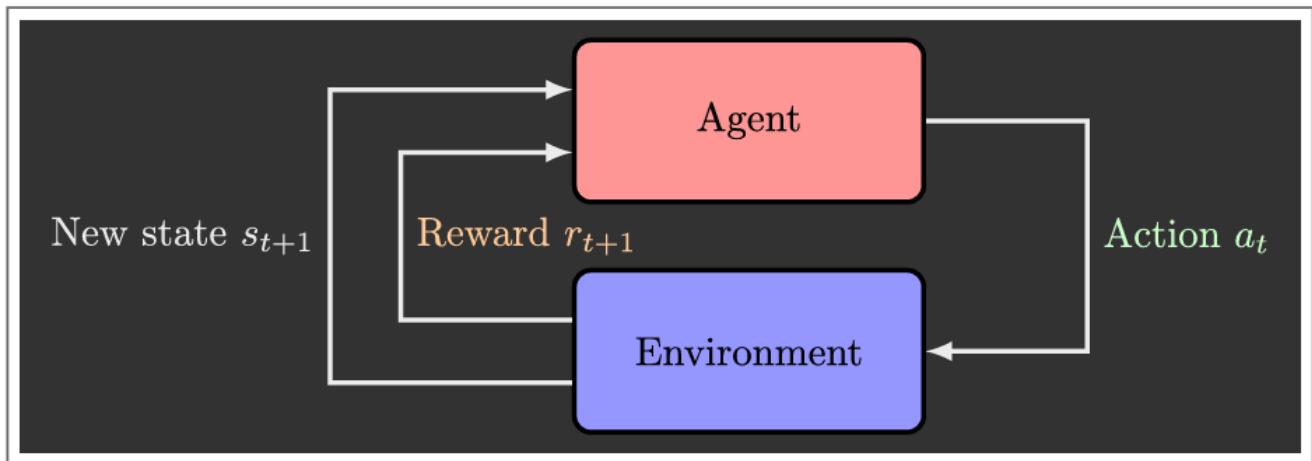


# 1. Search Problems

## Rational Agents



New state: Objective scenarios after environmental updates (e.g., new position of vehicles in autonomous driving)

**新状态:** 环境更新后的客观情形 (如自动驾驶中车辆新位置)

Reward: Immediate numerical feedback from the environment on the action (e.g. +1 for successful obstacle avoidance, -10 for collision)

**奖励:** 环境对动作的即时数值反馈 (如成功避障+1, 碰撞-10)

An **agent** is an entity that perceives(by sensors) and acts(by actuators), like auto-driven cars

代理是一个能感知(传感器)和行动(执行器)的实体，就像自动驾驶汽车一样

A **rational agent** selects actions that maximize its (expected) utility.

理性的代理会选择使其（预期）效用最大化的行动。

Characteristics of the percepts, environment, and action space dictate techniques for selecting rational actions.

感知、环境和行动空间的特点决定了选择合理行动的技巧。

Rational agents are **not omniscient**.(can only acquire limited environment information by sensor)

理性代理**并非无所不知** (只能通过传感器获取有限的环境信息)

Rational agents are **not clairvoyant**.(can not predict future state transfer)

理性代理**不是千里眼** (不能预测未来的状态转移)

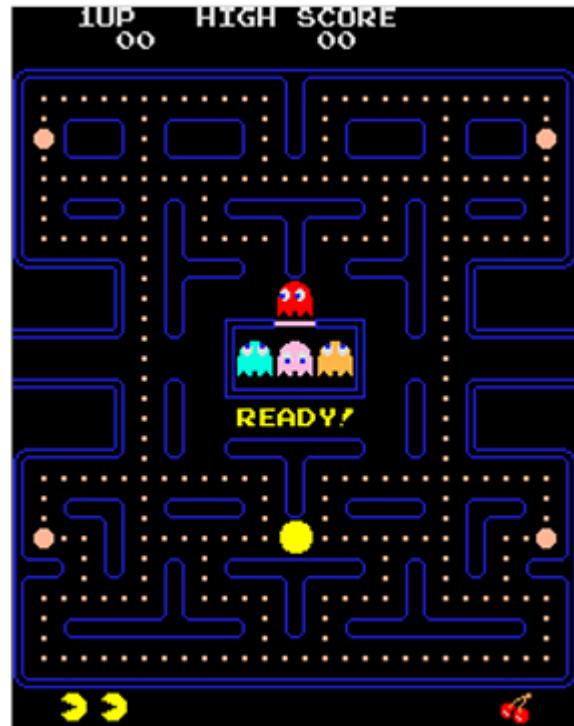
Rational agents can **explore** and **learn**.

