



南京大學

NANJING UNIVERSITY

人工智能导论

对抗搜索

(Adversarial Search)

郭兰哲

南京大学 智能科学与技术学院

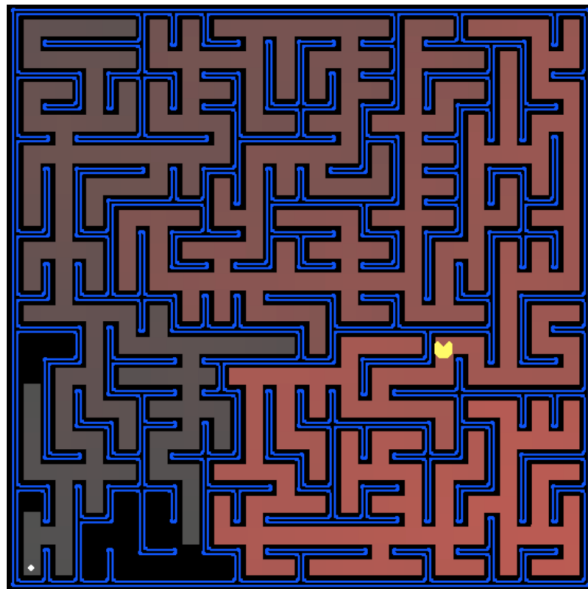
<https://www.lamda.nju.edu.cn/guolz>

Email: guolz@nju.edu.cn

通知：作业1

Project 1: Search

Due: **2026.04.06**.



介绍

在这个项目中，你的吃豆人智能体将在他的迷宫世界中寻找路径，包括到达一个特定的位置，以及有效地收集食物。你需要结合我们课堂所学的知识，实现 `depthFirstSearch`、`uniformCostSearch` 等通用搜索算法及启发式函数，并将其应用于吃豆人场景。

通知：作业2

Project 2: 技能五子棋AI

Due: **04.30**



本次实验中, 你将要实现一个基于搜索的技能五子棋AI, 并且能够在提供的环境中进行对弈.

本次作业所需的文件为: [Lab2.zip](#)

本次实验中包含以下几个文件:

如何做科研？

人工智能领域研究的大致流程

Topic -> Problem -> Idea -> Concrete Work

(Theoretical analysis, experiments, etc.)

-> Paper Writing -> Submit

以课程作业为例：

Topic，已经给定：

- 大模型数学推理
- 大模型逻辑推理
- 旅行规划(Travel Planning)
- LLM Agent
- 抽象视觉推理ARC

以课程作业为例：

如何找到Problem?

1. 先尝试调研已有的相关论文
2. 复现代表性的方法
3. 分析实验结果，Bad Case，
寻找当前方法在哪些方面仍存在不足

我想针对抽象视觉推理ARC-AGI数据做一些实验分析，你可以帮我调研一些相关的有代表性的文章，并帮我设计一个实验方案吗？

方法 M1: Direct Prompting (基线, 1-2周)

目标: 建立最基础的基线，理解 LLM 在无任何适配下的能力上限。

实现步骤:

将每道 ARC 任务序列化为文本，格式如下:

```
Task:
Example 1:
  Input: [[0,1,0],[1,1,1],[0,1,0]]
  Output: [[0,0,0],[0,1,0],[0,0,0]]
...
Test Input: [[...]]
Output:
```

需要对比的变量 (每个变量独立控制):

变量	选项
网格表示	数字矩阵 / 颜色名称 / ASCII字符
提示策略	Zero-shot / Few-shot CoT / 结构化分析提示
模型	GPT-4o-mini / Qwen2.5-7B-Instruct (本地可跑)

评测: Pass@2 (两次尝试至少一次完全正确)，记录每道题的错误类型。

以课程作业为例：

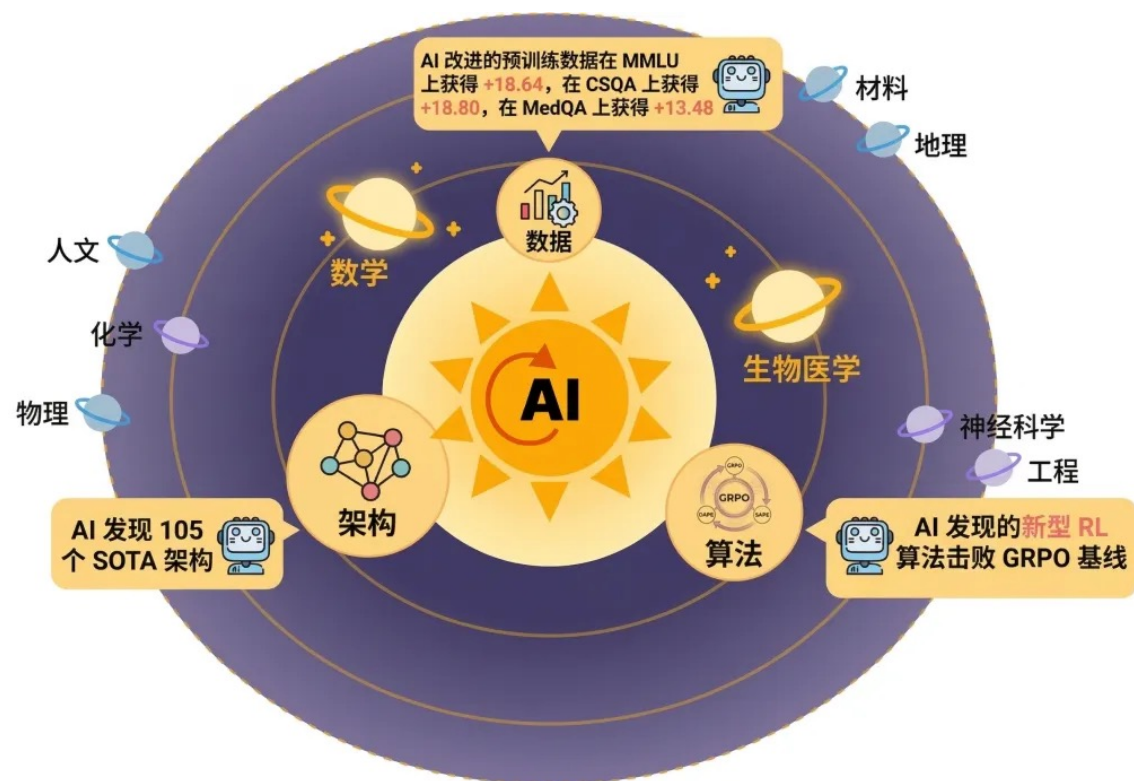
如何提出Idea?

这是考验你聪明才智的时候了

- 大量的相关论文阅读
- 寻求AI的帮助

人类研究员首次全线落败——AI架构、数据、算法三战获科学发现级突破

机器之心 2026年3月29日 10:54 北京 97人



以课程作业为例：

Concrete Work

Idea需要支持

AI领域通常是实验分析

-> 在领域常用的3-5个benchmark上，实验对比你的方法和已有方法，证明你所提的方法能够实现更好的性能

论文示例

科学论文具有清晰的结构，一般包括

- Title
- Abstract
- Introduction
- Previous Work
- Your Contribution
- Experiments
- Discussion
- Conclusion
- Reference

ARC Is a Vision Problem!

Keya Hu Ali Cy Linlu Qiu Xiaoman Delores Ding
Runqian Wang Yeyin Eva Zhu Jacob Andreas Kaiming He
MIT

Abstract

The Abstraction and Reasoning Corpus (ARC) is designed to promote research on abstract reasoning, a fundamental aspect of human intelligence. Common approaches to ARC treat it as a language-oriented problem, addressed by large language models (LLMs) or recurrent reasoning models. However, although the puzzle-like tasks in ARC are inherently visual, existing research has rarely approached the problem from a vision-centric perspective. In this work, we formulate ARC within a vision paradigm, framing it as an image-to-image translation problem. To incorporate visual priors, we represent the inputs on a “canvas” that can be processed like natural images. It is then natural for us to apply standard vision architectures, such as a vanilla Vision Transformer (ViT), to perform image-to-image mapping. Our model is trained from scratch solely on ARC data and generalizes to unseen tasks through test-time training. Our framework, termed Vision ARC (VARC), achieves 60.4% accuracy on the ARC-1 benchmark, substantially outperforming existing methods that are also trained from scratch. Our results are competitive with those of leading LLMs and close the gap to average human performance.¹

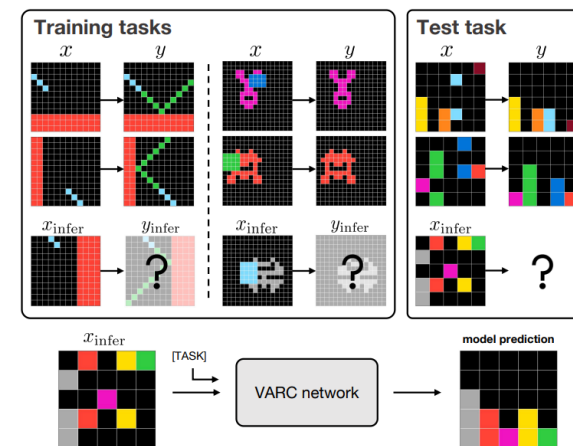


Figure 1. The ARC benchmark (top) consists of a collection of many different tasks, where each task has a few (e.g., 2-4) examples. We propose the Vision ARC (VARC) framework, which addresses the ARC problem as an image-to-image translation problem, from a computer vision perspective (bottom). In this illustration, the underlying concepts of the three tasks can be roughly described by humans as: “reflection” (left), “symmetry” (middle), and “gravity” (right). These concepts are closely related to the visual and physical world.

论文示例

科学论文具有清晰的结构，一般包括

- Title
- Abstract
- Introduction
- Previous Work
- Your Contribution
- Experiments
- Discussion
- Conclusion
- Reference

system	#params	ARC-1	ARC-2
<i>large language models (LLMs)</i>			
Deepseek R1 [21]	671B	15.8	1.3
Claude 3.7 8k [18]	N/A	21.2	0.9
o3-mini-high [18]	N/A	34.5	3.0
GPT-5 [18]	N/A	44.0	1.9
Grok-4-thinking [18]	1.7T	66.7	16.0
Bespoke (Grok-4) [8]	1.7T	79.6	29.4
<i>recurrent models</i>			
HRM [53]	27M	40.3	5.0
TRM [27]	7M	44.6	7.8
<i>vision models</i>			
VARC	18M	<u>54.5</u>	<u>8.3</u>
VARC (ensemble)	73M	60.4	11.1
<i>human results</i>			
avg. human [31]	-	60.2	-
best human [18]	-	98.0	100.0

相关竞赛

<https://www.kaggle.com/competitions/arc-prize-2026-arc-agi-3>



ABSTRACTION AND REASONING CORPUS · FEATURED CODE COMPETITION · 7 MONTHS TO GO

Join Competition



ARC Prize 2026 - ARC-AGI-3

Create an AI capable of fluid intelligence



Overview Data Code Models Discussion Leaderboard Rules

Overview

Build systems that learn and adapt to novel, human-solvable tasks they've never seen before and advance AI's ability to learn new skills efficiently.

