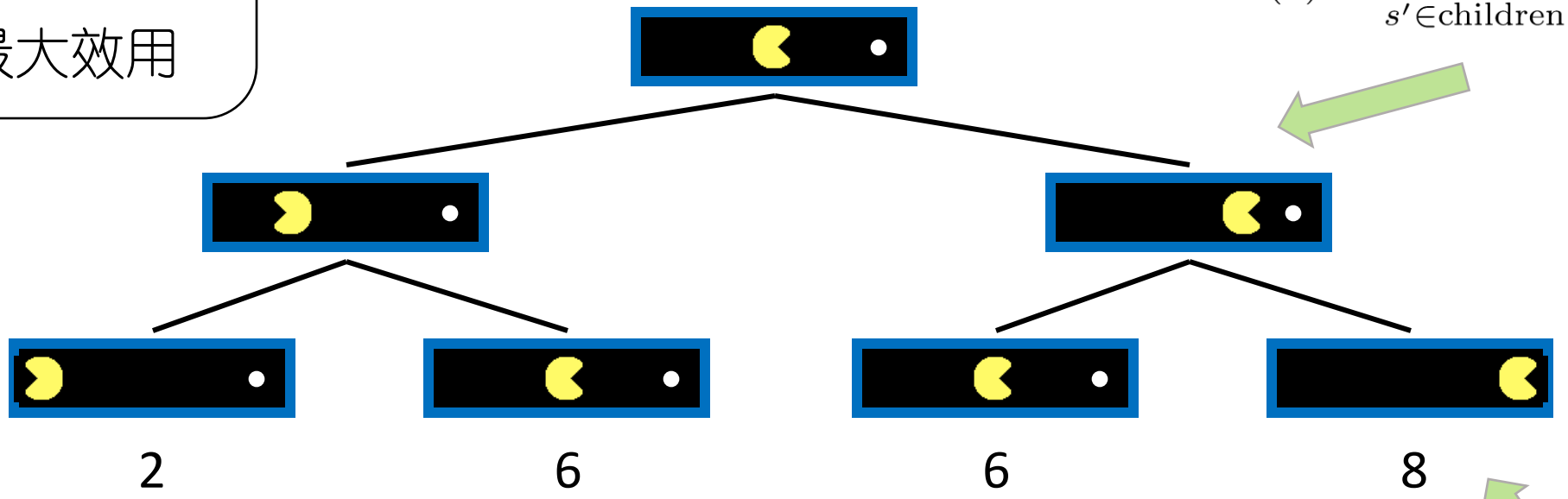
单一Agent搜索树

状态的价值 V

从当前状态出发能获得的最大效用

Non-Terminal States:

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



Terminal States:

$$V(s) = \text{known}$$

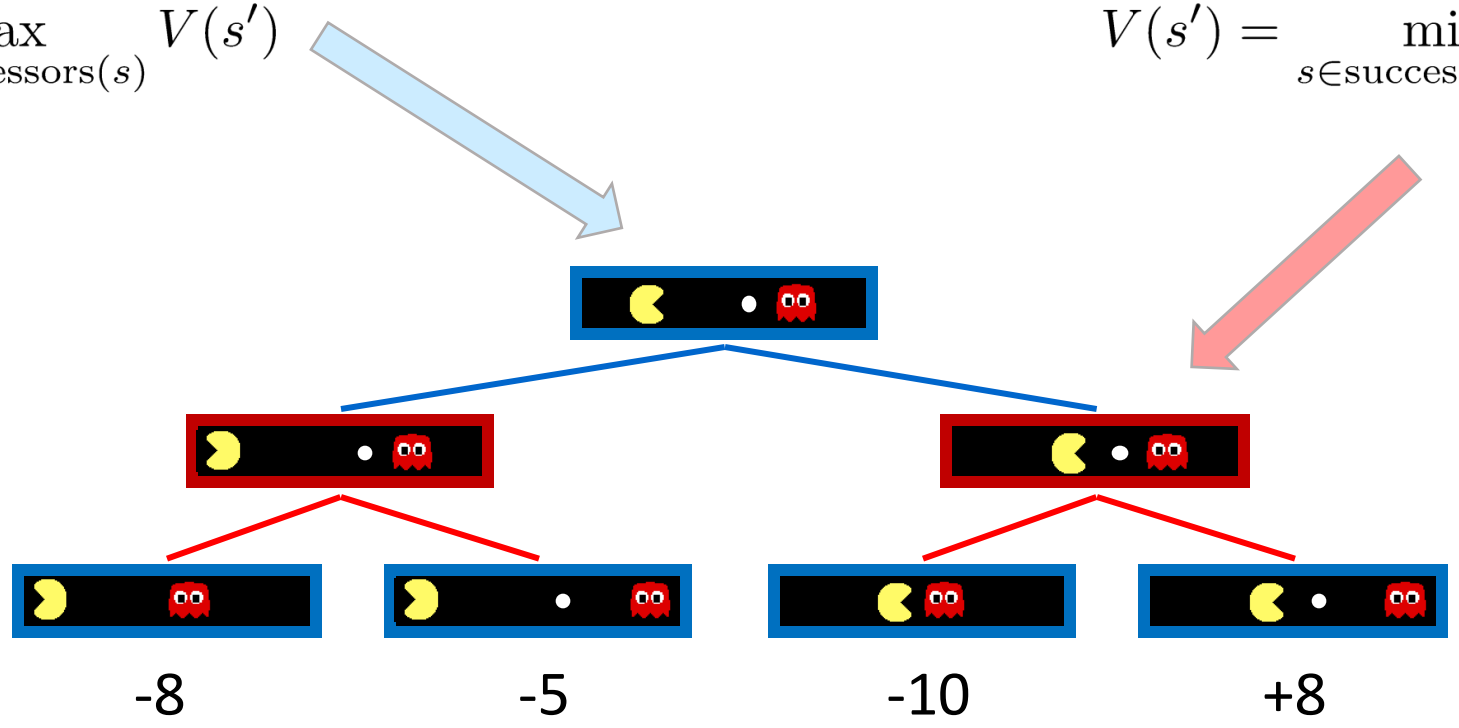
博弈搜索树

States Under Agent's Control:

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

States Under Opponent's Control:

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

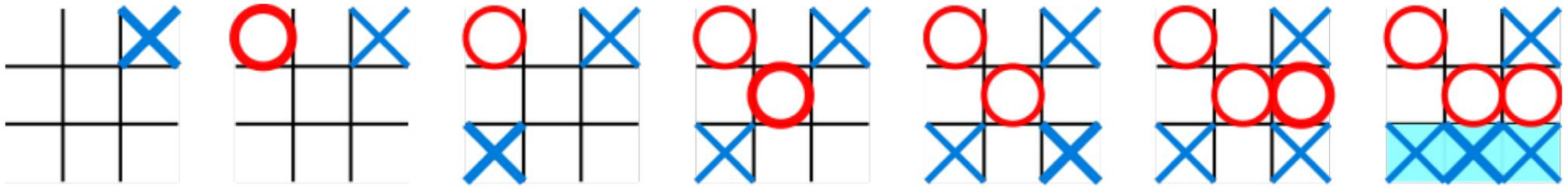


Terminal States:

$$V(s) = \text{known}$$

多步搜索

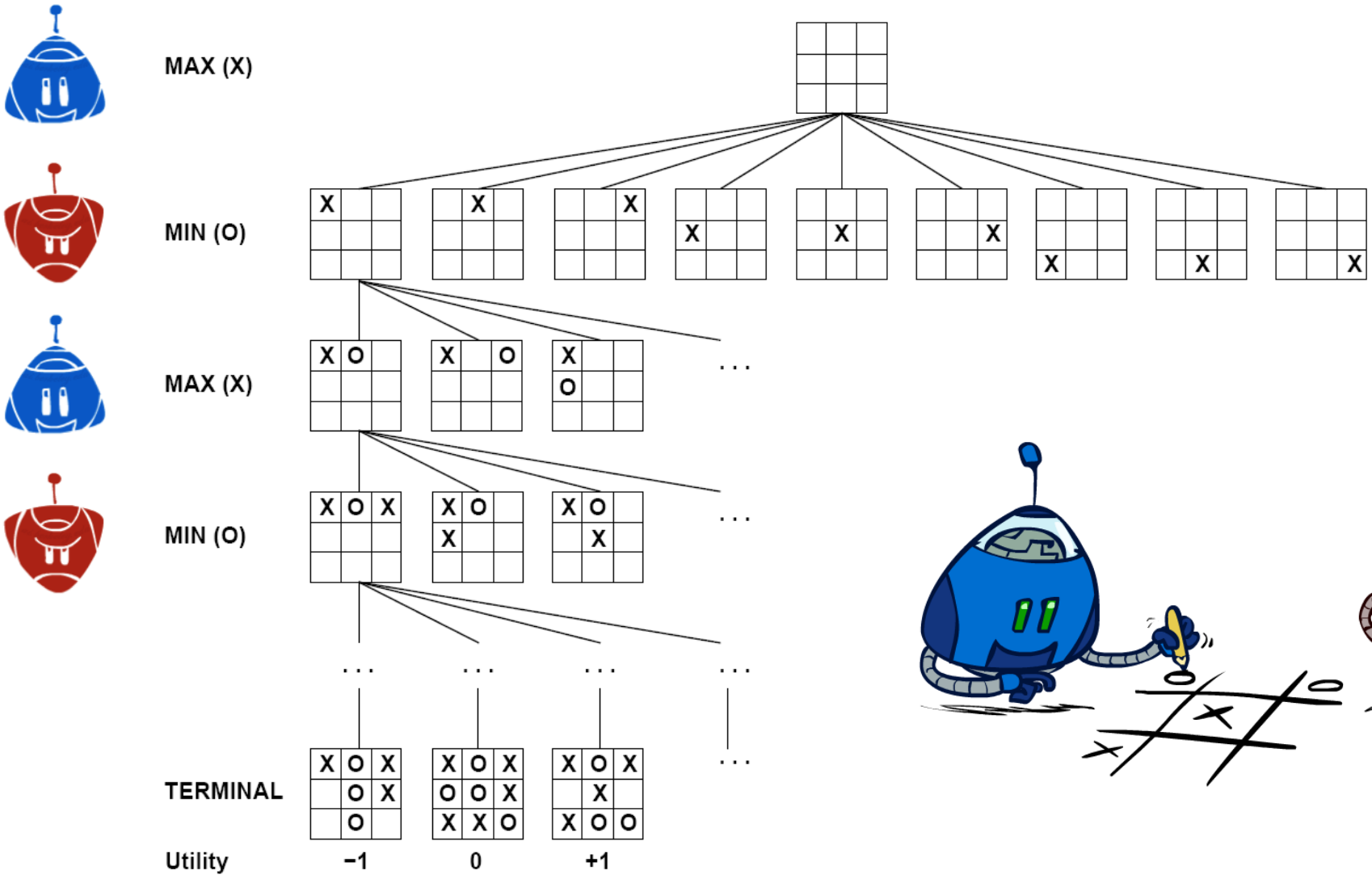
两人轮流在一有九格方盘上划加字或圆圈，谁先把三个同一记号排成横线、直线、斜线，即是胜者



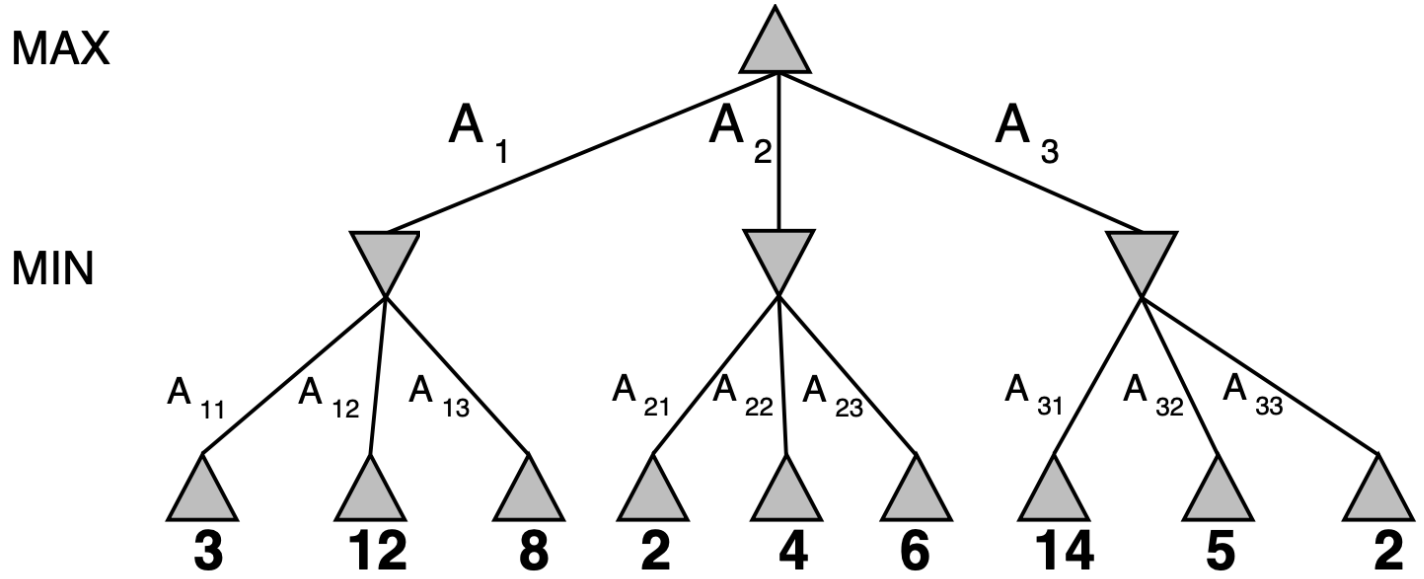
问题定义

- **状态**：状态s包括当前的游戏局面和当前行动的玩家
- **动作**：给定状态s，动作指的是$player(s)$在当前局面下可以采取的操作a，记动作集合为$actions(s)$
- **状态转移**：给定状态s和动作$a \in actions(s)$，状态转移函数$result(s,a)$决定了在s状态采取a动作后所得后继状态
- **终局状态检测**：终止状态检测函数$terminal_test(s)$用于测试游戏是否在状态s结束
- **终局得分**：终局得分$utility(s,p)$表示在终局状态s时玩家p的得分