理性代理可以**探索和学习**。

So, rational agents are not necessarily successful, but they are **autonomous**.

因此，理性代理不一定成功，但他们是**自主的**。

Examples: Pacman vs. Robotaxi



# Environment categorization

	Pacman	Robotaxi
Fully or Partially Observable	fully (whole map visible)	partial (Sensors have blind spots)
Single or Multi Agent	multi (Pacman + 4 ghosts)	multi (Other vehicles/pedestrians)
Deterministic or Stochastic	deterministic (action results predictable)	stochastic 随机性 (unpredictable accidents)
Static or Dynamic	static (only agents' action change the environment)	dynamic (Active changes in external factors)
Discrete or Continuous	discrete(Gridded Mobility)	continuous

Can a **reflex agent** be rational?

which:

- Choose actions based on current observation (and maybe memory) 基于当前感知（可能结合短期记忆）的即时反应
- May have memory or a model of the world's current state 记忆范围仅限当前状态，不包含未来预测
- Do not consider future consequences of actions 无长期价值函数计算
- Consider how the world IS as opposed to how it would be 拒绝状态预测 (no forward simulation) 仅响应已发生的环境变化

Yes when: Fully observable and static environment, Single-step movements directly maximize utility, No need to deal with delayed rewards/long-term impacts

是的，什么时候？ 完全可观察的静态环境、单步运动直接实现效用最大化、无需处理延迟奖励/长期影响

## AGENTS THAT PLAN AHEAD 计划先行代理

- Decision based on predicted consequences of actions 通过预测动作的长期后果进行决策的智能体
- Must have a transition model : how the world evolves in responses to actions 必须有一个过渡模型：预测世界如何随着行动而演变
- Must formulate a goal 必须制定目标
- Consider how the world WOULD BE as opposed to how it is 考虑世界本应如何，而不是目前如何

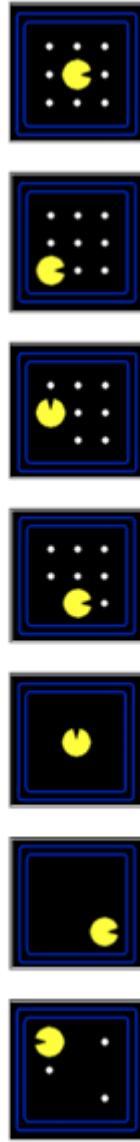
## Spectrum of Deliberativeness: 慎思范围

- Generate complete, optimal plan offline, then execute 离线生成完整、优化的计划，然后执行
- Generate simple, greedy plan online, start executing, replan if necessary 在线生成简单、贪婪的计划，开始执行，必要时重新扫描

# Search Problems

A search problem consists of:

- A state space: The set of all possible states 所有可能状态的集合

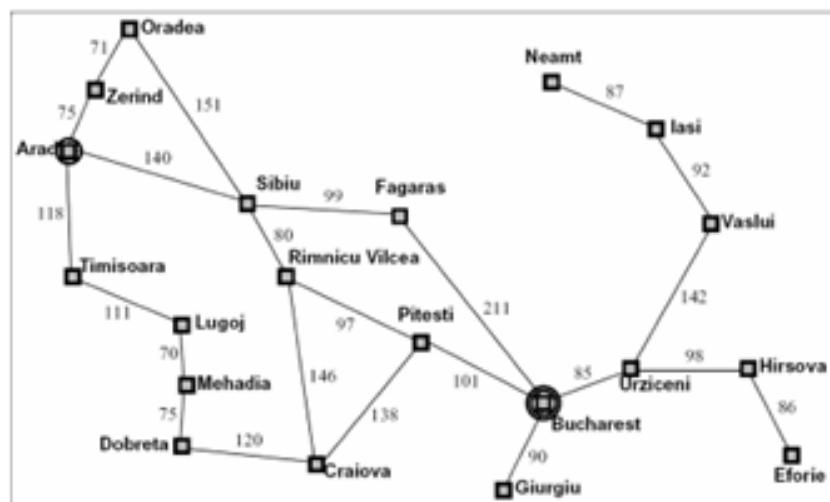


- A successor function 后继函数: Define how the action changes state (action, costs) 定义动作如何改变状态
- Start state 起始状态
- Goal Test 目标测试: Conditions for determining whether a state is a target 判断状态是否为目标的条件

A solution is a sequence of actions (a plan) which transforms the start state to the goal test.