Start

4 days ago



Close

7 months to go



Entry

Competition Host

Abstraction and Reasoning Corpus



Prizes & Awards

\$850,000 ⓘ

Awards Points & Medals

Participation

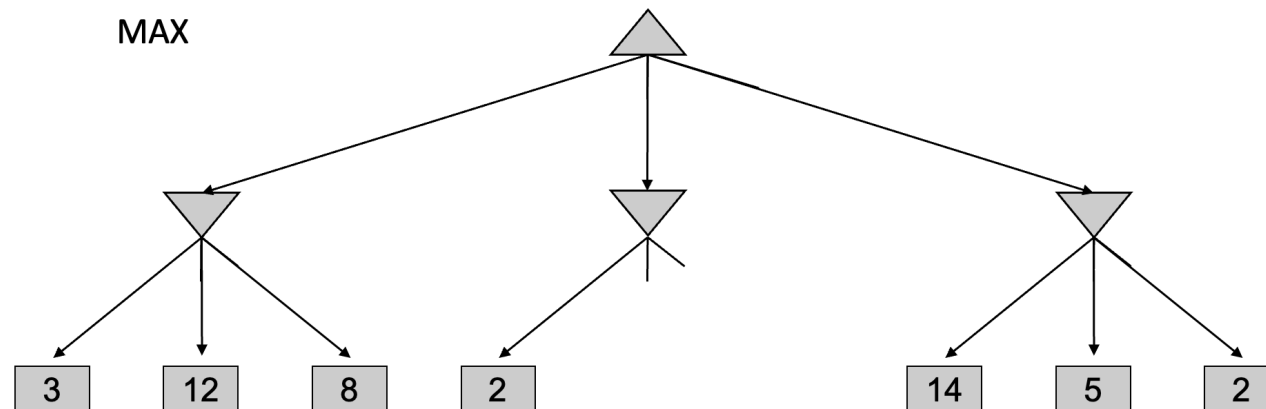
1,634 Entrants

190 Participants

182 Teams

306 Submissions

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,

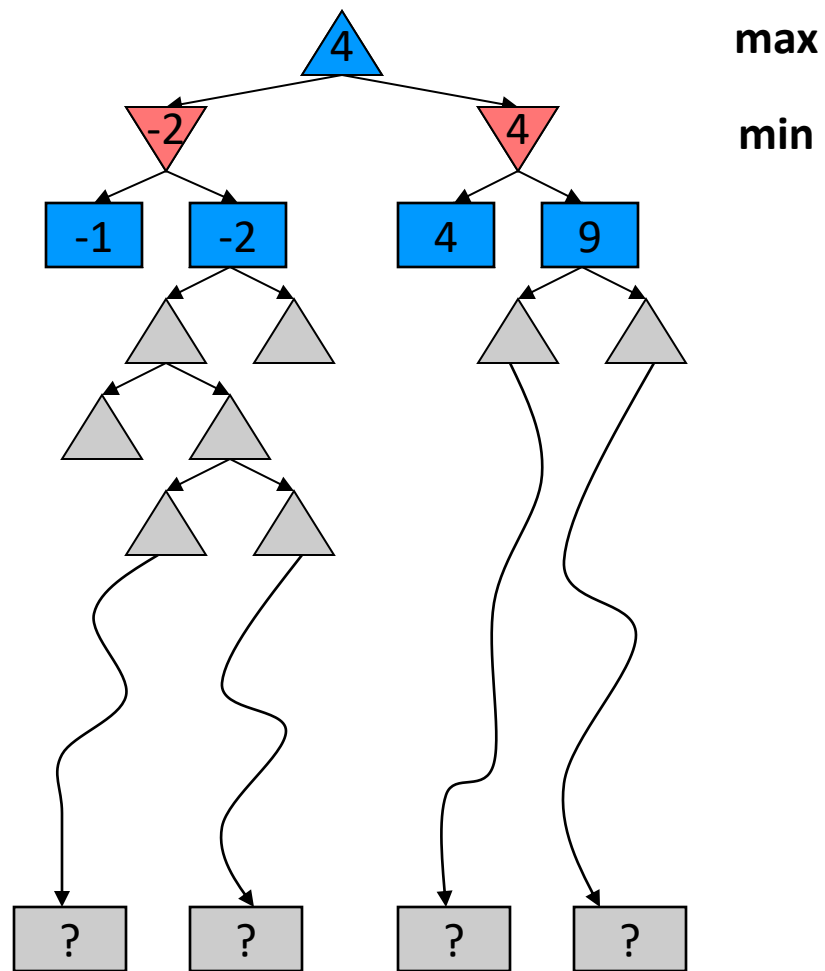
例如, 对于象棋, 可以设计排序规则: 吃子>威胁>前进>后退

资源受限

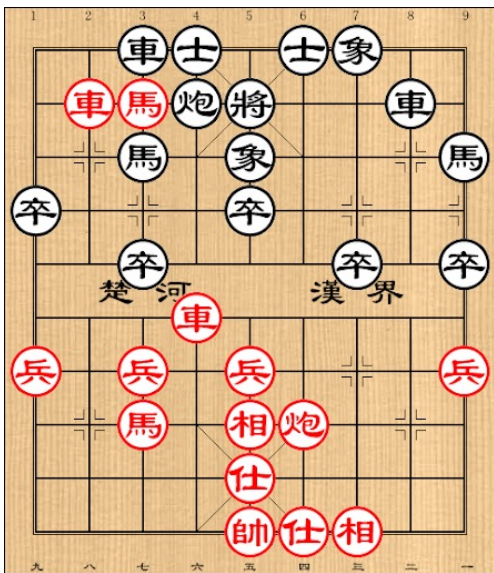
尽管alpha-beta剪枝能够避免搜索完整的空间，
但是仍然要**搜索部分空间直至终止状态**，这样
的搜索深度也是不现实的

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？

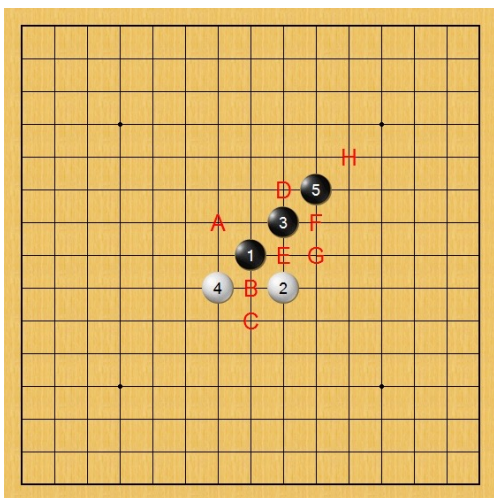
以象棋为例，要考虑兵的数目、车的数目、马的数目等等

兵1分，马3分，车5分...