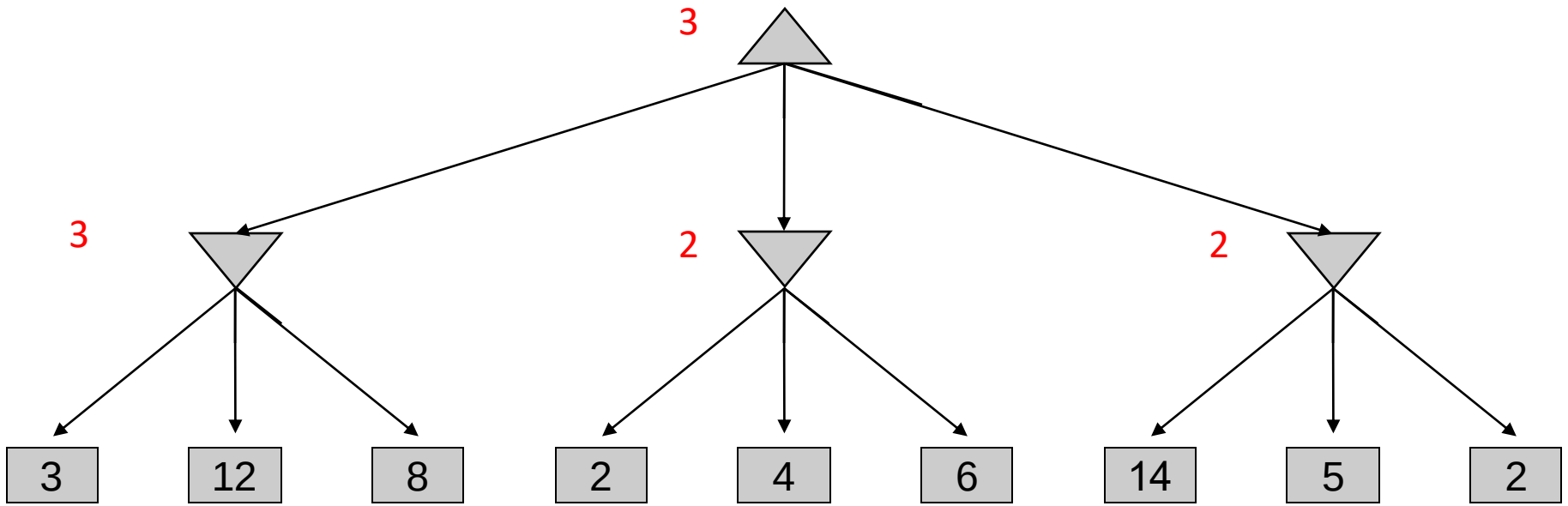
搜索树



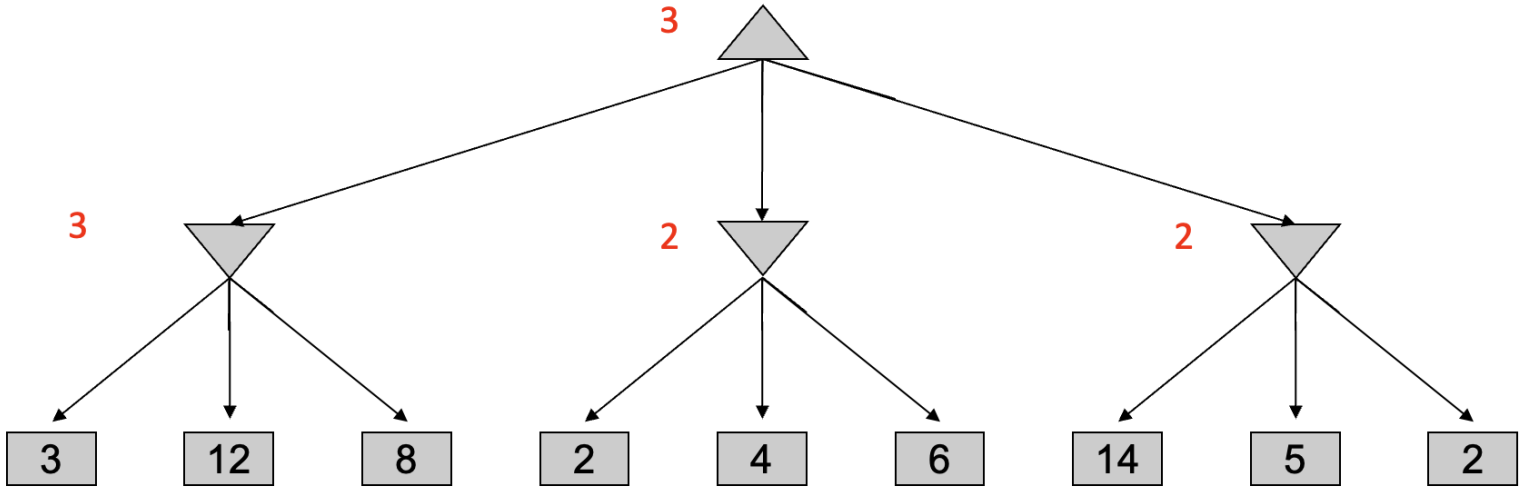
最优策略


MAX
MIN
A 1
A 2
A 3
A 11
A 12
A 13
A 21
A 22
A 23
A 31
A 32
A 33
3
12
8
2
4
6
14
5
2

最优策略



最优策略

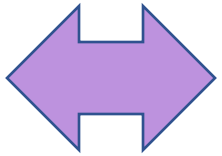


给定一棵博弈树，最优策略可以通过检查每个节点的极小极大值来决定，记为minimax(n)

$$\text{minimax}(s) = \begin{cases} \text{utility}(s), & \text{if terminal\_test}(s) \\ \max_{a \in \text{actions}(s)} \text{minimax}(\text{result}(s,a)), & \text{if player}(s) = MAX \\ \min_{a \in \text{actions}(s)} \text{minimax}(\text{result}(s,a)), & \text{if player}(s) = MIN \end{cases}$$

最小最大搜索

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v
```



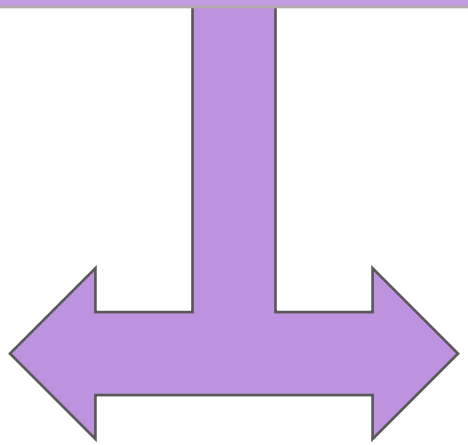
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v
```