解决方案是将起始状态转换为目标测试的一系列行动（计划）。

Example: TRAVELING IN ROMANIA



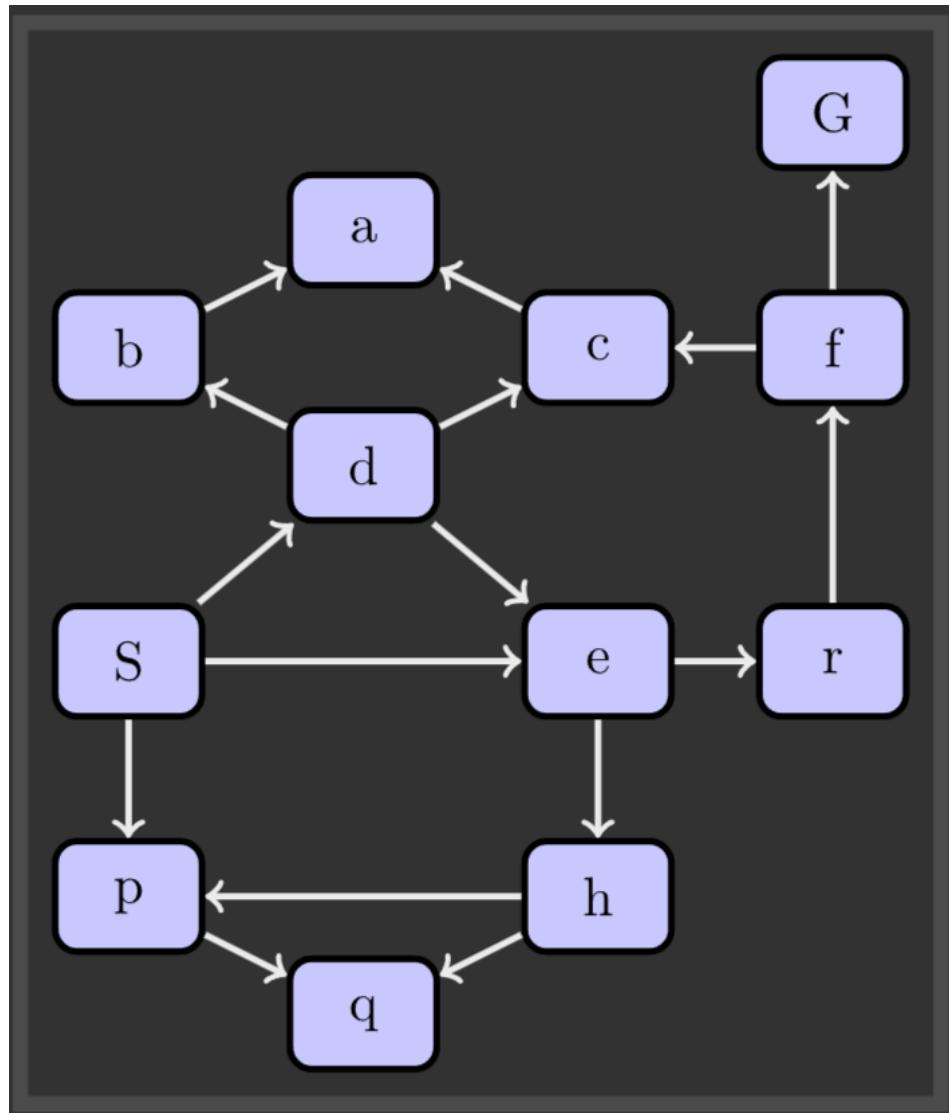
- State Space: Cities
- Successor Function: Roads: travel to adjacent city with cost=distance
- Start Space: Arad
- Goal Test: Is state == Bucharest?
- Solution? Any path that goes to Bucharest

## 2. Uninformed Search

### State Space Graphs

State Space Graph: A mathematical representation of a search problem

状态空间图: 搜索问题的数学表示



- Nodes: (abstracted) world configurations 节点: (抽象的) 世界配置
- Arcs: successors (action results) 弧: 后继 (行动结果)
- Goal Nodes: a set of nodes satisfying goal test