形式化描述：加权线性函数

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

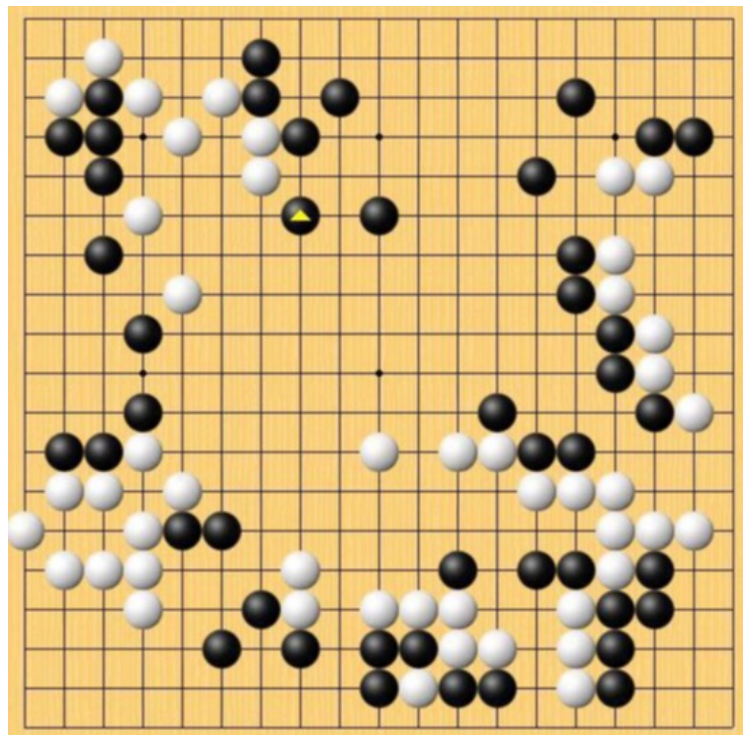
w_i 表示权重， f_i 是棋局的某个特征



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

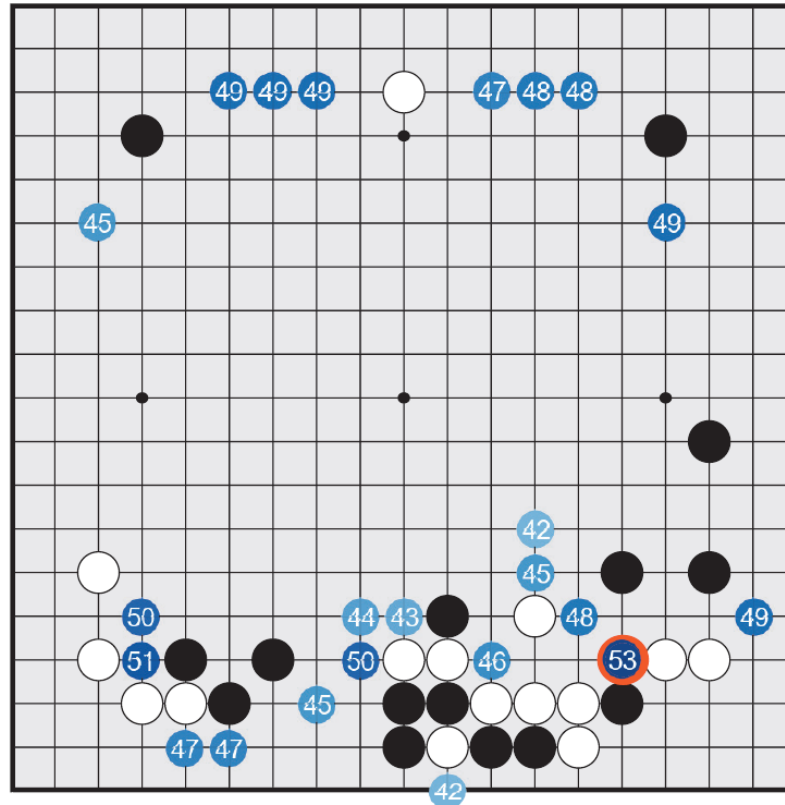
- 最大最小搜索
- Alpha-beta 剪枝

□ 基于模拟的搜索

- 蒙特卡洛树搜索

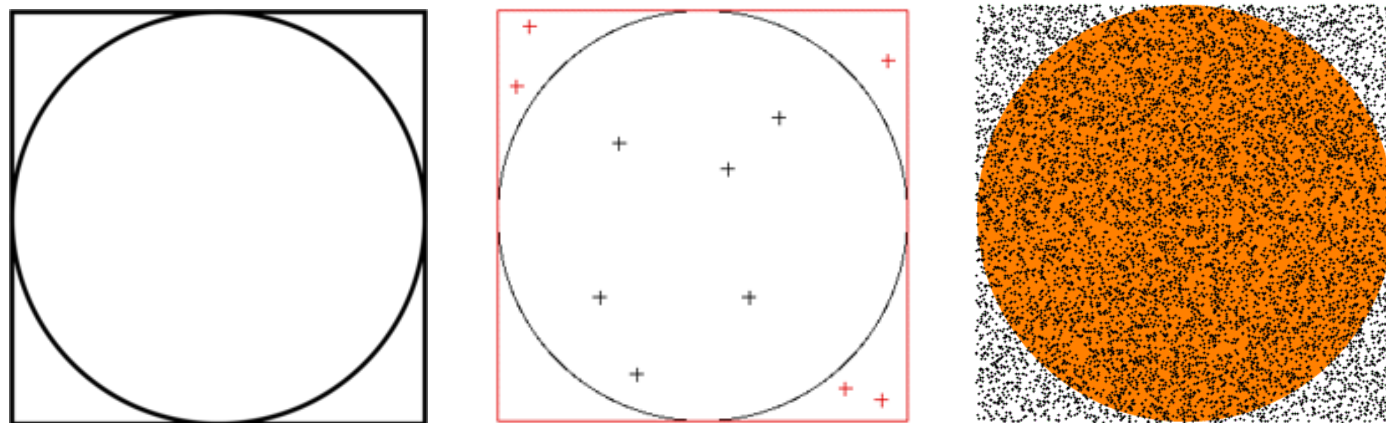
状态评估

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

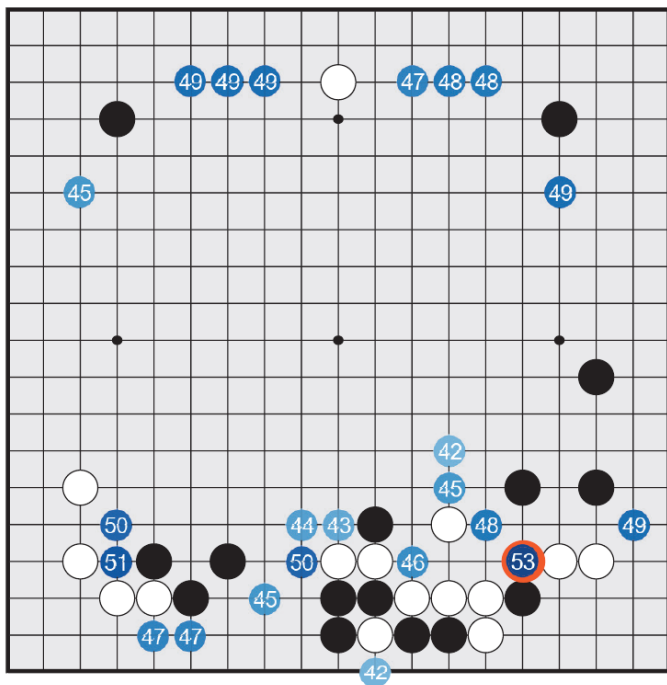
- 蒙特卡洛方法（Monte-Carlo methods）是一类广泛的计算算法
 - 依赖于重复随机抽样来获得数值结果
- 例如，计算圆的面积