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

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

最小最大搜索

```
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)
```

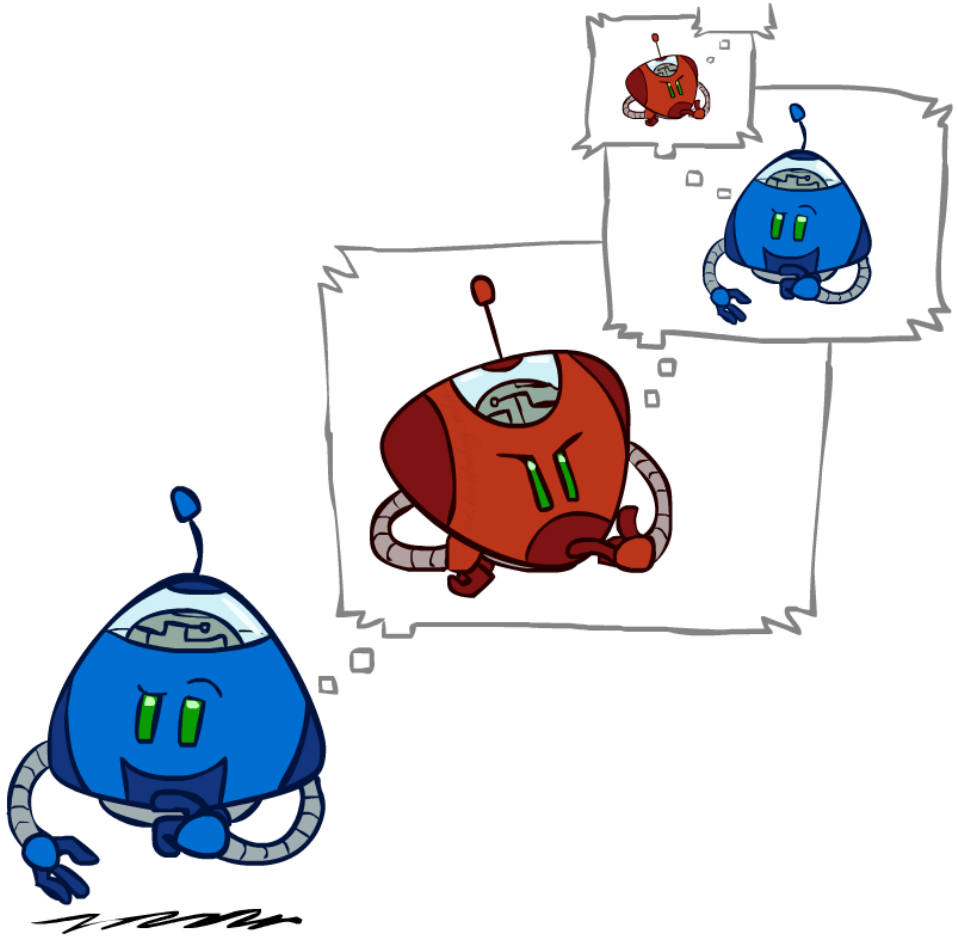


```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

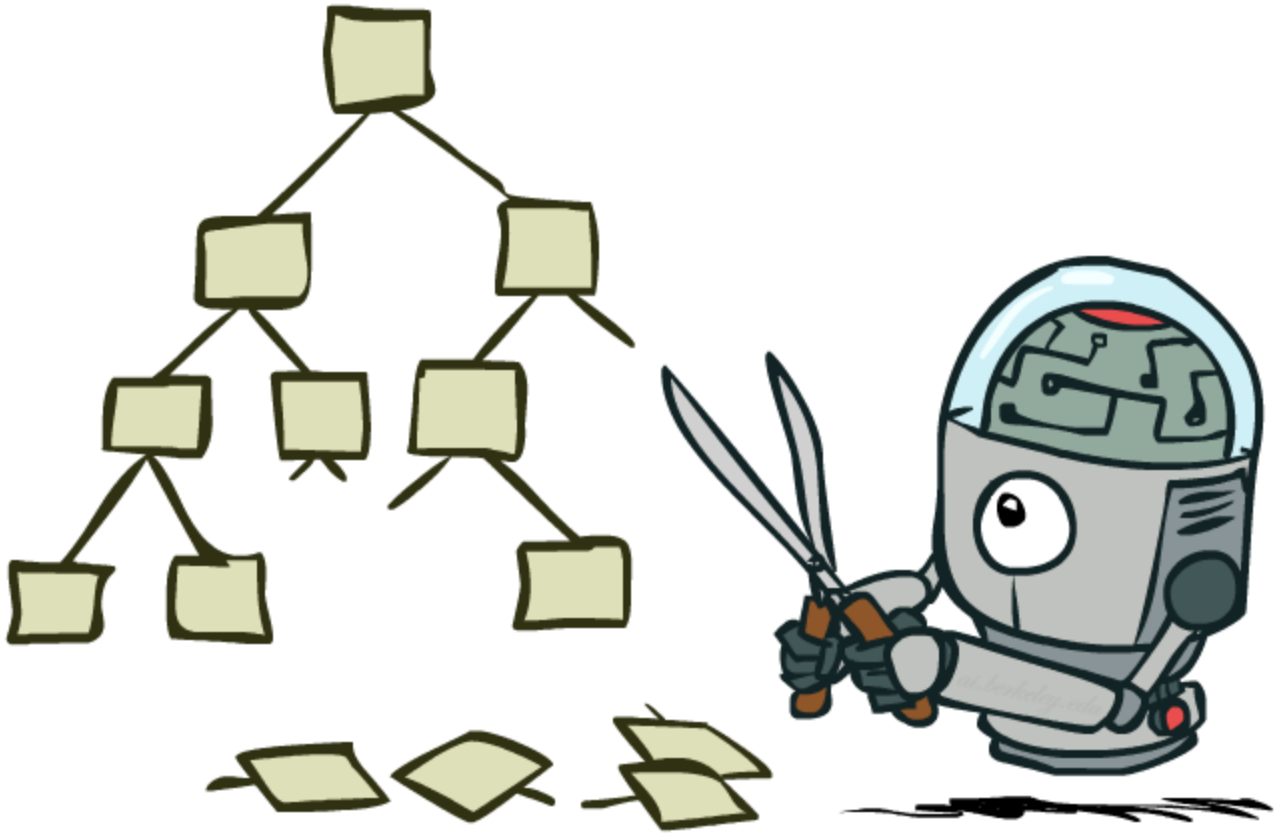
性能分析

- 时间和空间?
  - 和 DFS 类似
  - 时间复杂度: $O(b^m)$
  - 空间复杂度: O(bm)



- Example: For chess, $b \approx 35$, $m \approx 100$
  - 精确的搜索几乎是不可行的

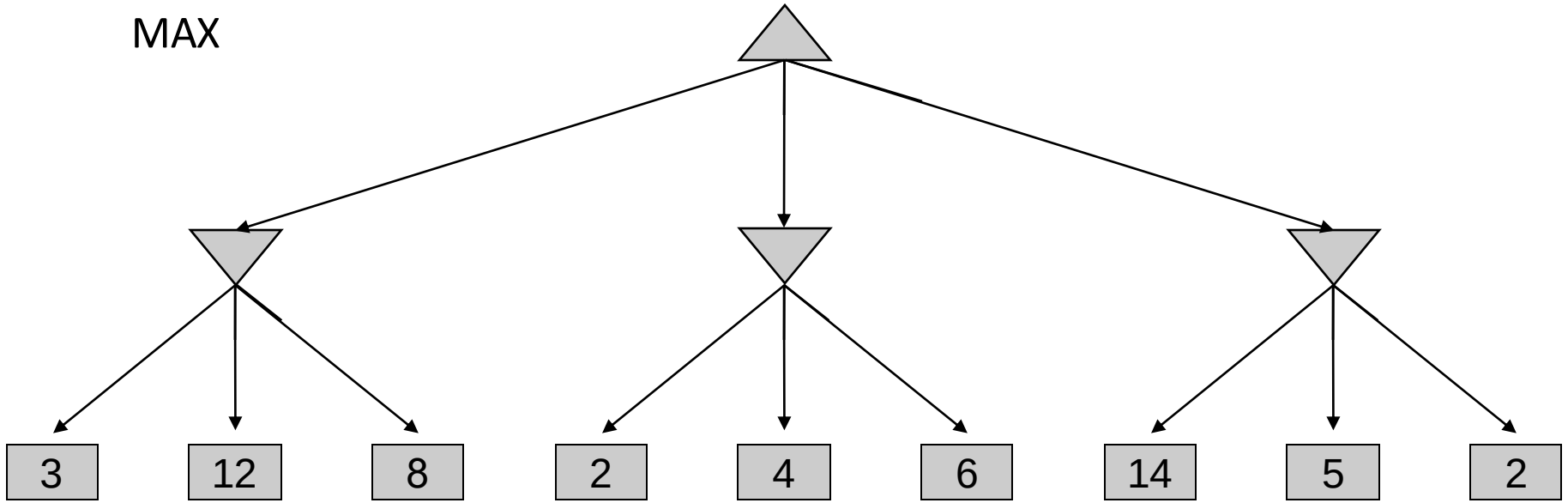
树剪枝



提纲

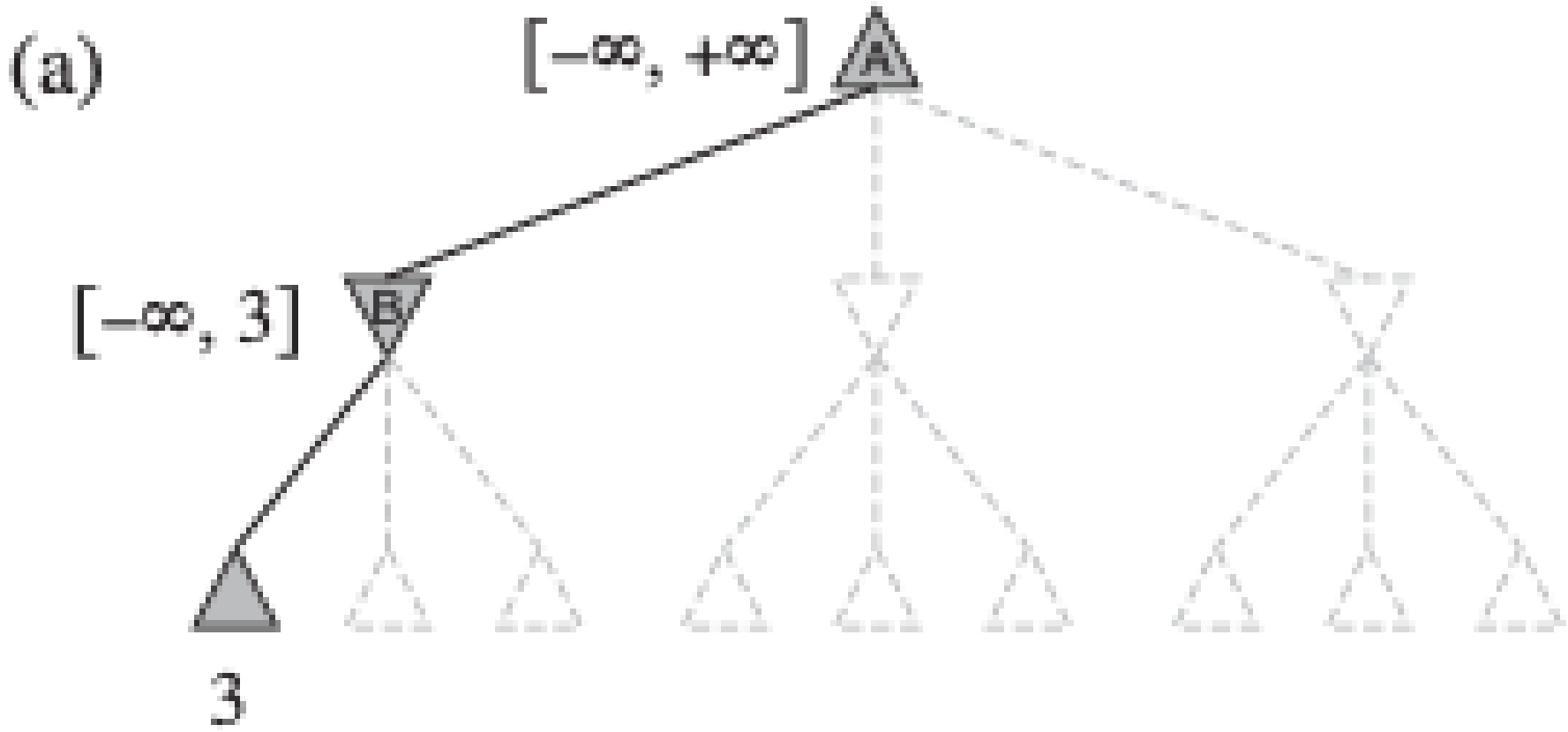
- 对抗博弈
  - 双人零和博弈
- 确定性搜索
  - 最大最小搜索
  - Alpha-beta 剪枝
- 基于模拟的搜索
  - 蒙特卡洛树搜索

树剪枝



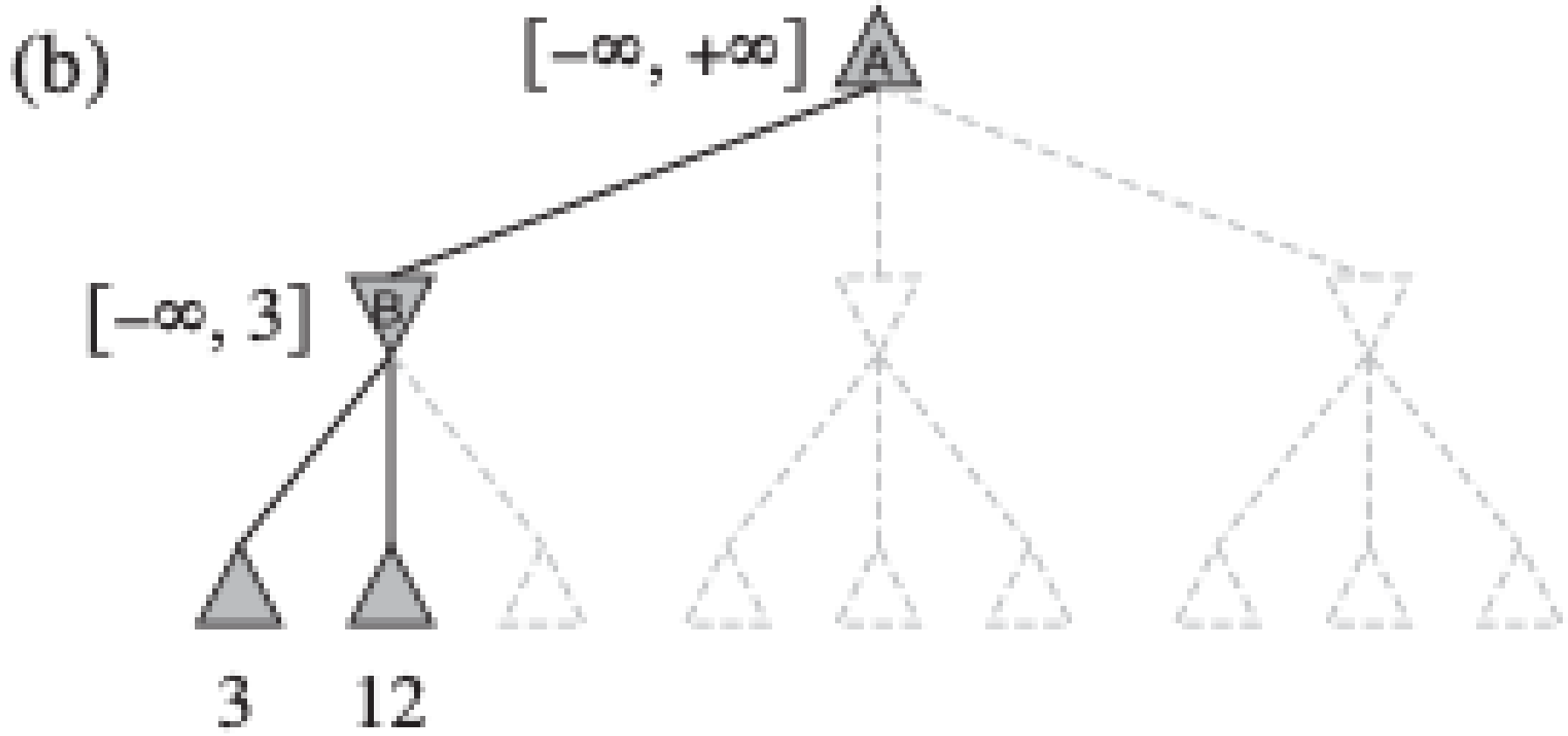
树剪枝

Vector:[alpha, beta]



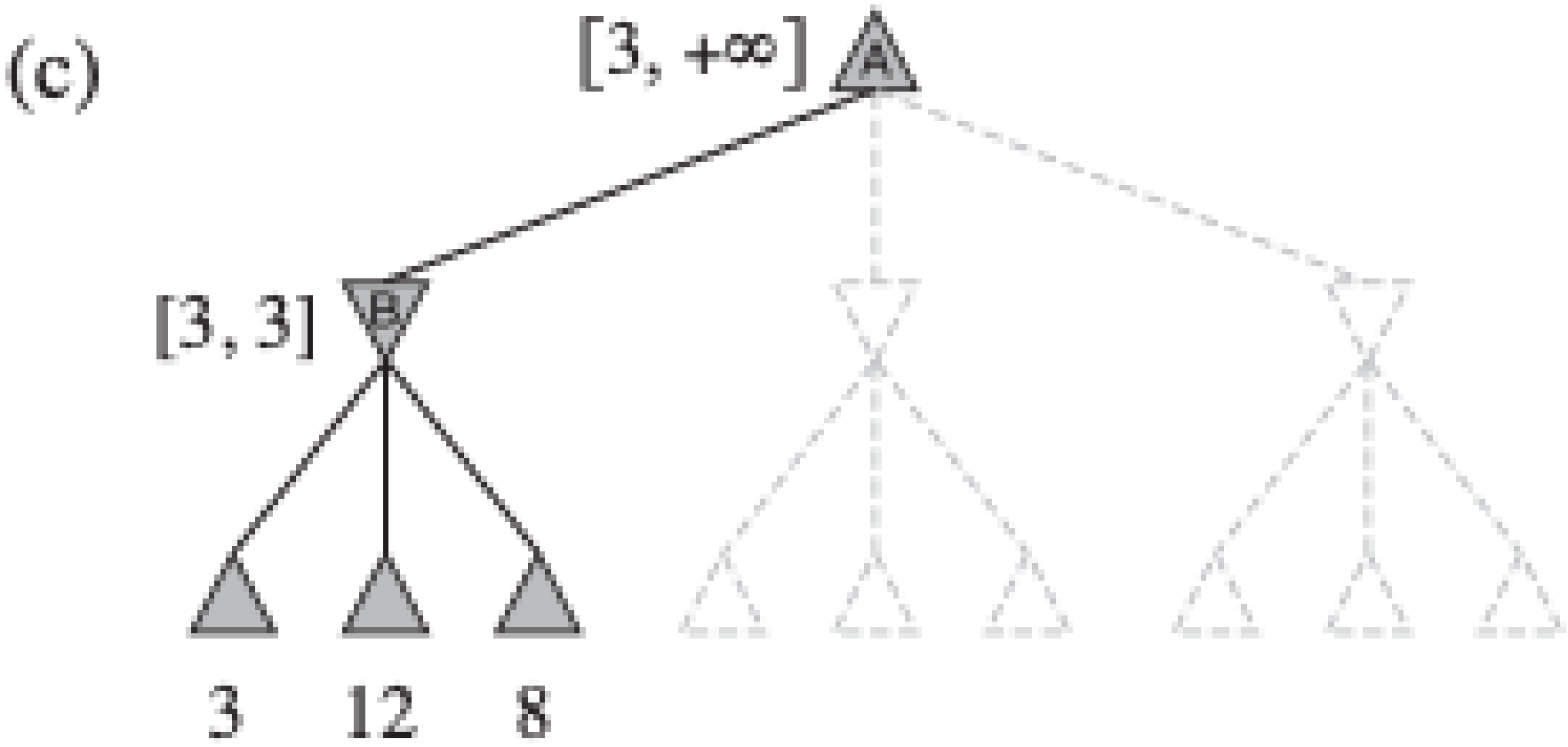
树剪枝

Vector:[alpha, beta]



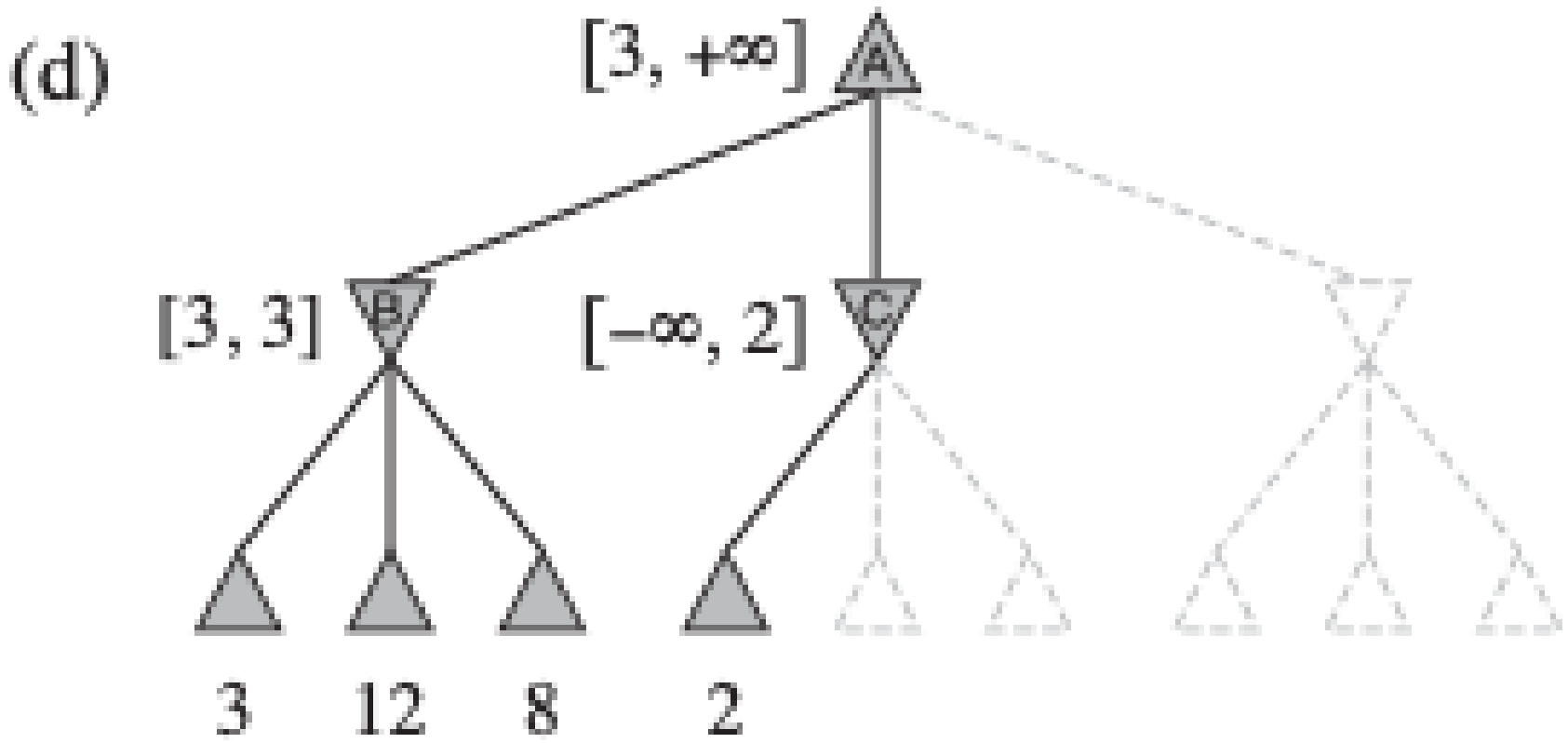
树剪枝

Vector:[alpha, beta]



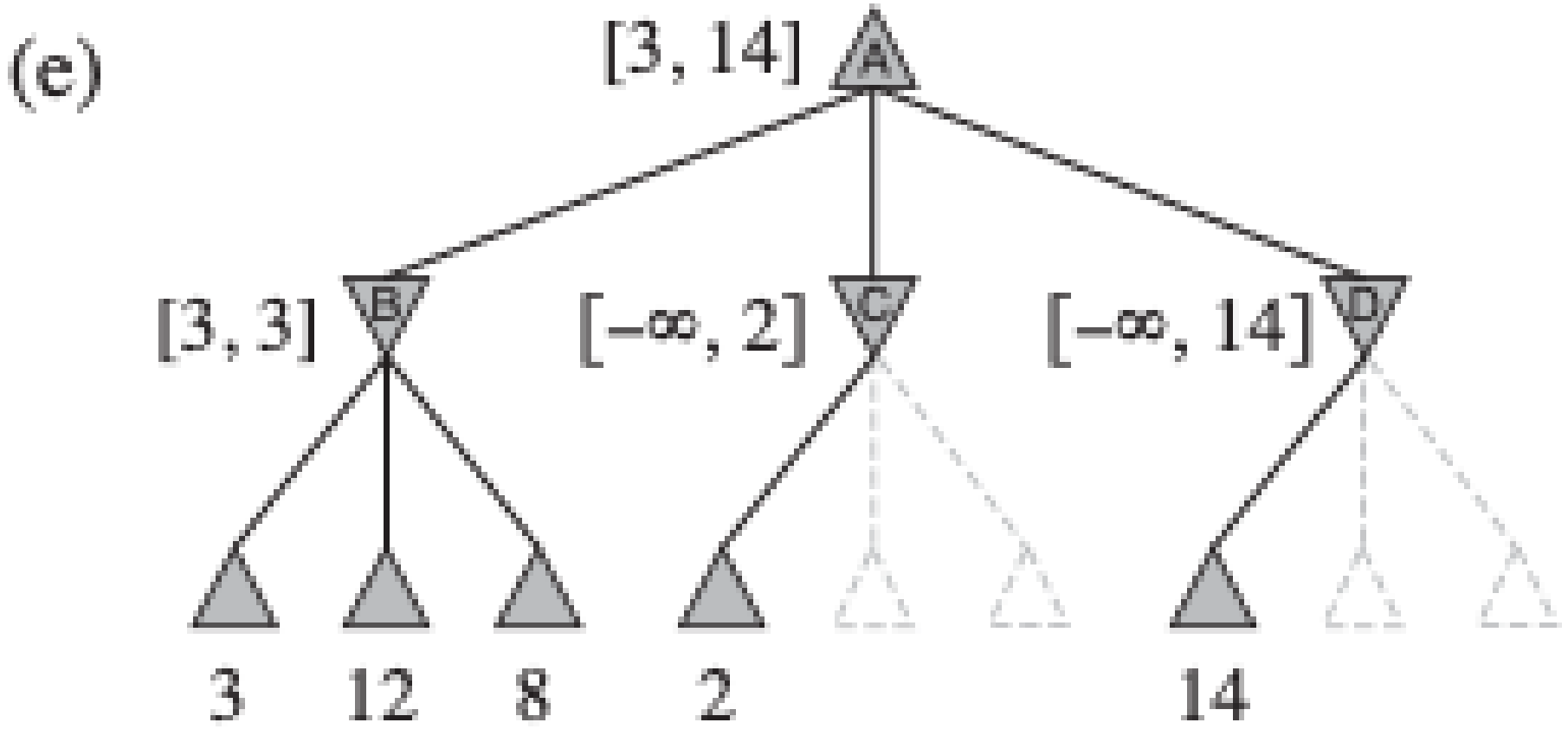
树剪枝

Vector:[alpha, beta]



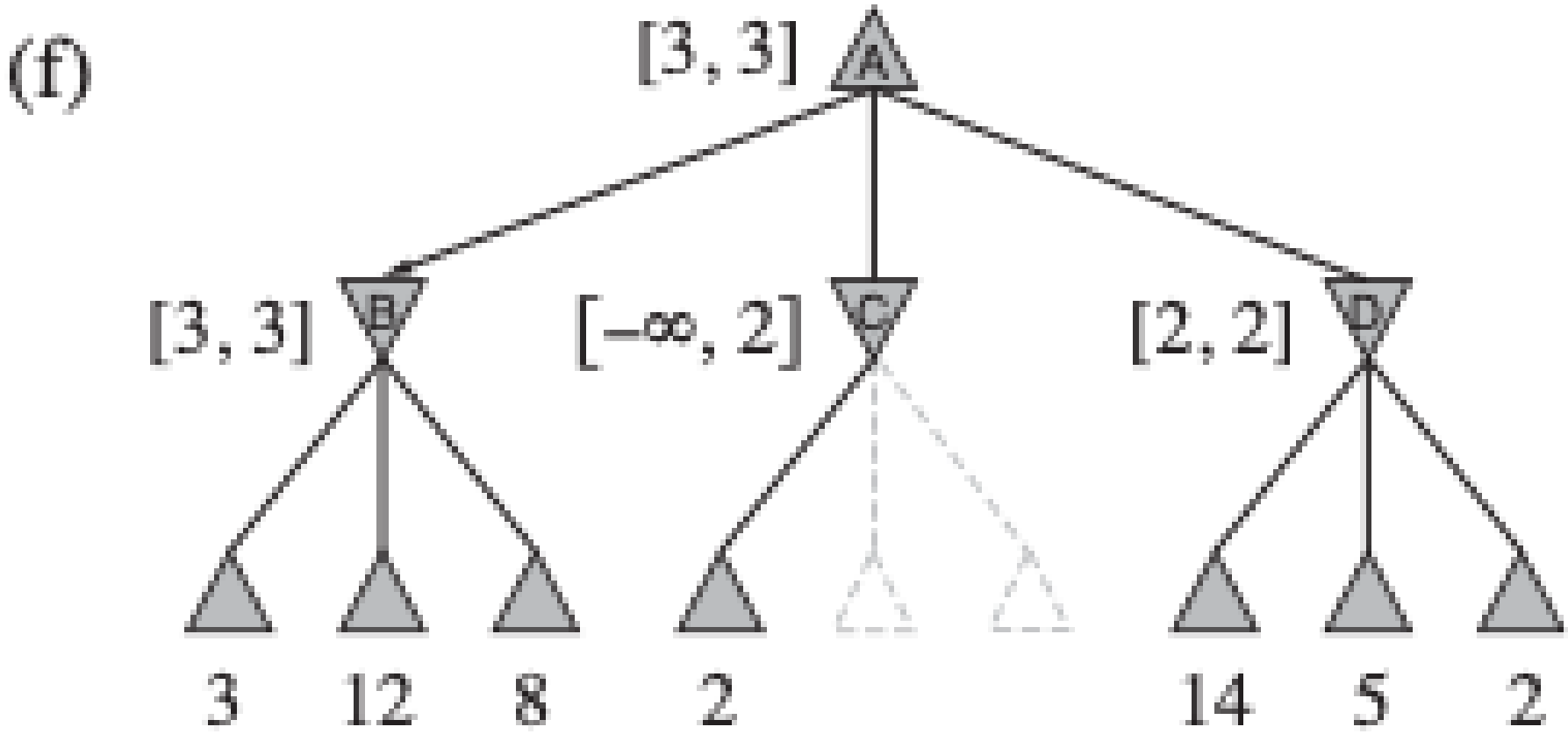
树剪枝

Vector:[alpha, beta]

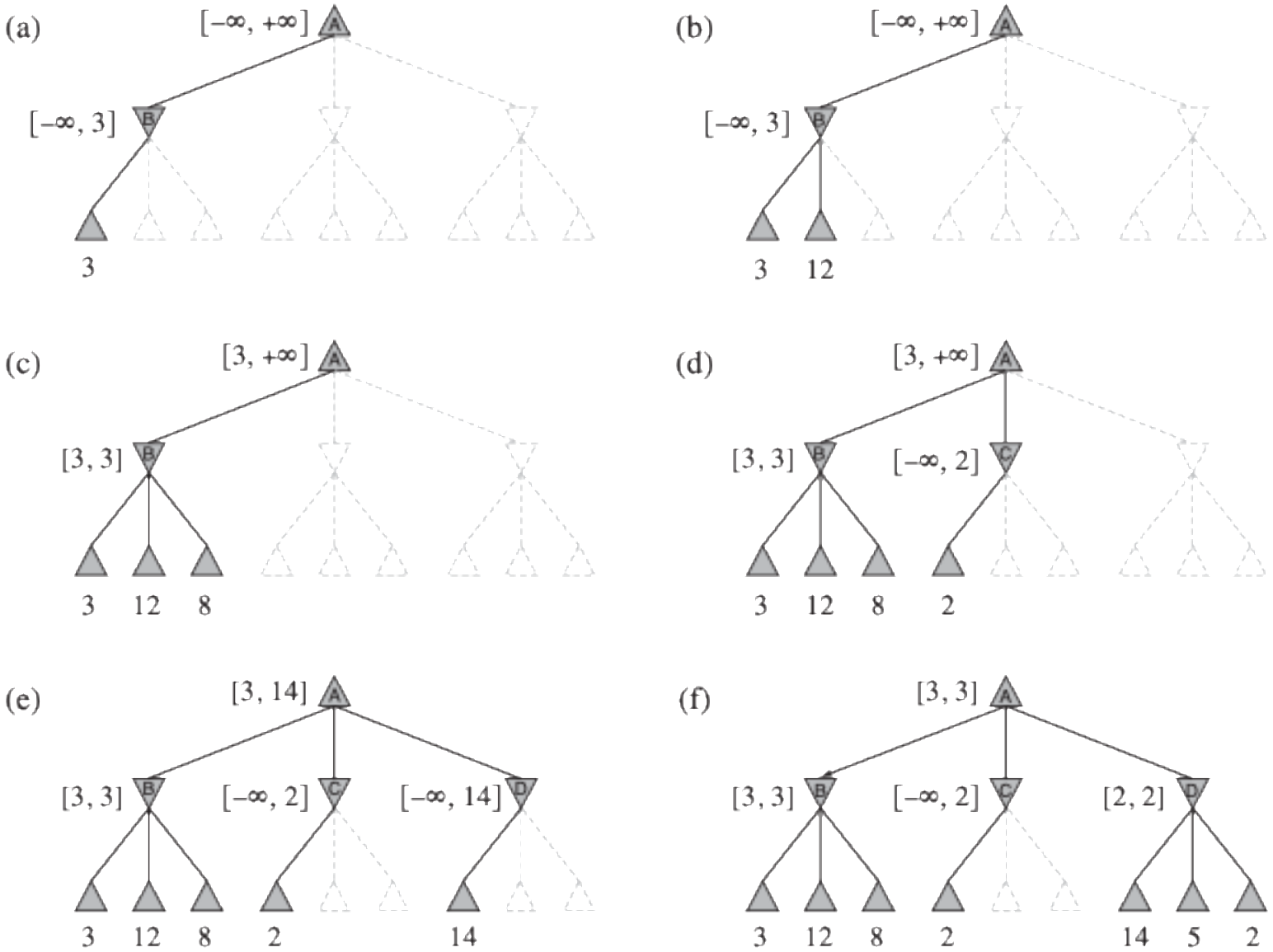


树剪枝

Vector:[alpha, beta]



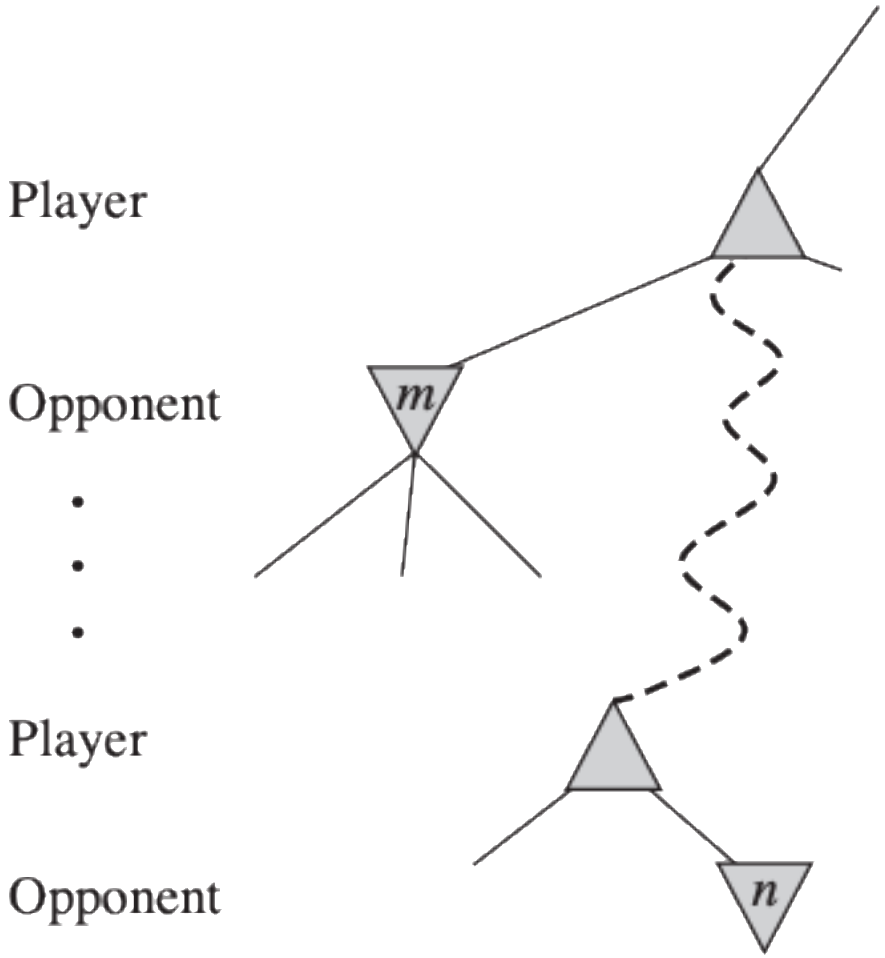
树剪枝


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3, +∞]