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

蒙特卡洛方法

围棋对弈：估计当前状态下的胜率

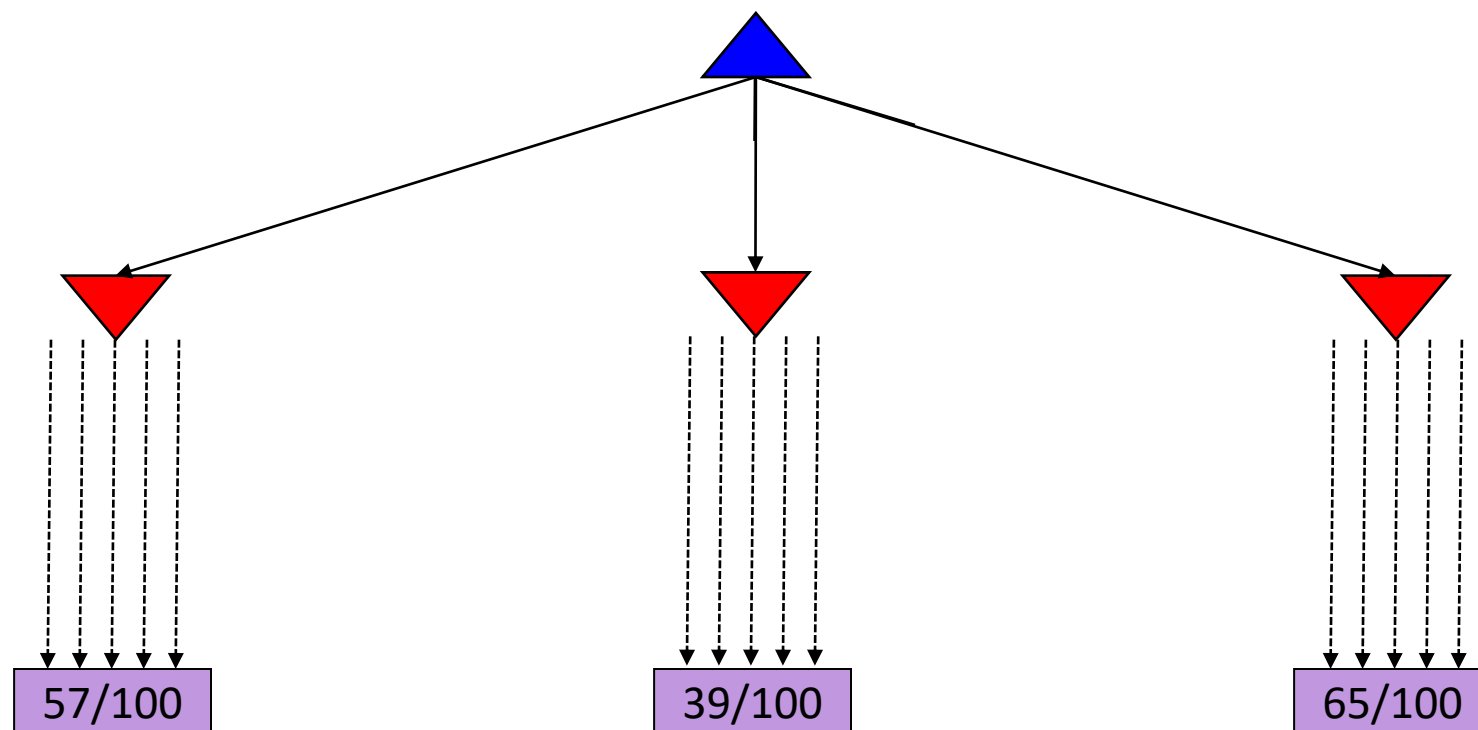


从当前状态出发，做随机模拟

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

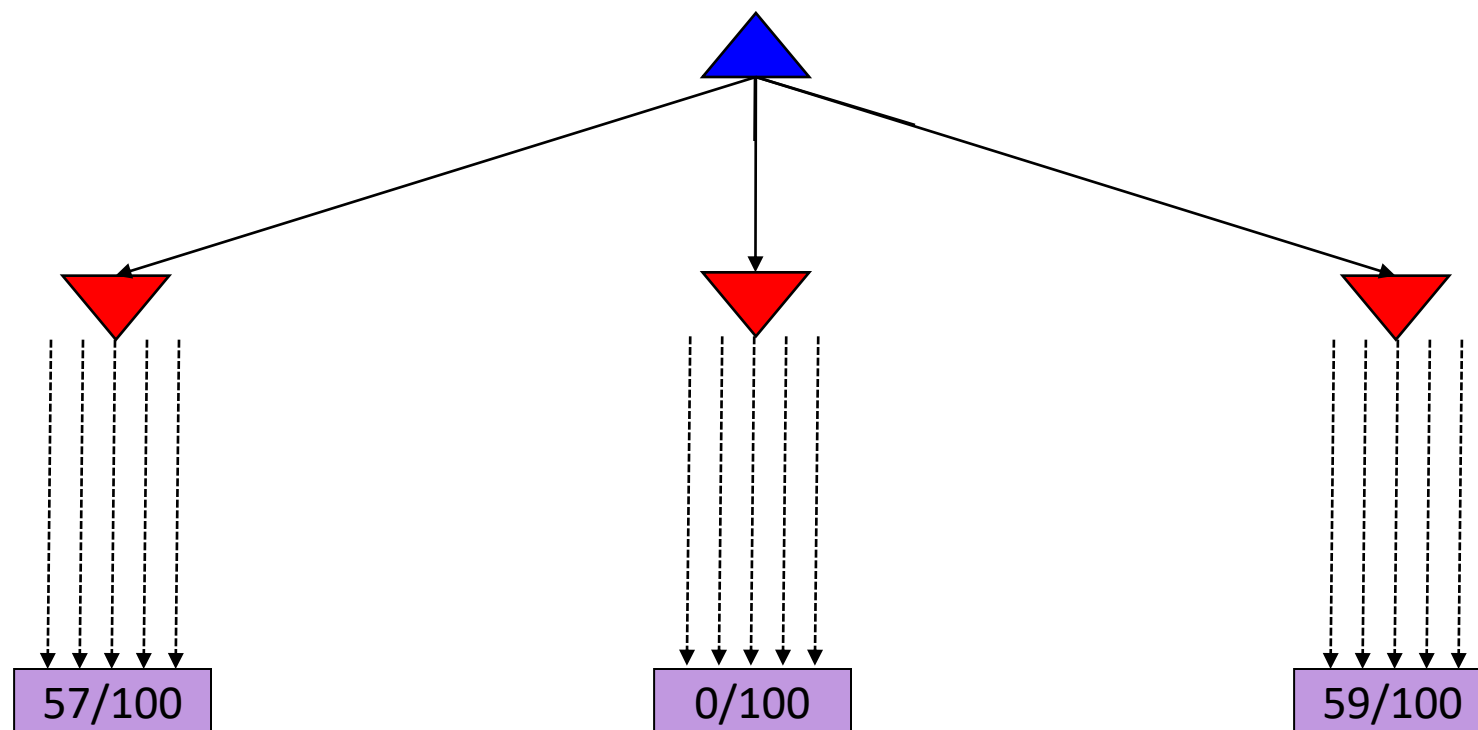
蒙特卡洛搜索

模拟： 从一个节点出发，进行大量模拟，记录赢的次数



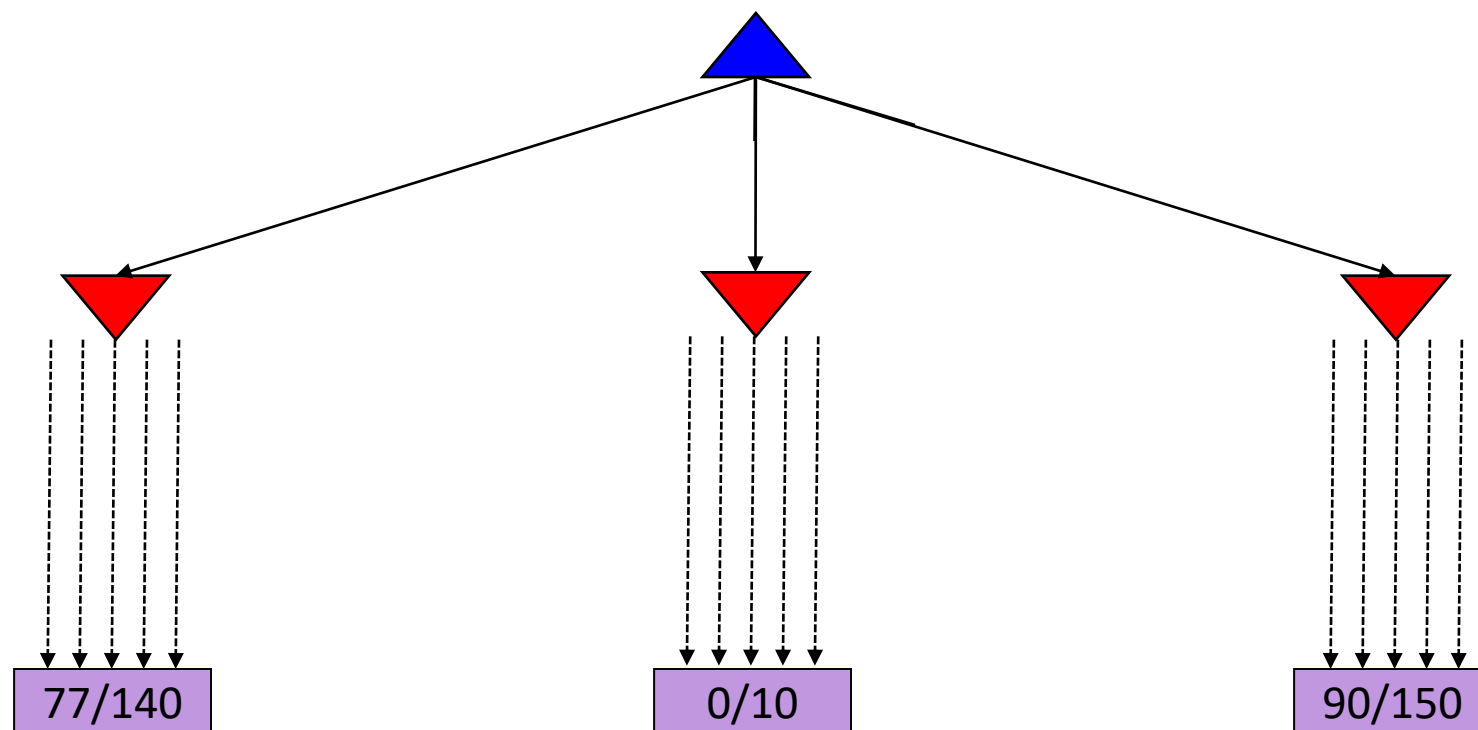
蒙特卡洛搜索

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

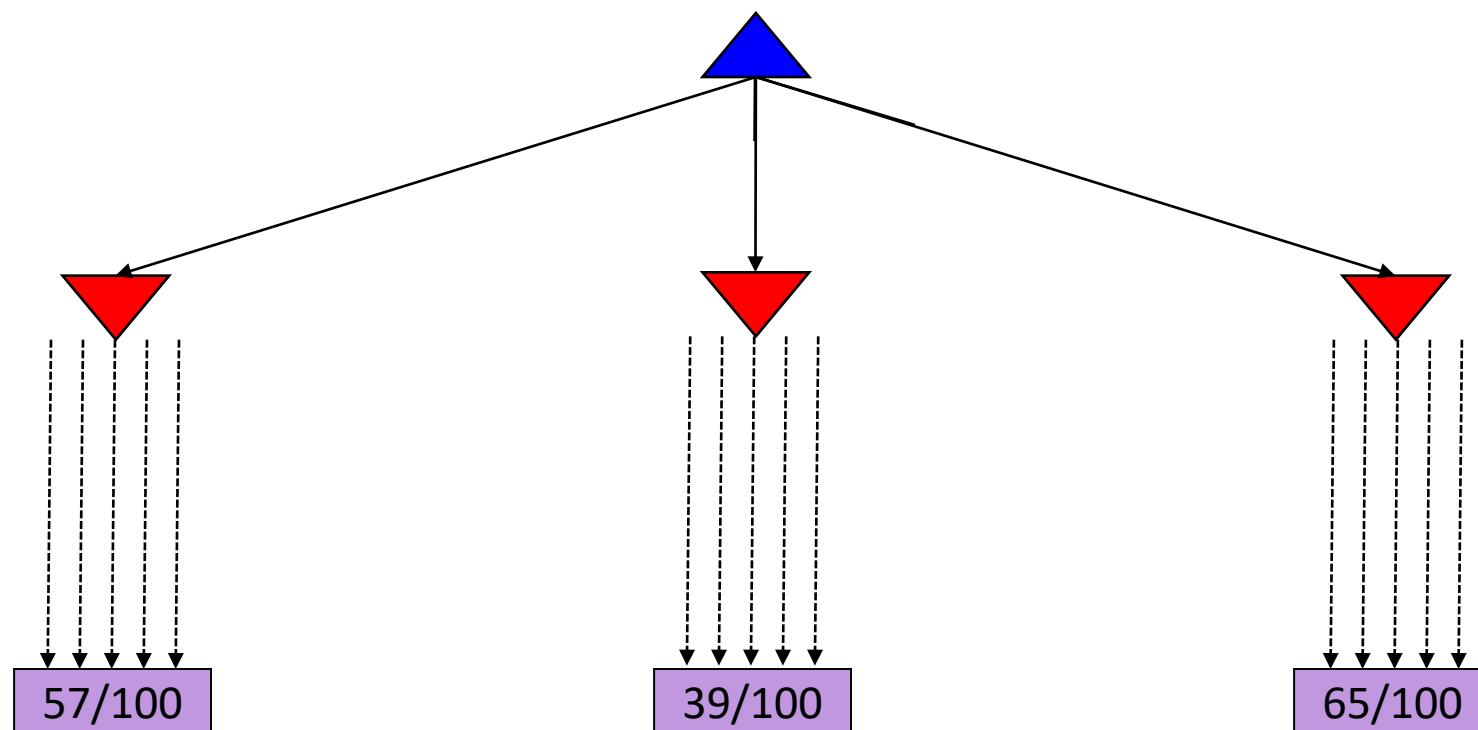
模拟： 从一个节点出发，进行大量模拟，记录赢的次数



蒙特卡洛搜索

资源是有限的，选择哪些节点进行模拟？

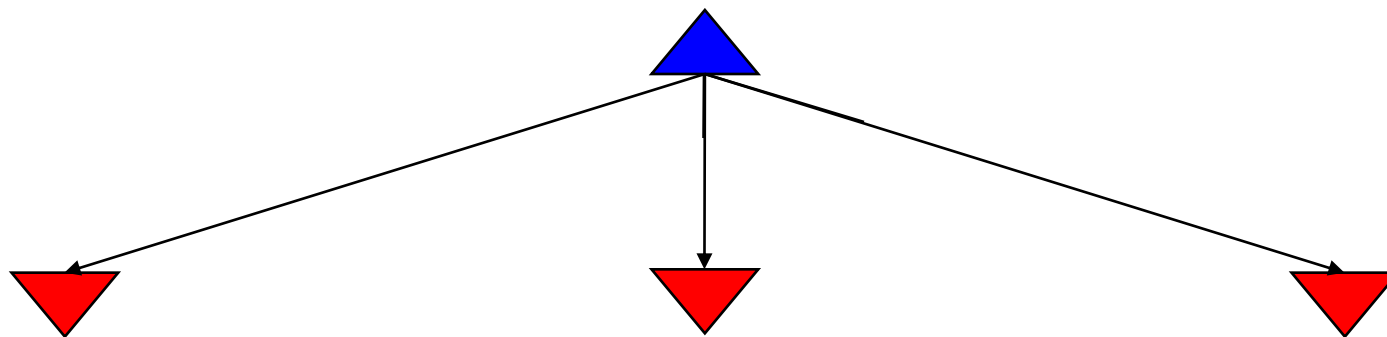
选择更有希望的节点进行模拟



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

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？

探索与利用：

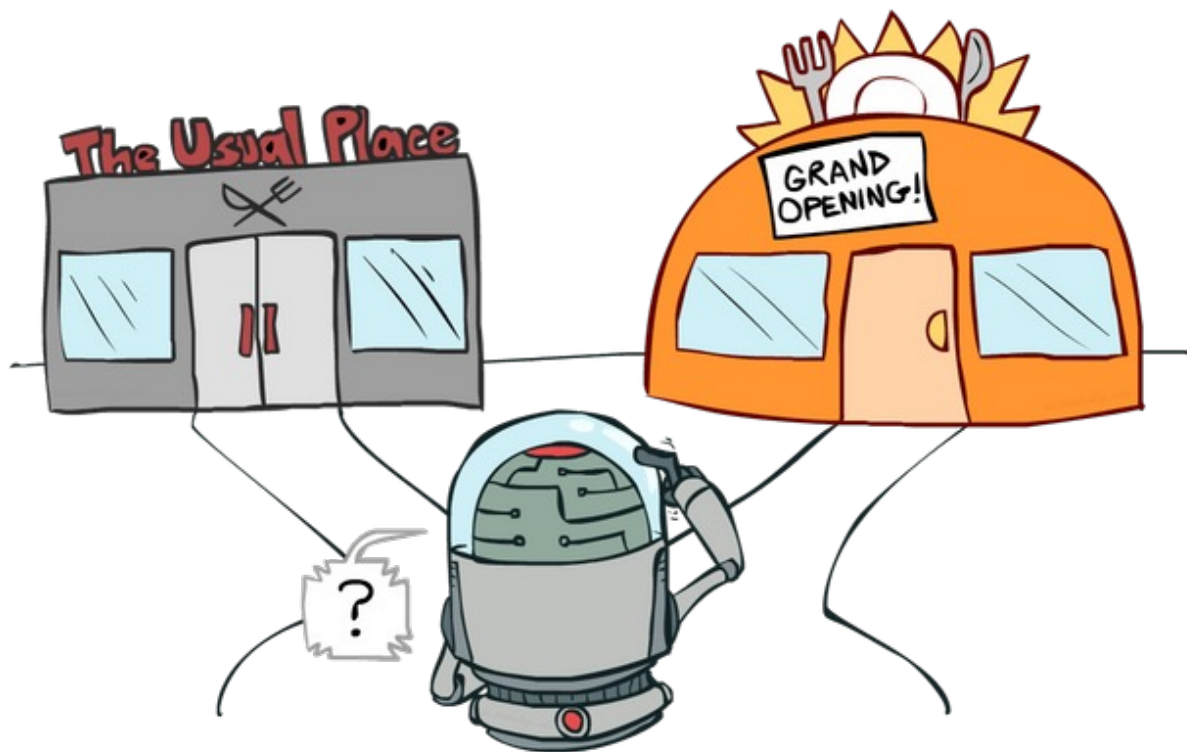
- 探索(Exploration): 估计不同摇臂的优劣 (奖赏期望的大小)
- 利用(Exploitation): 选择当前最优的摇臂