A
[3, 3]
B
3
12
8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3
12
8
2
(e)
[3, 14]
A
[3, 3]
B
[−∞, 2]
C
[−∞, 14]
D
3
12
8
2
14
(f)
[3, 3]
A
[3, 3]
B
[−∞, 2]
C
[2, 2]
D
3
12
8
2
14
5
2

Alpha-Beta剪枝

α:目前为止路上发现的 MAX的最佳（即极大值）选择，即α= “至少”

β:目前为止路径上发现的MIN的最佳（即极小值）选择，即β= “至多”



如果对于玩家来说， m好于n，

那么我们永远不会在博弈中到达n

Alpha-Beta剪枝

- 对于MAX节点，如果其孩子结点（MIN结点）的收益大于当前的α值，则将α值更新为该收益；

  对于MIN结点，如果其孩子结点（MAX结点）的收益小于当前的β值，则将β值更新为该收益。

  根结点（MAX结点）的α值和β值分别被初始化为$-\infty$和$+\infty$

- 随着搜索算法不断被执行，每个结点的α值和β值不断被更新。大体来说，每个结点的$[\alpha, \beta]$从其父结点提供的初始值开始，取值按照如下形式变化：α逐渐增加、β逐渐减少。不难验证，如果一个结点的α值和β值满足$\alpha > \beta$的条件，则该结点尚未被访问的后续结点就会被剪枝，因而不会被智能体访问

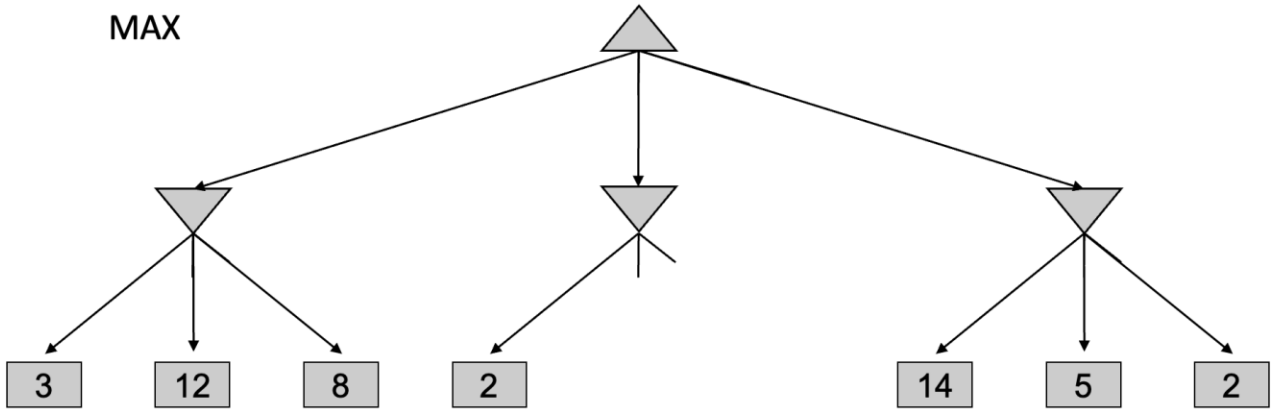
Alpha-Beta剪枝

α:目前为止路上发现的 MAX的最佳（即极大值）选择
β:目前为止路径上发现的MIN的最佳（即极小值）选择

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state, α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

思考



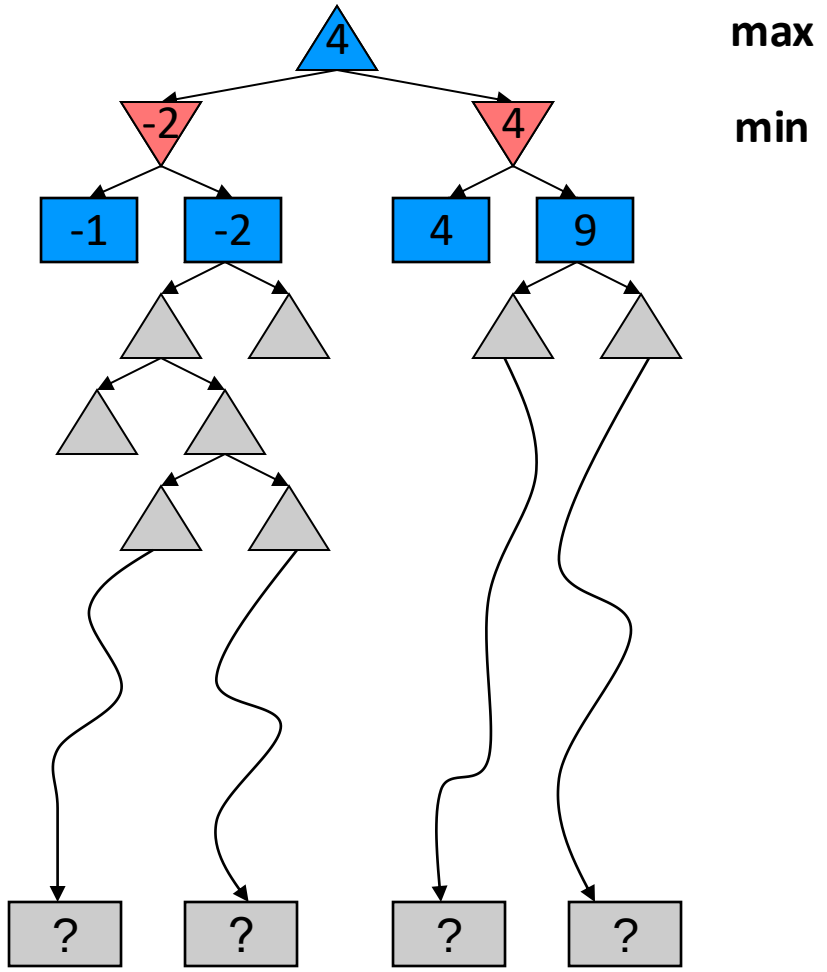
启发？　　搜索顺序很重要

可以设计方案对后继状态进行排序，

例如，对于象棋，可以设计排序规则：吃子>威胁>前进>后退

资源受限

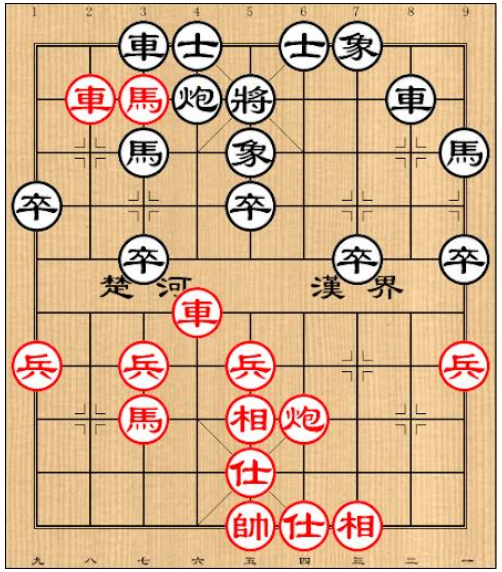
尽管alpha-beta剪枝能够避免搜索完整的空间，但是仍然要**搜索部分空间直至终止状态**，这样的搜索深度也是不现实的

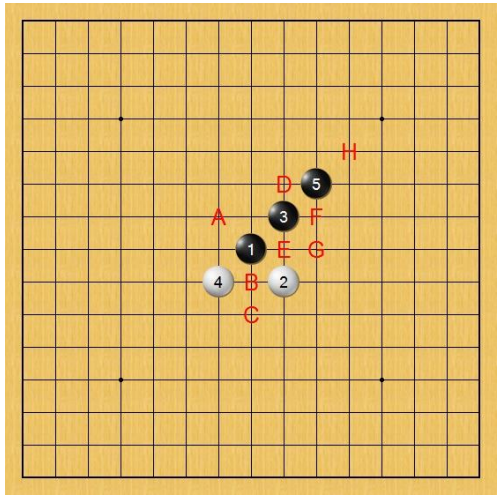


一个可行的思路：

参考启发式搜索，设计评估函数用于搜索中的状态，有效地把非终止节点变成终止节点