探索-利用

基于目前策略获取已知最优收益还是尝试不同的决策

- Exploitation: 执行能够获得已知最优收益的决策
- Exploration: 尝试更多可能的决策, 不一定会是最优收益



探索-利用窘境

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  $\epsilon$ 。  
  
过程：  
1:  $r = 0$ ；  
2:  $\forall i = 1, 2, \dots, K : Q(i) = 0, \text{count}(i) = 0$ ；  
3: for  $t = 1, 2, \dots, T$  do  
4:   if  $\text{rand}() < \epsilon$  then  
5:      $k =$  从  $1, 2, \dots, K$  中以均匀分布随机选取  
6:   else  
7:      $k = \arg \max_i Q(i)$   
8:   end if  
9:    $v = R(k)$ ；  
10:   $r = r + v$ ；  
11:   $Q(k) = \frac{Q(k) \times \text{count}(k) + v}{\text{count}(k) + 1}$ ；  
12:   $\text{count}(k) = \text{count}(k) + 1$ ；  
13: end for  
输出：累积奖赏  $r$ 
```

Softmax

- Softmax: 基于当前已知的摇臂平均奖赏来对探索与利用折中
 - 若某个摇臂当前的平均奖赏越大, 则它被选择的概率越高
 - 概率分配使用Boltzmann分布:

$$P(k) = \frac{e^{\frac{Q(k)}{\tau}}}{\sum_{i=1}^K e^{\frac{Q(i)}{\tau}}}$$

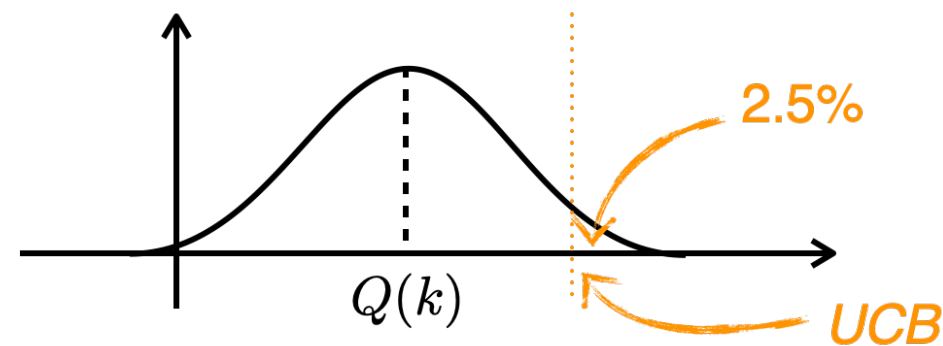
ϵ -贪心与Softmax算法都有一个折中参数 (ϵ, τ) , 算法性能孰好孰坏取决于具体应用问题

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



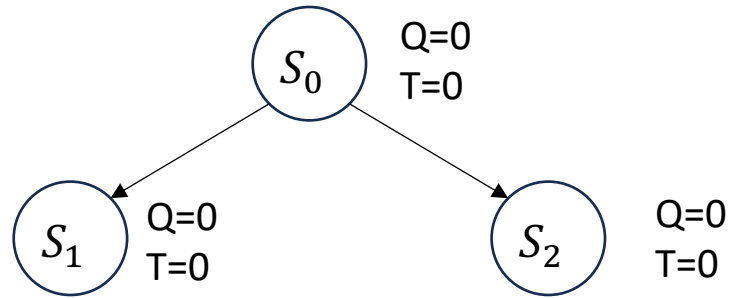
$$UCB(k) = Q_k + C * \sqrt{\ln T / T_k}$$

蒙特卡洛搜索

- **选择 (selection)** : 从根节点开始, 按照某种选择策略 (通常为UCB) 向下选择子节点, 直至到达叶子结点
- **扩展 (expansion)** : 为所选节点生成一个新的子节点 (注意: MCTS有不同实现方式, 有的是每次随机生成一个子节点, 有的是生成所有后继节点再进行选择)
- **模拟 (simulation)** : 从新生成的节点出发出发, 模拟扩展搜索树, 获得模拟结果
- **回溯 (backpropagation)** : 基于模拟结果自底向上更新路径的节点的奖励均值和被访问次数 (注意: 只有胜方的获胜次数和模拟次数都会增加, 败方节点只增加模拟次数)

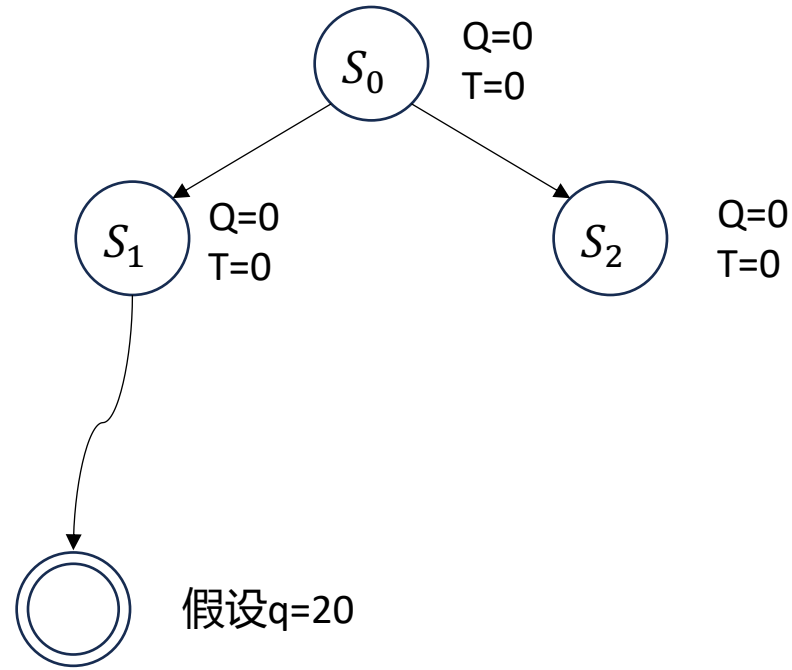
蒙特卡洛搜索 (单Agent版)

$$UCB(k) = Q_k + 2 * \sqrt{\ln T / T_k}$$



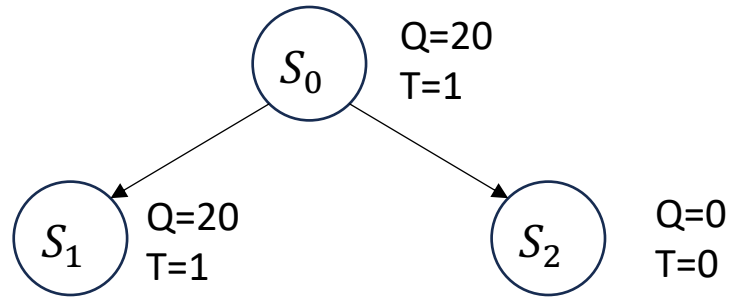
蒙特卡洛搜索 (单Agent版)

$$UCB(k) = Q_k + 2 * \sqrt{\ln T / T_k}$$



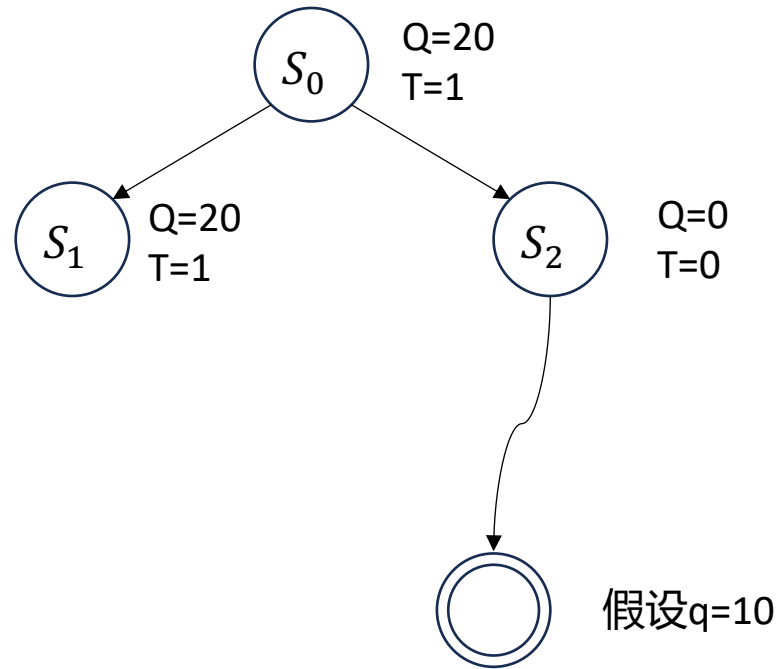
蒙特卡洛搜索 (单Agent版)

$$UCB(k) = Q_k + 2 * \sqrt{\ln T / T_k}$$



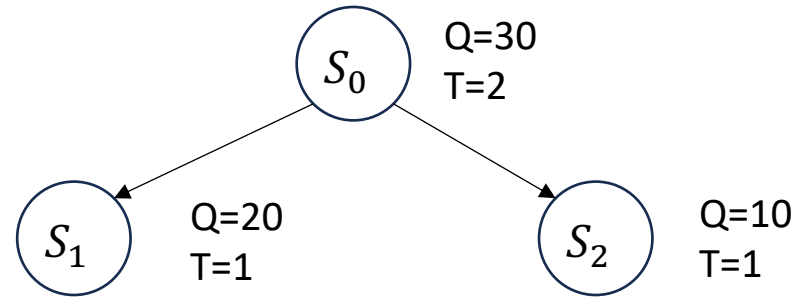
蒙特卡洛搜索 (单Agent版)

$$UCB(k) = Q_k + C * \sqrt{\ln T / T_k}$$



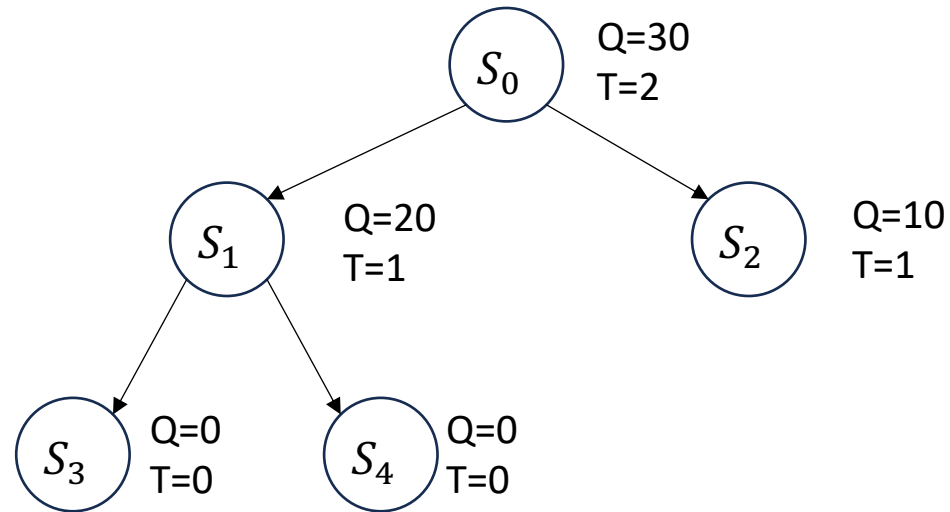
蒙特卡洛搜索 (单Agent版)

$$UCB(k) = Q_k + 2 * \sqrt{\ln T / T_k}$$



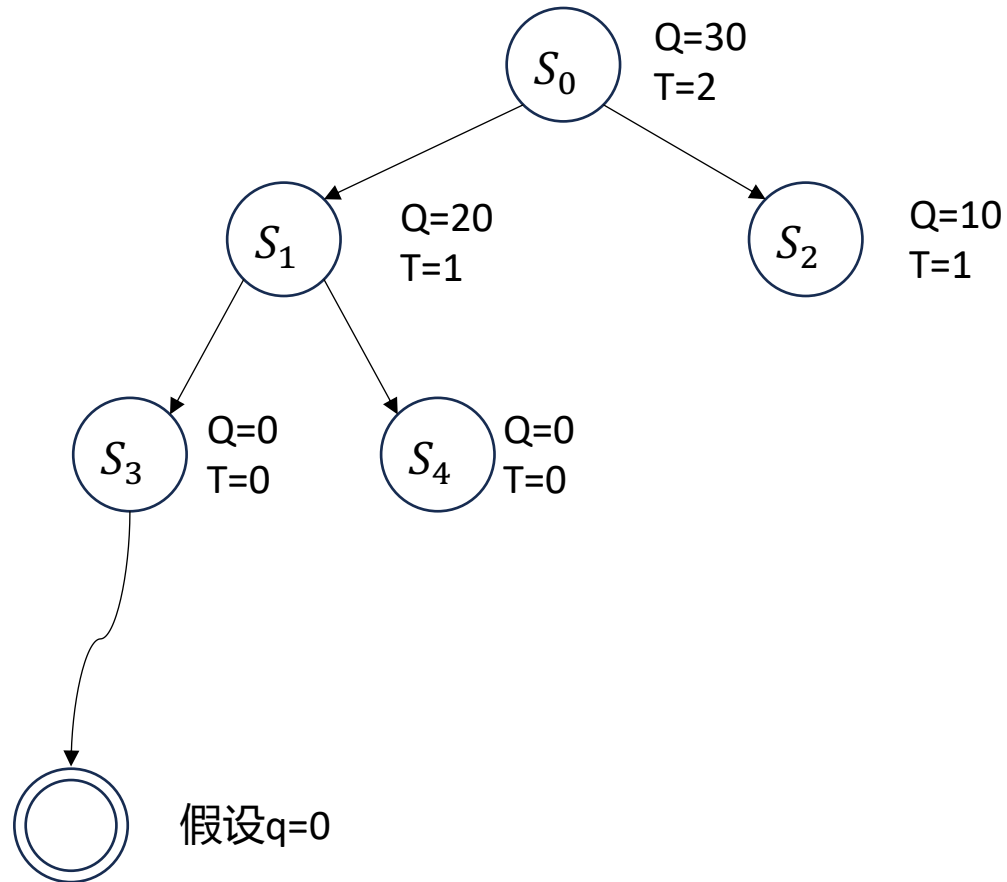
蒙特卡洛搜索 (单Agent版)

$$UCB(k) = Q_k + 2 * \sqrt{\ln T / T_k}$$



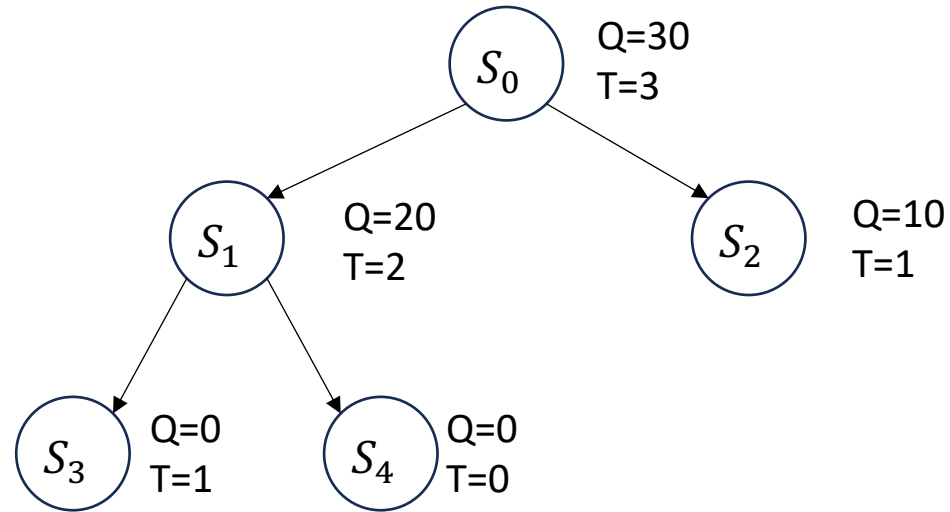
蒙特卡洛搜索 (单Agent版)

$$UCB(k) = Q_k + 2 * \sqrt{\ln T / T_k}$$



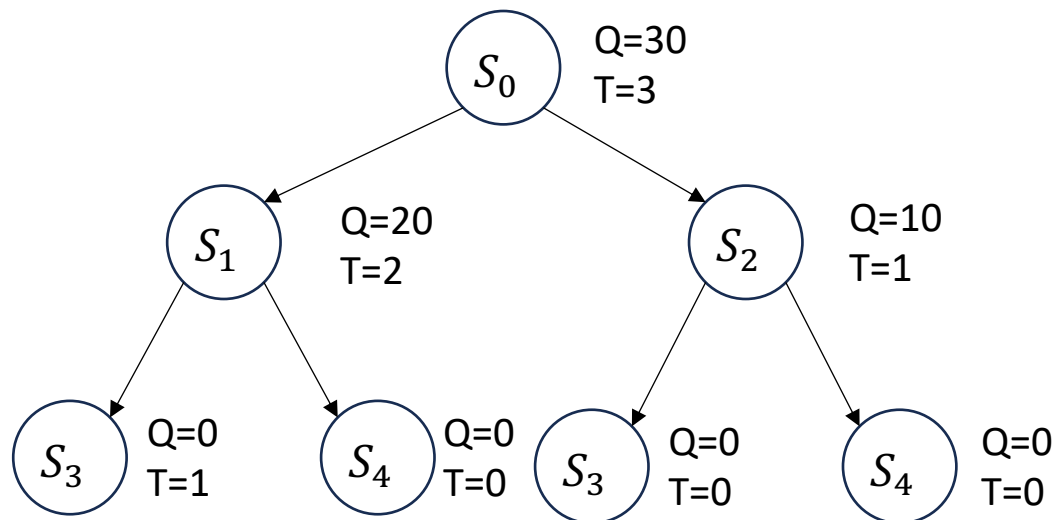
蒙特卡洛搜索 (单Agent版)

$$UCB(k) = Q_k + 2 * \sqrt{\ln T / T_k}$$



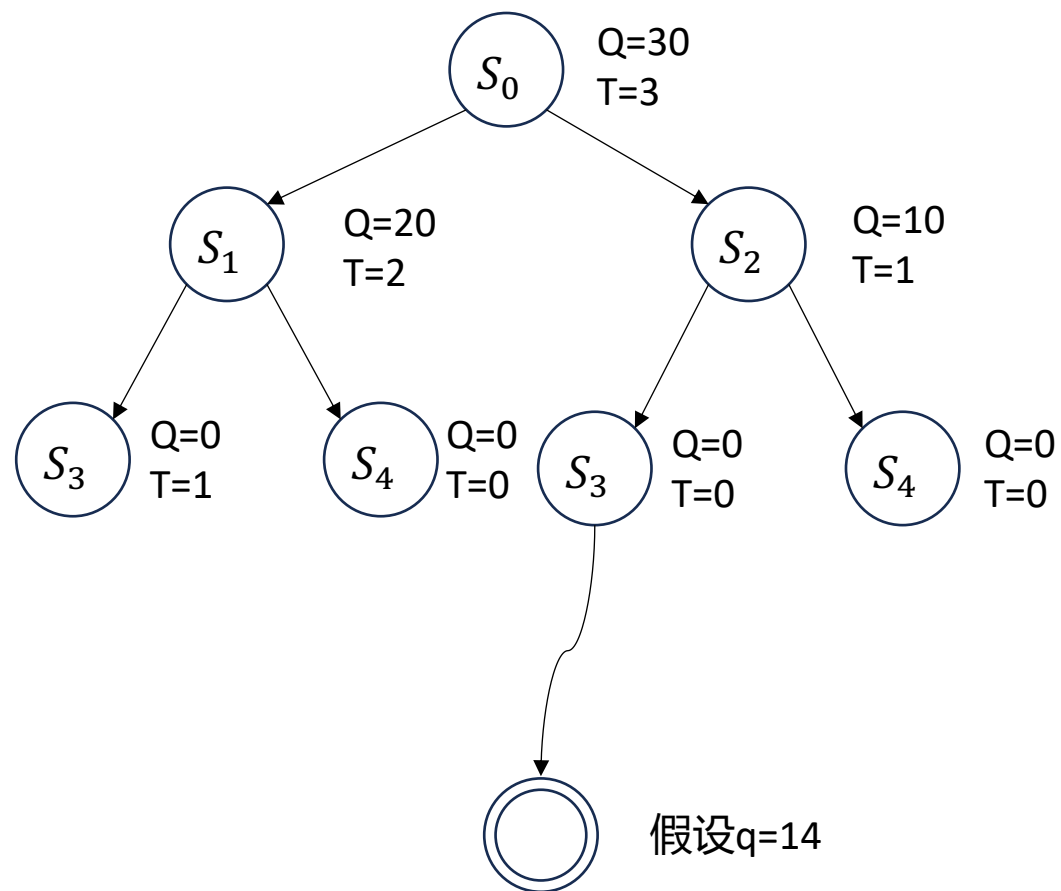
蒙特卡洛搜索 (单Agent版)

$$UCB(k) = Q_k + 2 * \sqrt{\ln T / T_k}$$



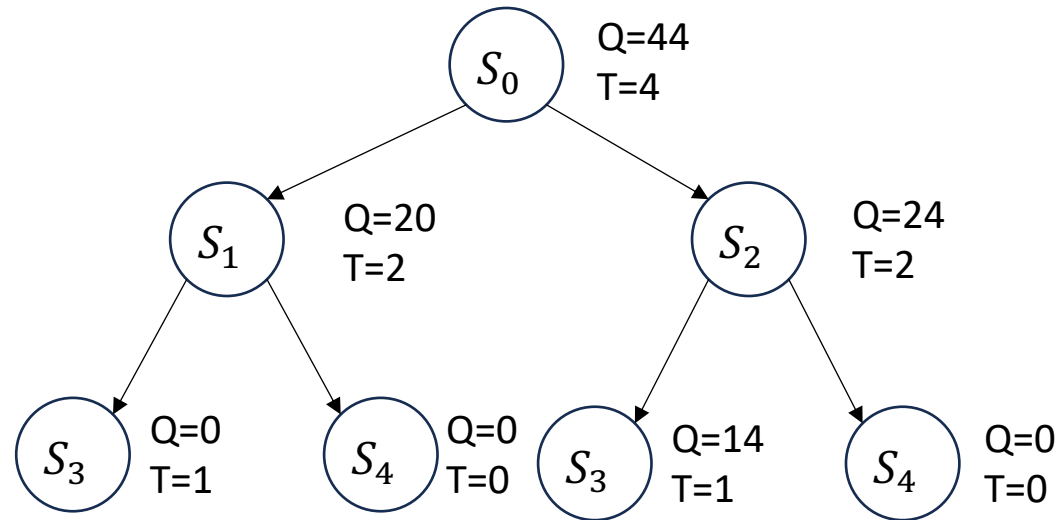
蒙特卡洛搜索 (单Agent版)

$$UCB(k) = Q_k + 2 * \sqrt{\ln T / T_k}$$

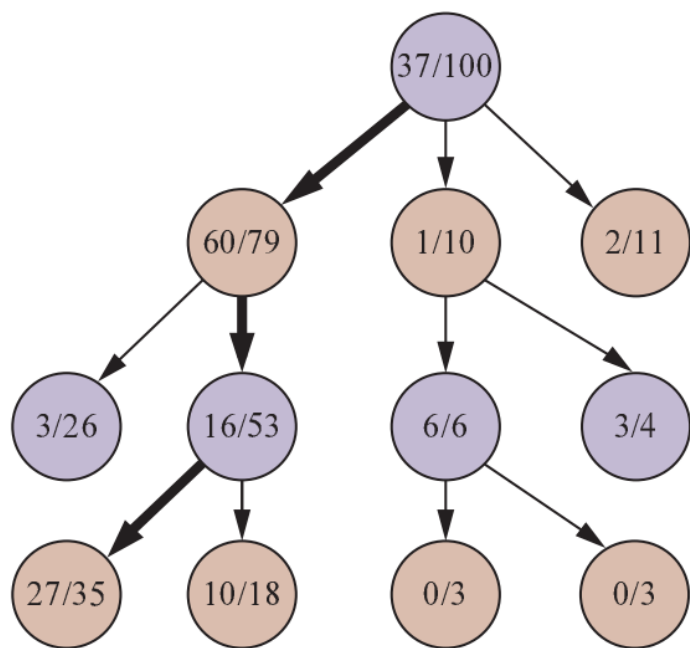


蒙特卡洛搜索 (单Agent版)

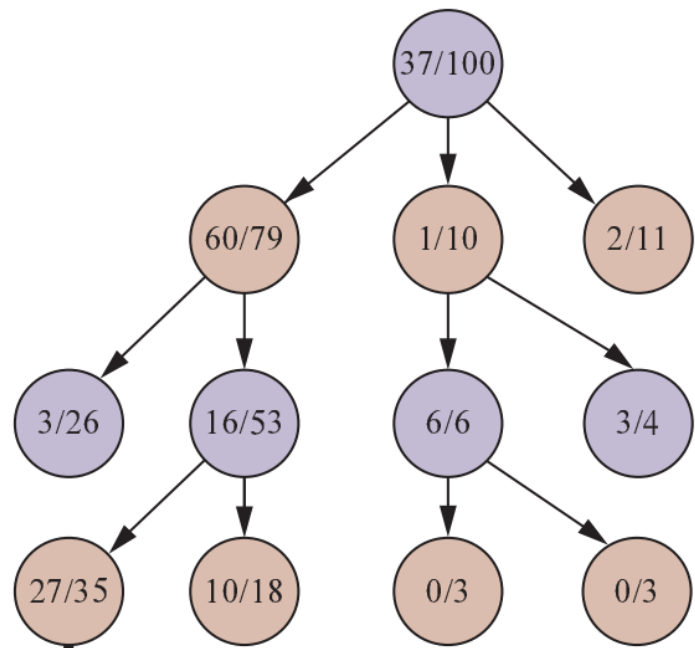
$$UCB(k) = Q_k + 2 * \sqrt{\ln T / T_k}$$



蒙特卡洛搜索 (对抗搜索)

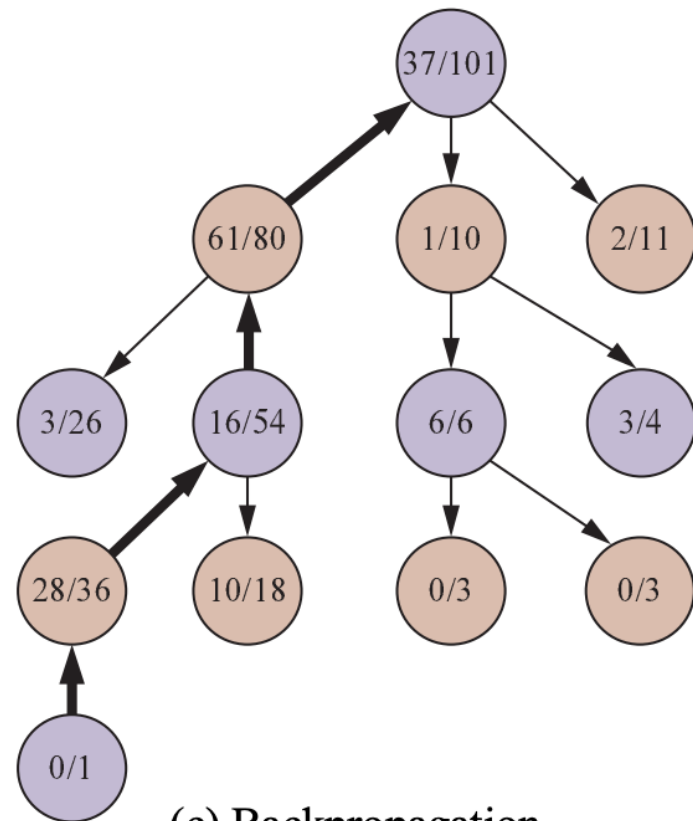


(a) Selection



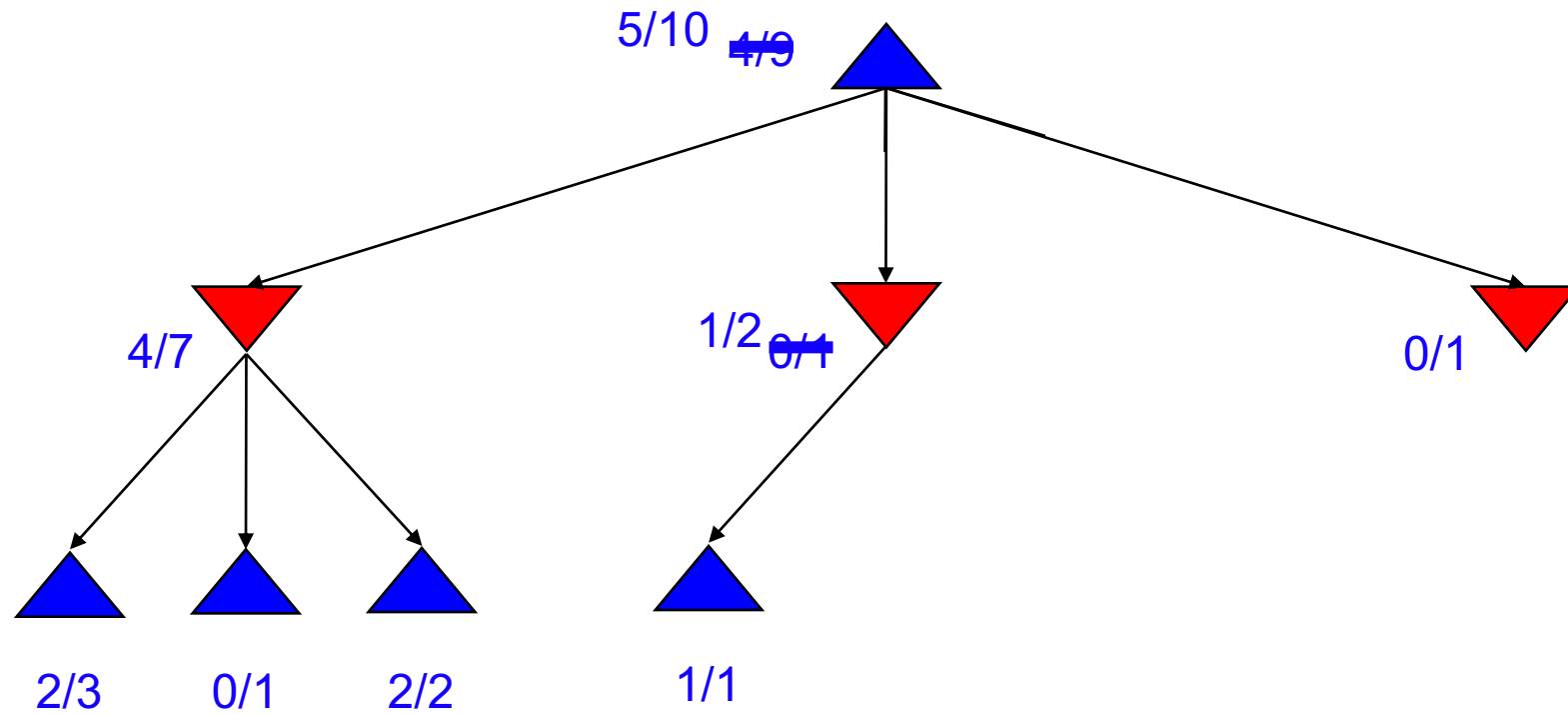
black wins

(b) Expansion and simulation



(c) Backpropagation

蒙特卡洛搜索 (对抗搜索)



MCTS到AlphaGo

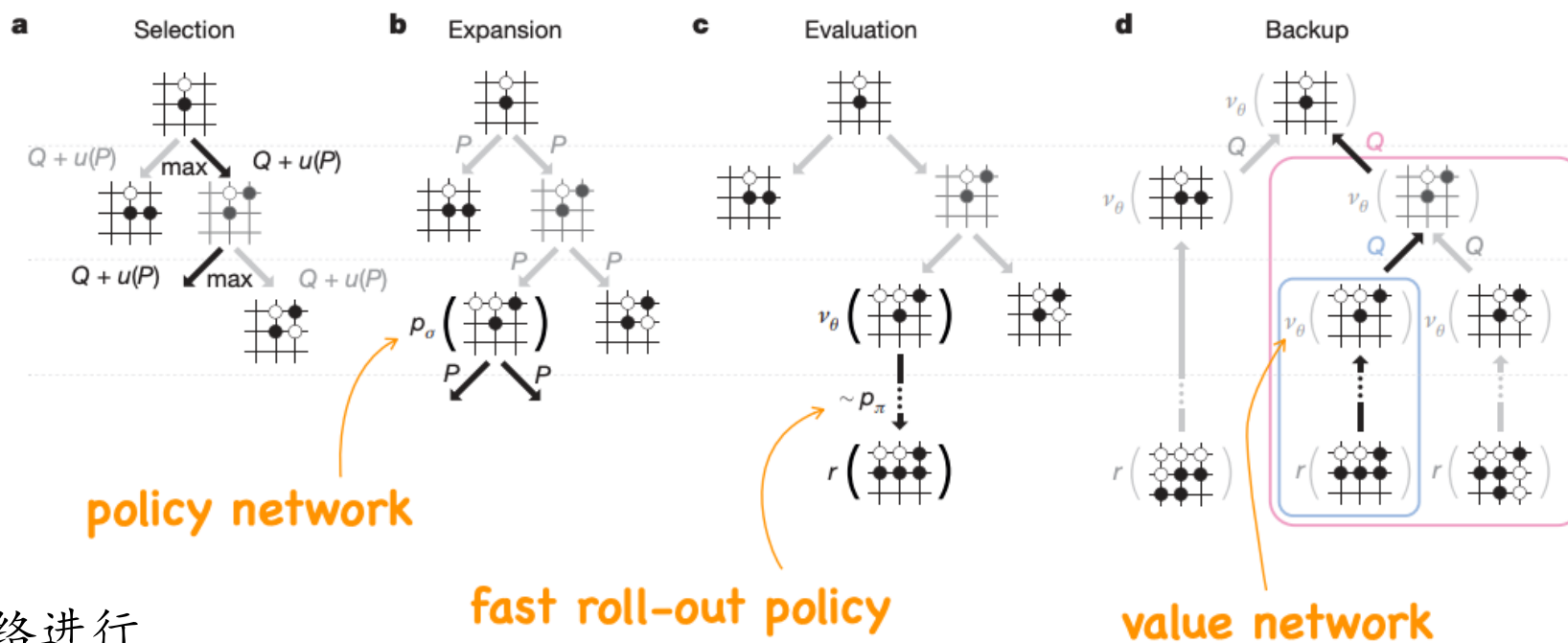
MCTS通过采样搜索和模拟评估，让围棋可以通过搜索算法来解决

但是，

- 搜索空间仍然很大，每一步需要考虑所有的可能动作
- 仍然需要模拟到终局，时间开销非常庞大

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

Mastering the game of Go with deep neural networks and tree search

David Silver^{1*}, Aja Huang^{1*}, Chris J. Maddison¹, Arthur Guez¹, Laurent Sifre¹, George van den Driessche¹, Julian Schrittwieser¹, Ioannis Antonoglou¹, Veda Panneershelvam¹, Marc Lanctot¹, Sander Dieleman¹, Dominik Grewe¹, John Nham², Nal Kalchbrenner¹, Ilya Sutskever², Timothy Lillicrap¹, Madeleine Leach¹, Koray Kavukcuoglu¹, Thore Graepel¹ & Demis Hassabis¹

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses ‘value networks’ to evaluate board positions and ‘policy networks’ to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game, from every board position or state s , under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game’s breadth (number of legal moves per position) and d is its depth (game length). In large games, such as chess ($b \approx 35$, $d \approx 80$)¹ and especially Go ($b \approx 250$, $d \approx 150$)¹, exhaustive search is infeasible^{2,3}, but the effective search space can be reduced by two general principles. First, the depth of the search may be reduced by position evaluation: truncating the search tree at state s and replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s . This approach has led to superhuman performance in chess⁴, checkers⁵ and othello⁶, but it was believed to be intractable in Go due to the complexity of the game⁷. Second, the breadth of the search may be reduced by sampling actions from a policy $p(a|s)$ that is a probability distribution over possible moves a in position s . For example, Monte Carlo rollouts⁸ search to maximum depth without branching at all, by sampling long sequences of actions for both players from a policy p . Averaging over such rollouts can provide an effective position evaluation, achieving superhuman performance in backgammon⁸ and Scrabble⁹, and weak amateur level play in Go¹⁰.

Monte Carlo tree search (MCTS)^{11,12} uses Monte Carlo rollouts to estimate the value of each state in a search tree. As more simulations are executed, the search tree grows larger and the relevant values become more accurate. The policy used to select actions during

policies^{13–15} or value functions¹⁶ based on a linear combination of input features.

Recently, deep convolutional neural networks have achieved unprecedented performance in visual domains: for example, image classification¹⁷, face recognition¹⁸, and playing Atari games¹⁹. They use many layers of neurons, each arranged in overlapping tiles, to construct increasingly abstract, localized representations of an image²⁰. We employ a similar architecture for the game of Go. We pass in the board position as a 19×19 image and use convolutional layers to construct a representation of the position. We use these neural networks to reduce the effective depth and breadth of the search tree: evaluating positions using a value network, and sampling actions using a policy network.

We train the neural networks using a pipeline consisting of several stages of machine learning (Fig. 1). We begin by training a supervised learning (SL) policy network p_π directly from expert human moves. This provides fast, efficient learning updates with immediate feedback and high-quality gradients. Similar to prior work^{13,15}, we also train a fast policy p_π that can rapidly sample actions during rollouts. Next, we train a reinforcement learning (RL) policy network p_ρ that improves the SL policy network by optimizing the final outcome of games of self-play. This adjusts the policy towards the correct goal of winning games, rather than maximizing predictive accuracy. Finally, we train a value network v_θ that predicts the winner of games played by the RL policy network against itself. Our program AlphaGo efficiently combines the policy and value networks with MCTS.

Mastering the game of Go without human knowledge