评估函数





如何设置评估函数？

以象棋为例，要考虑兵的数目、车的数目、马的数目等等

兵1分，马3分，车5分…

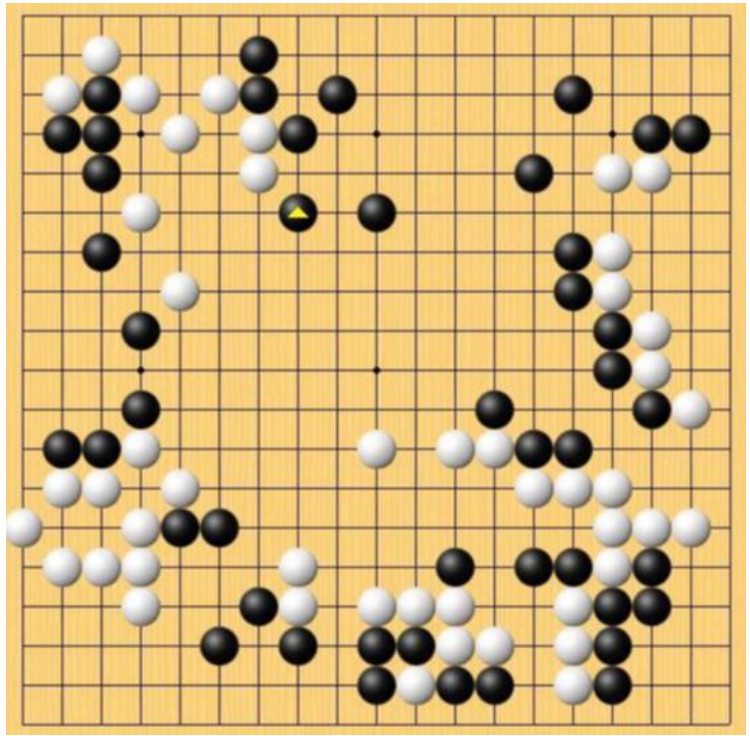
形式化描述：加权线性函数

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

w_i表示权重，f_i是棋局的某个特征

评估函数

Alpha-beta搜索用于围棋会面临什么挑战？



- 分支因子大

围棋分支因子开始时为361，搜索层数受限

- 评估函数难设置

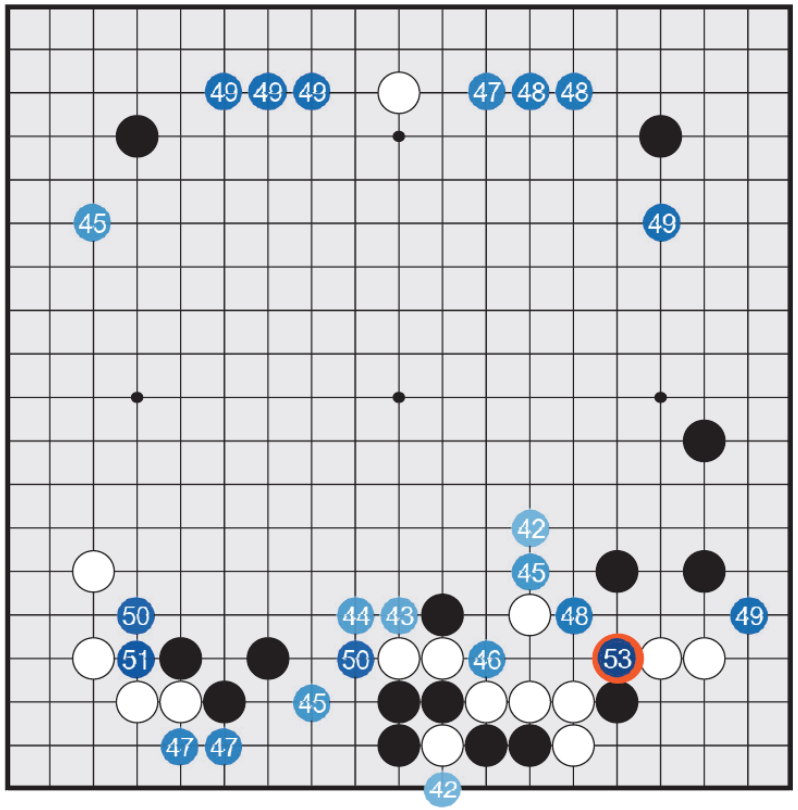
现代围棋程序基本不采用**alpha-beta搜索**

提纲

- 对抗博弈
  - 双人零和博弈
- 确定性搜索
  - 最大最小搜索
  - Alpha-beta 剪枝
- 基于模拟的搜索
  - 蒙特卡洛树搜索

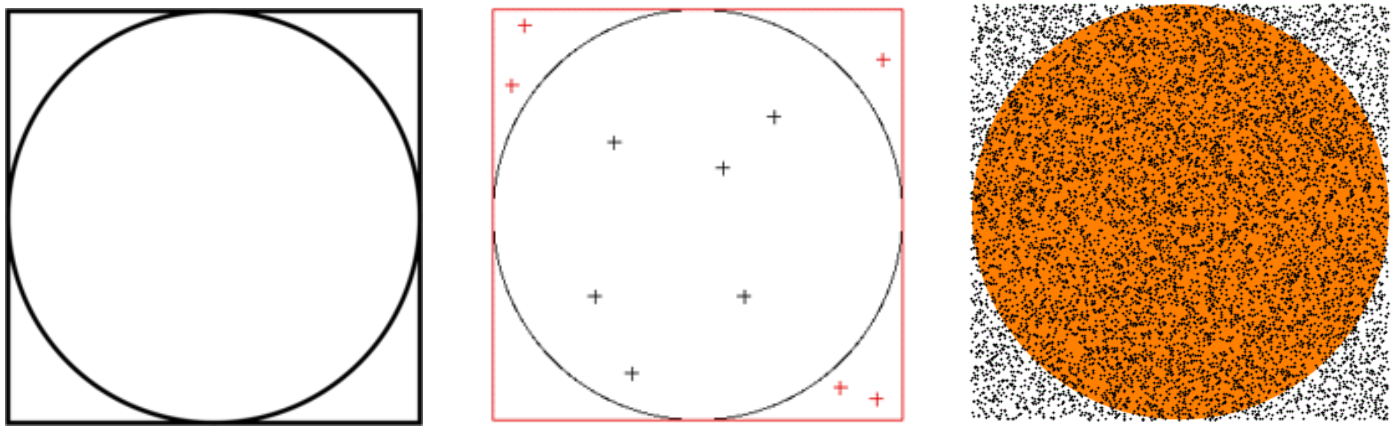
蒙特卡洛搜索

如果不采用评价函数，有没有其他办法评估状态的好坏？



蒙特卡洛方法

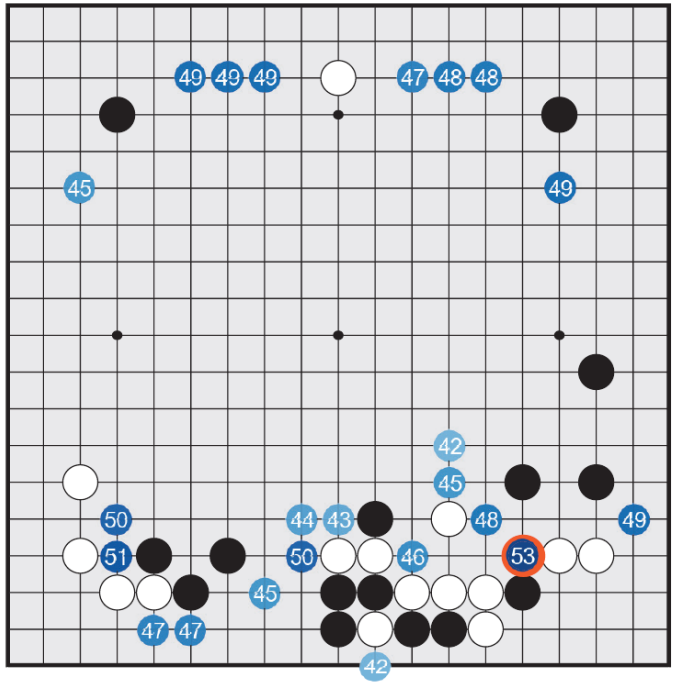
- 蒙特卡洛方法（Monte-Carlo methods）是一类广泛的计算算法
  - 依赖于重复随机抽样来获得数值结果
- 例如，计算圆的面积



$$\text{Circle Surface} = \text{Square Surface} \times \frac{\#\text{points in circle}}{\#\text{points in total}}$$

蒙特卡洛方法

- 围棋对弈：估计当前状态下的胜率

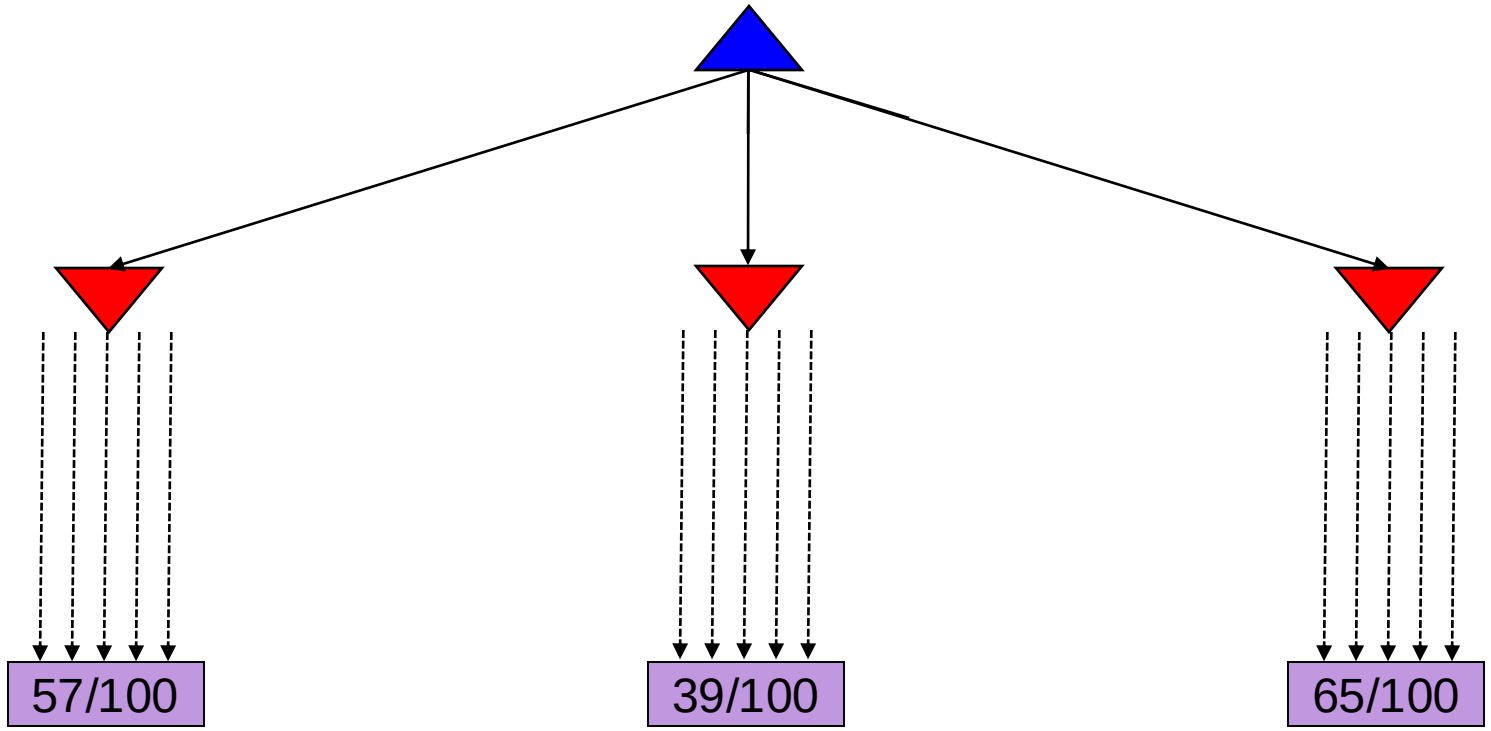


从当前状态出发，做随机模拟

$$\text{Win Rate}(s) = \frac{\#\text{win simulation cases started from } s}{\#\text{simulation cases started from } s \text{ in total}}$$

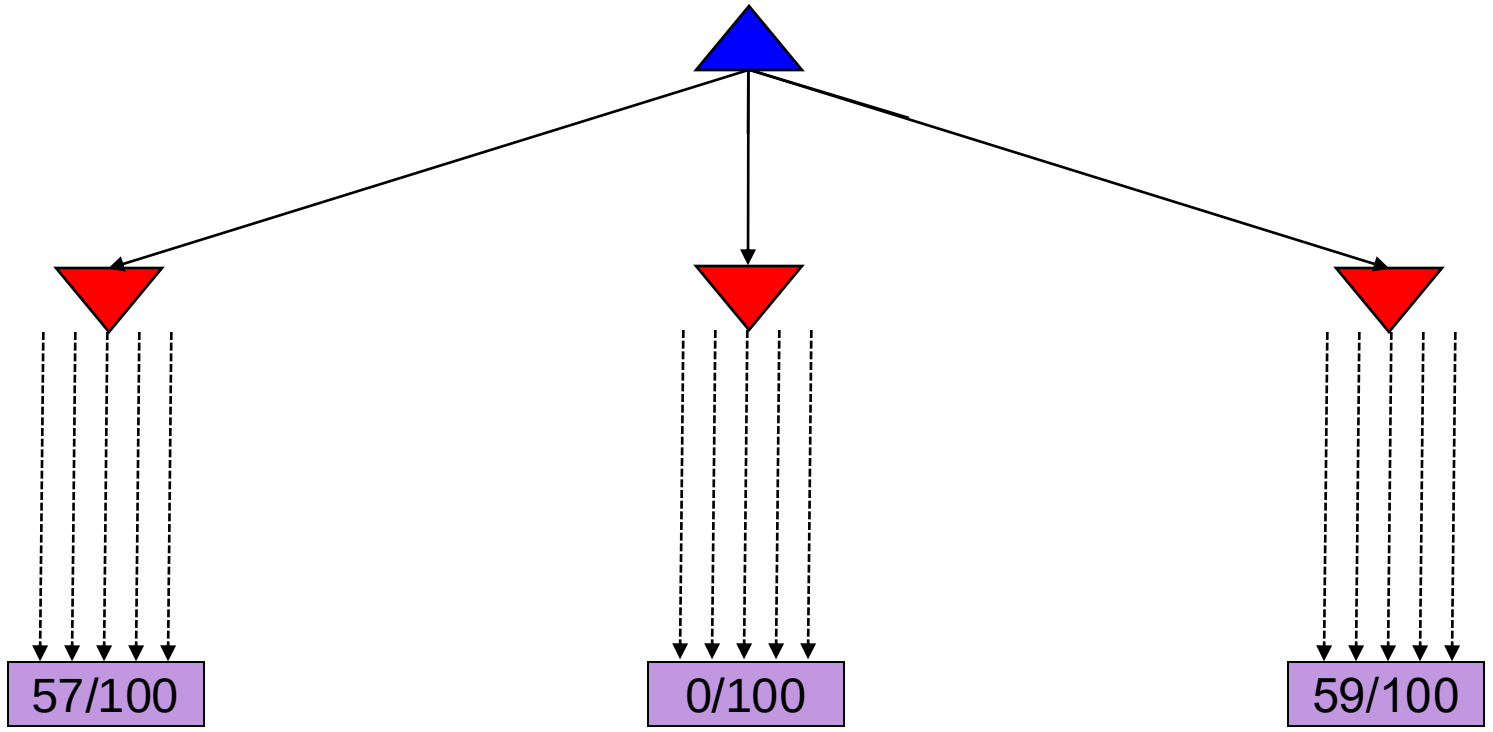
蒙特卡洛搜索

模拟：从一个节点出发，进行大量模拟，记录赢的次数



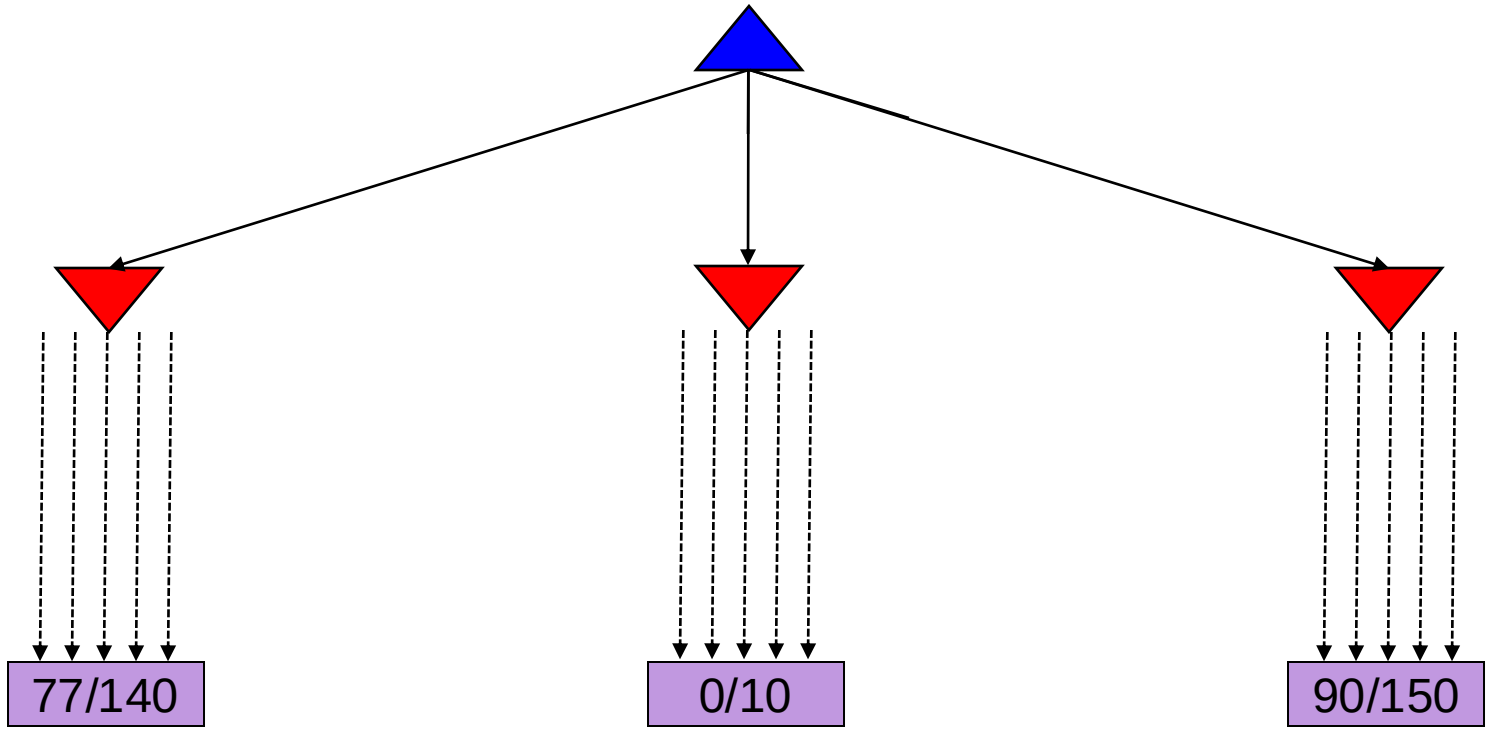
蒙特卡洛搜索

模拟：从一个节点出发，进行大量模拟，记录赢的次数



蒙特卡洛搜索