David Silver^{1*}, Julian Schrittwieser^{1*}, Karen Simonyan^{1*}, Ioannis Antonoglou¹, Aja Huang¹, Arthur Guez¹, Thomas Hubert¹, Lucas Baker¹, Matthew Lai¹, Adrian Bolton¹, Yutian Chen¹, Timothy Lillicrap¹, Fan Hui¹, Laurent Sifre¹, George van den Driessche¹, Thore Graepel¹ & Demis Hassabis¹

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo’s own move selections and also the winner of AlphaGo’s games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting *tabula rasa*, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.

Much progress towards artificial intelligence has been made using supervised learning systems that are trained to replicate the decisions of human experts^{1–4}. However, expert data sets are often expensive, unreliable or simply unavailable. Even when reliable data sets are available, they may impose a ceiling on the performance of systems trained in this manner⁵. By contrast, reinforcement learning systems are trained from their own experience, in principle allowing them to exceed human capabilities, and to operate in domains where human expertise is lacking. Recently, there has been rapid progress towards this goal, using deep neural networks trained by reinforcement learning. These systems have outperformed humans in computer games, such as Atari^{6,7} and 3D virtual environments^{8–10}. However, the most challenging domains in terms of human intellect—such as the game of Go, widely viewed as a grand challenge for artificial intelligence¹¹—require a precise and sophisticated lookahead in vast search spaces. Fully general methods have not previously achieved human-level performance in these domains.

AlphaGo was the first program to achieve superhuman performance in Go. The published version¹², which we refer to as AlphaGo Fan, defeated the European champion Fan Hui in October 2015. AlphaGo Fan used two deep neural networks: a policy network that outputs move probabilities and a value network that outputs a position eval-

uated solely by self-play reinforcement learning, starting from random play, without any supervision or use of human data. Second, it uses only the black and white stones from the board as input features. Third, it uses a single neural network, rather than separate policy and value networks. Finally, it uses a simpler tree search that relies upon this single neural network to evaluate positions and sample moves, without performing any Monte Carlo rollouts. To achieve these results, we introduce a new reinforcement learning algorithm that incorporates lookahead search inside the training loop, resulting in rapid improvement and precise and stable learning. Further technical differences in the search algorithm, training procedure and network architecture are described in Methods.

Reinforcement learning in AlphaGo Zero

Our new method uses a deep neural network f_θ with parameters θ . This neural network takes as an input the raw board representation s of the position and its history, and outputs both move probabilities and a value, $(\mathbf{p}, v) = f_\theta(s)$. The vector of move probabilities \mathbf{p} represents the probability of selecting each move a (including pass), $p_a = \Pr(a|s)$. The value v is a scalar evaluation, estimating the probability of the current player winning from position s . This neural network combines the roles of both policy network and value network¹² into a single architecture.

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)

大模型能胜任博弈AI任务吗？

我们来下五子棋吧

我下在：👉 H7 (正上方，先抢中心控制)

现在轮到你了 ●

直接说坐标即可 (比如 G8 / H9 / I8 都是常见选择)

📄 🔊 🗨️ 🔄 ...

现在轮到你 ●

你可以考虑继续做“三连”，比如 F8 或 H9 🙄

📄 🔊 🗨️ 🔄 ...

F8

G8

很好！你已经形成三连 (F8-G8-H8)，很有进攻性 🙌

当前棋盘：

id="wiy6d5"

📄 复制

	D	E	F	G	H	I	J	K	L
6
7	●
8	.	.	●	●	●	●	.	.	.
9
10

好啊，来！ 😊

我们用简单坐标来下 (类似围棋棋盘)。

🎮 规则说明 (简化版)

• 棋盘：15×15 (也可以先用 10×10)

• 用坐标表示：

• 横轴：A B C D E F G H I J ...

• 纵轴：1 2 3 4 5 ...

👉 例如：

• “H8” 表示中间位置

🧩 约定

• 你：● (黑子，先手)

• 我：○ (白子)

🚀 开始吧！

请你先下第一步，直接说一个坐标，比如：

好棋，开始有进攻意图了 🙌

当前棋盘：

id="s0k0s3"

📄 复制

	D	E	F	G	H	I	J	K	L
6
7	○
8	.	.	●	●
9
10

我这一步下：👉 I8

👉 目的：

• 防止你横向发展 (G8-H8-?)

• 同时保持对中线的控制

我必须防守了，这步下：👉 E8

👉 否则你下一步 D8 就成活四 (必胜)

现在轮到你 ●

你已经有不错的先手优势，可以考虑转向纵向或斜线进攻，比如 H9 / G9 等 🙄

📄 🔊 🗨️ 🔄 ...

↓

H9

大模型能胜任博弈AI任务吗？

给定王者荣耀对局中的实时盘面信息，作为主玩家的助手，给出决策建议。# 盘面信息 <game_state> </game_state> # 思考的要点：盘面理解（英雄信息、发育状态、兵线态势、防御塔状态、野区资源、局面感知、视野情况等等）阵容与策略（英雄/阵容特点、强势弱势期、个人责任、风险收益平衡）实时动态时机观察（局势顺逆僵持、双方动向意图、交战情况、资源取舍、战术选择、追击逃跑）特殊场景（顺风、逆风、关键资源抢夺）、特殊英雄机制与特殊战术# 建议选择思考给出决策意见后，再从不互斥的候选选项中选出最接近的1-2个建议。思考过程放入 <think> </think>, 行动建议使用", "分隔放入 <answer> </answer>。候选选项有 <action_candidates> </action_candidates>

Game State after Rendering:	Model Response:
	<p><think> 我需要分析当前盘面，帮助玩家扁鹊做出最佳决策。首先，观察到暗影主宰已刷新在主宰龙坑，且正受到攻击，这可能是由我方铠在处理。由于主宰对团队增益重大，优先处理它能扭转局势。其次，敌方马可波罗在我方蓝区，但距离较远，且扁鹊当前位置到主宰龙坑较近，移动方便。最后，确保没有其他紧急威胁，如敌方英雄接近或需要防守的塔，因此建议立即前往主宰龙坑协助。</p> <p></think></p> <p><result> 协助铠拿下暗影主宰，获取团队增益。</p> <p></result></p>

大模型能胜任博弈AI任务吗？

werewolf **villager** **witch** **guard** **seer**

1 Good morning everyone. It seems like we had a peaceful night. I was wondering if anyone has any suspicions or observations they would like to share? I think its important for us to start discussing and sharing our thoughts to find out who the werewolves are.

2 Good morning everyone. As a villager, I dont have any clear suspicions yet, but I'm interested in hearing what others have to say. Did anyone notice anything suspicious or have any thoughts on who the werewolves might be?

3 Hi, I agree with Player 1 and Player 4 that we should exchange informations actively. Based on my observation, I think Player 5 is a little suspicious.

4 Well, I agree with Player 1, 3 and Player 4. Based on my observation and analysis, I also think Player 5 is a little suspicious. But we might not vote to kill him casually.

5 I have to defend myself because I am innocent. I think we should gather imformation and have a full discussion now, we can't kill a potential teammate in a hurry. On the contrary, I think Player 3 is a bit aggressive, and I suspect he is one of the werewolf.

6 I observed that Player 5 was killed last night. As a witch, I used my antidote to save him. Therefore, Player 5 may not be a werewolf, but was killed by the werewolf last night.

7 I agree with Player1,2 and 6. I think we have to actively gather information and seriously speculate about who the werewolves are. We should protect ourselves as much as possible and avoid being killed by the werewolves. I will use my special ability wisely at night.

大模型能胜任博弈AI任务吗？



Adventure



Competition



Communication



Cooperation



Simulation



Crafting & Exploration

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, UCS
- 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、演化搜索
- 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，如黑白棋、五子棋、象棋等