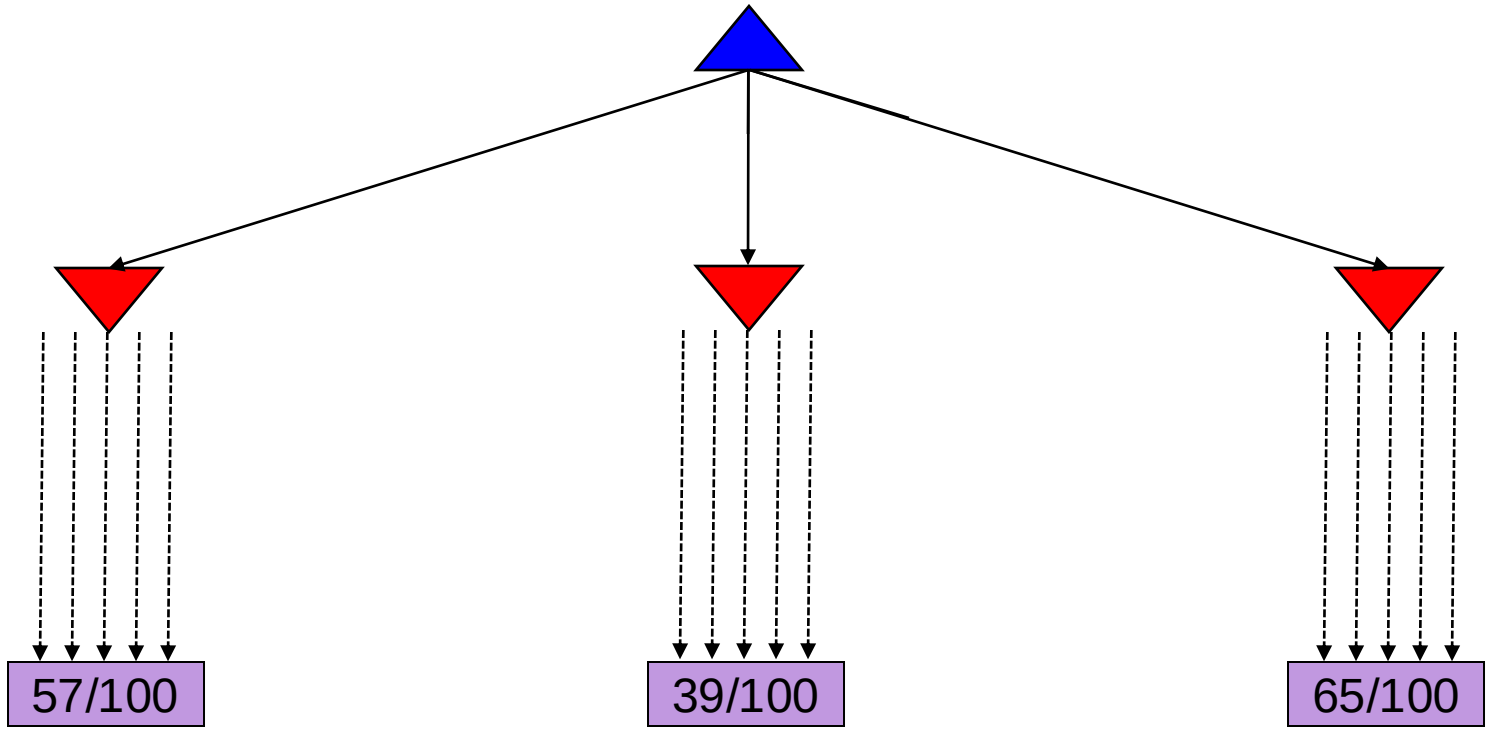
模拟：从一个节点出发，进行大量模拟，记录赢的次数



蒙特卡洛搜索

资源是有限的，选择哪些节点进行模拟？

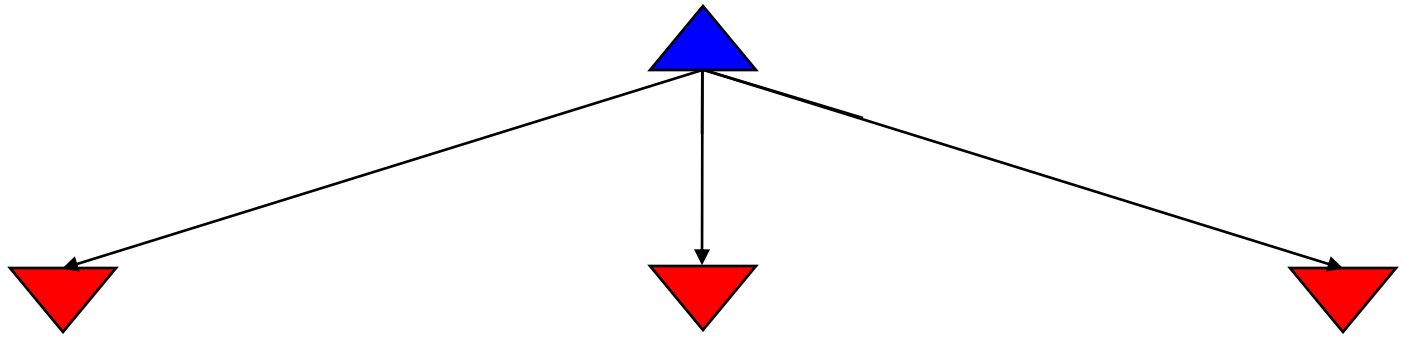
选择更有希望的节点进行模拟



蒙特卡洛搜索

如何评估节点的价值？

大量的模拟



先有鸡还是先有蛋？

多臂老虎机

◘ Multi-Armed Bandits

赌博机有 K 个摇臂，每次转动一个赌博机摇臂，
赌博机则会随机吐出一些硬币

如何在有限次数的尝试中使收益最大化？

多臂老虎机

如何在有限次数的尝试中使收益最大化？

- 仅探索 (Exploration-only)

每个摇臂摇动 T / K 次

不足：浪费次数在收益较差的摇臂上

- 仅利用 (Exploitation-only)

1. 每个摇臂摇动一次，记录收益
2. 剩余的T-K次全部用在收益最大的摇臂上

不足：一次估计的结果不可靠

探索利用窘境

Exploration-Exploitation Dilemma

ϵ-贪心算法

ϵ-贪心算法： 在探索与利用之间进行平衡的搜索算法

在第t步，ϵ-贪心算法按照如下机制来选择摇动赌博机：

- 以 **$1-\epsilon$ 的概率**，选择在过去$t-1$次摇动赌博机所得**平均收益最高的摇臂**进行摇动；
- 以 **ϵ 的概率，随机选择一个摇臂**进行摇动。

不足：没有考虑每个摇臂被探索的次数

输入： 摇臂数 K;
奖赏函数 R;
尝试次数 T;
探索概率 ϵ.

过程：

$r = 0$;
$\forall i = 1, 2, \ldots K : Q(i) = 0,\ \mathrm{count}(i) = 0$;
for $t = 1, 2, \ldots, T$ **do**
  if rand()$< \epsilon$ **then**
    $k =$ 从 $1, 2, \ldots, K$ 中以均匀分布随机选取
  else
    $k = \arg\max_i Q(i)$
  end if
  $v = R(k)$;
  $r = r + v$;
  $Q(k) = \frac{Q(k) \times \mathrm{count}(k) + v}{\mathrm{count}(k) + 1}$;
  $\mathrm{count}(k) = \mathrm{count}(k) + 1$;
end for

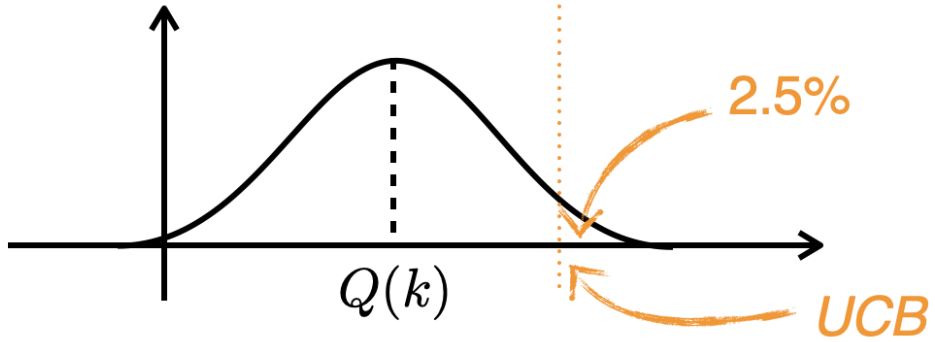
输出： 累积奖赏 r

上限置信区间（Upper-Confidence Bound）

上限置信区间算法（Upper Confidence Bounds, UCB）：为每个动作的奖励期望计算一个估计范围，优先采用估计范围上限较高的动作

假设每个摇臂的均值为$Q(k)$，估计的偏差为$\delta(k)$

每次根据$Q(k)+\delta(k)$选择摇臂

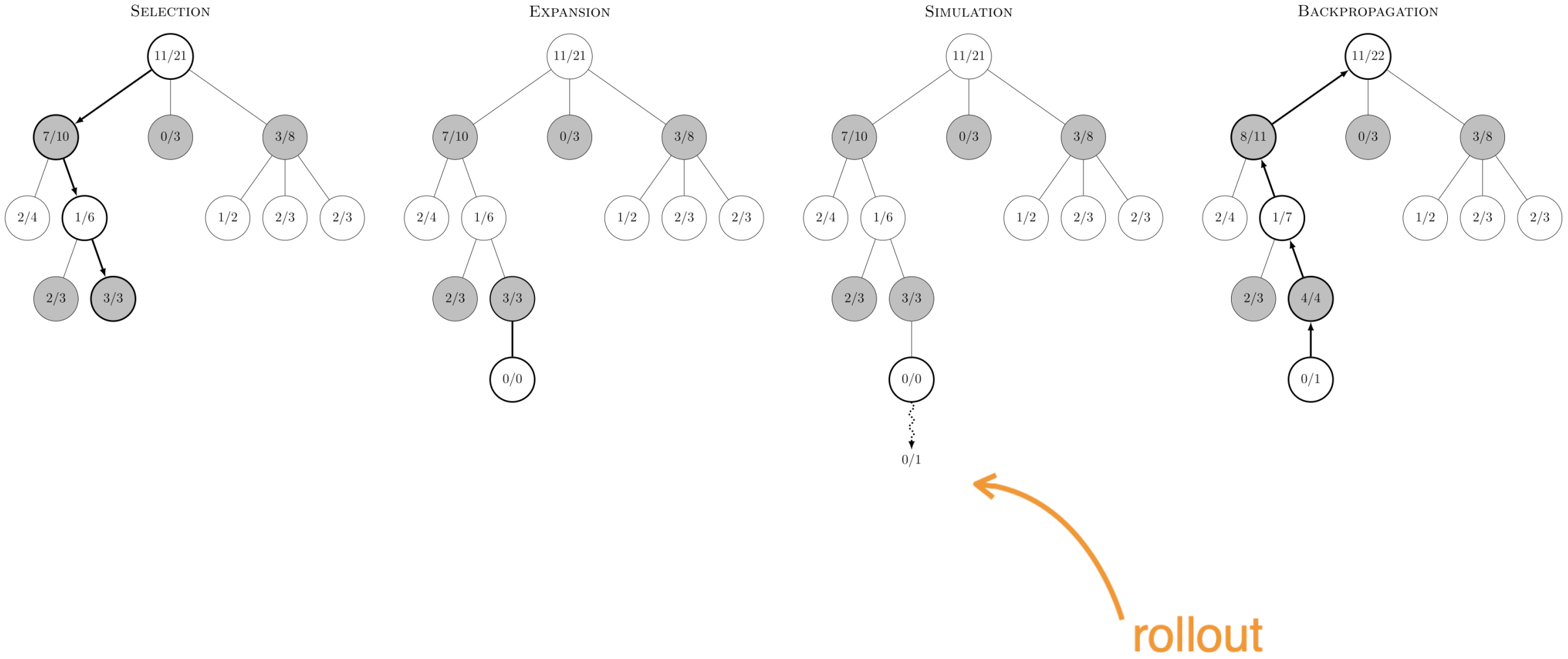


$$Q(k)+2\ln\sqrt{T/T_k}$$

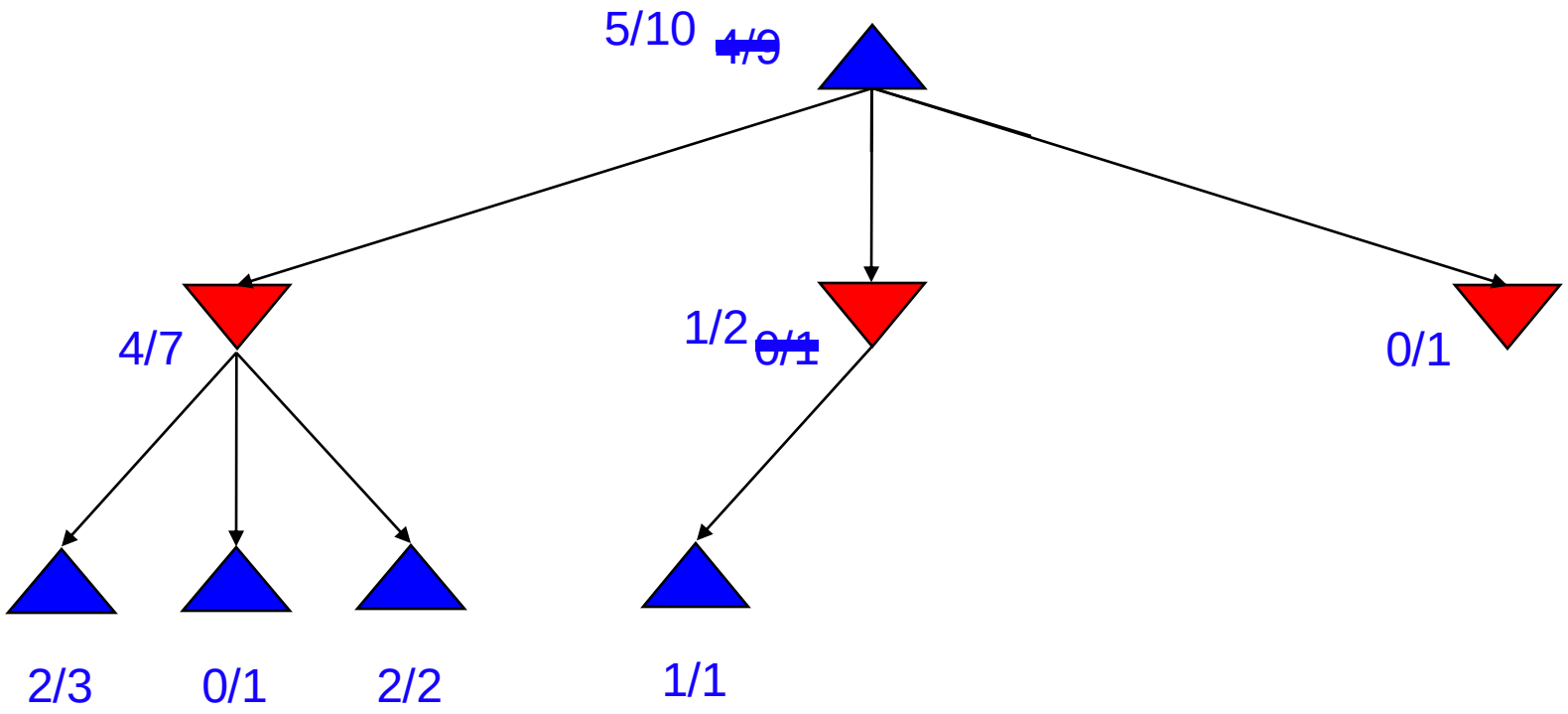
蒙特卡洛搜索

- **选择（selection）**：选择指算法从搜索树的根节点开始，向下递归选择子节点，直至到达叶子节点或者到达具有还未被扩展过的子节点的节点L，可以采用UCB算法
- **扩展（expansion）**：如果节点 L 不是一个终止节点（或对抗搜索的终局节点），则随机扩展它的一个未被扩展过的后继边缘节点M
- **模拟（simulation）**：从节点M出发，模拟扩展搜索树，直到找到一个终止节点
- **回溯（backpropagation）**：用模拟所得结果（终止节点的代价或游戏终局分数）回溯更新模拟路径中M以上（含M）节点的奖励均值和被访问次数

蒙特卡洛搜索

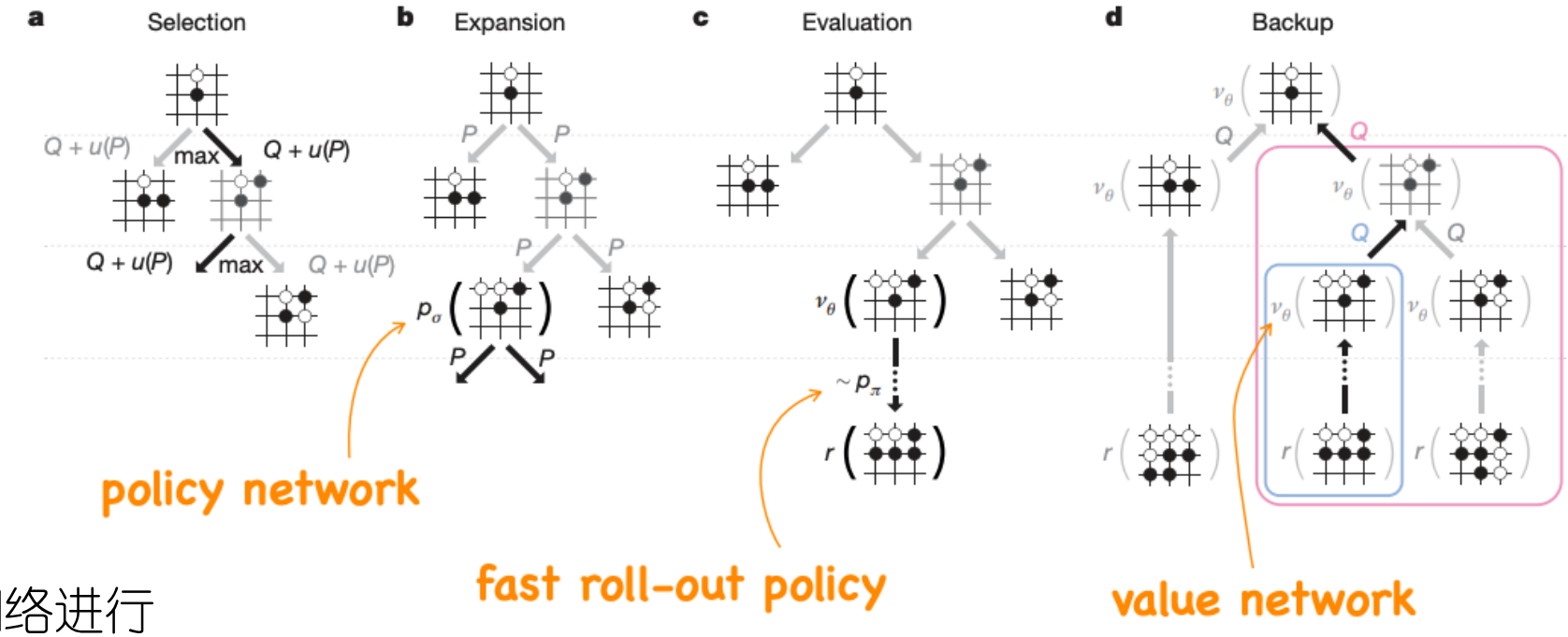

Selection
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3
Expansion
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0
Simulation
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0
0/1
Backpropagation
11/22
8/11
0/3
3/8
2/4
1/7
1/2
2/3
2/3
2/3
4/4
0/1
rollout

蒙特卡洛搜索



AlphaGO

综合使用蒙特卡洛搜索、神经网络、强化学习等技术



训练策略网络进行节点扩展

训练神经网络直接评估当前节点，无需模拟

Open AI O1

当人们写作或说话时，常常会停下来思考。然而，大语言模型在通过 Next Token Prediction 生成回答时，更像是一种“快思考”过程。由于缺乏详细的中间推理步骤，模型一开始可能会犯错，而这些错误可能会传播，最终导致生成的答案也是错误的。

蒙特卡洛树搜索（MCTS），将输出建模为一系列节点，这些节点可以是 Token 级别或句子级别。例如：

- **Token 级别的节点**：每个节点对应生成序列中的一个 Token。通过MCTS，模型可以探索不同的Token序列，最终生成更连贯的响应。
- **句子级别的节点**：在复杂推理任务中，每个节点可以代表一个完整的句子或推理步骤，帮助模型更好地处理多步推理任务。

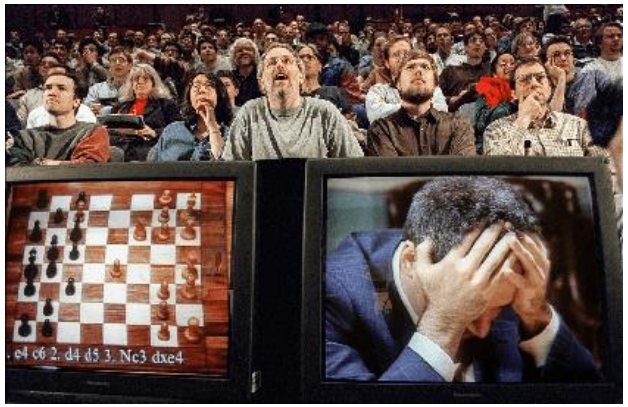
ReST-MCTS*: LLM Self-Training via Process Reward Guided Tree Search

博弈AI的发展现状

- **跳棋：** 1990年战胜人类冠军，使用alpha-beta搜索和存有390000亿个残局的数据库表现趋于完美

- **国际象棋：** IBM的深蓝国际象棋程序，1997年击败世界冠军Garry Kasparov，每步棋搜索最多至300亿个棋局，常规搜索深度是14步，某些情况下搜索深度可以达到40层，评估函数考虑了超过8000个特征

- **围棋：** AlphaGO，采用蒙特卡洛搜索+深度强化学习，AlphaZero，无需人类棋谱数据进行训练



博弈AI的发展现状

- **星际争霸：** DeepMind 团队基于多智能体深度强化学习推出的AlphaStar在星际争霸II中达到了人类大师级的水平，并且在《星际争霸II》的官方排名中超越了 99.8%的人类玩家

- **DOTA2：** OpenAI推出的“OpenAI Five”击败世界冠军

- **王者荣耀：** 腾讯推出的觉悟AI，可以击败97%的玩家，并且多次击败顶尖职业团队

	GO	MOBA
Action Space	$250^{150} \approx 10^{360}$ (250 pos available, 150 decisions per game in average)	10^{1500} (10 options, 1500 actions per game)
State Space	$3^{360} \approx 10^{170}$ (361 pos, 3 states each)	10^{20000} (10 heroes, 2000+pos * 10+states)

本章小结

- 双人零和博弈：两个Rational Agent之间的游戏
- 最大最小搜索：类似于DFS，通过递归实现
- Alpha-Beta剪枝：减去不会影响上层节点的分支
- 蒙特卡洛搜索：通过模拟判断节点的价值

搜索部分小结

- 掌握常见的无信息搜索算法，能够编程实现DFS，BFS
- 掌握常见的启发式搜索算法，能够编程实现A*搜索
- 掌握常见的局部搜索算法，能够编程实现爬山搜索
- 了解博弈搜索算法的基本思想，能够综合运用，解决现实博弈问题，如象棋、黑白棋等。