**Hard** to build full graph

### Search tree

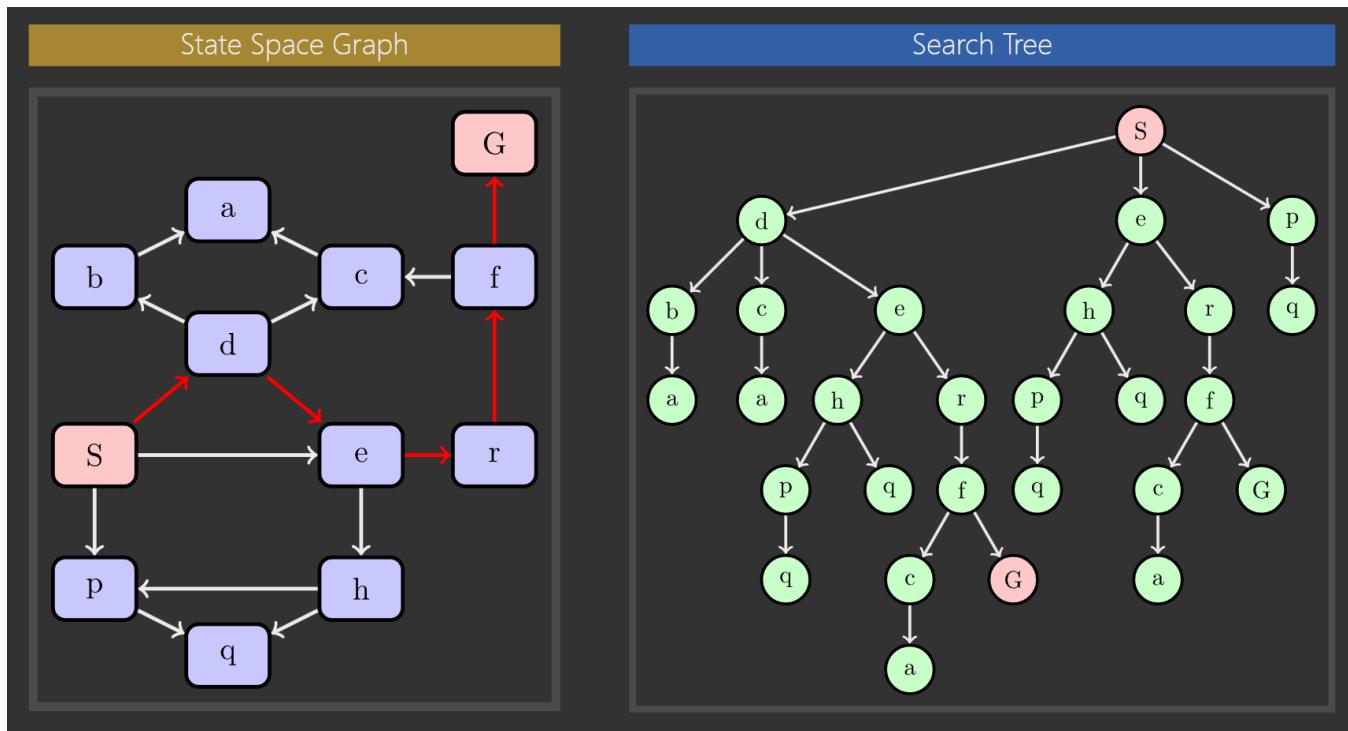
A "what if" tree of plans and their outcomes 计划及其结果的 "假设" 树

- Root Node: The start state 根节点: 起始状态
- Child nodes: action results corresponding to successors 与后续行动相对应的行动结果

Nodes show states, but correspond to plans/actions that achieve those states 节点显示状态，但与实现这些状态的计划/行动相对应

**Impossible** for most problems to build the whole tree.

对于大多数问题来说，**不可能建立整棵树。**



The state space graph is a global abstraction of the problem, revealing the relationships of all states; the search tree is the execution trace of the algorithm, recording the actual paths explored.

状态空间图是问题的全局抽象，揭示所有状态的关系；搜索树是算法的执行痕迹，记录实际探索的路径。

Each NODE in search tree is an entire PATH in state space graph

搜索树中的每个节点都是状态空间图中的一个完整路径

Construct both on demand – and construct as little as possible.

按需建造, 但尽可能少建造。

# Tree Search Algorithm

## Tree Search

```

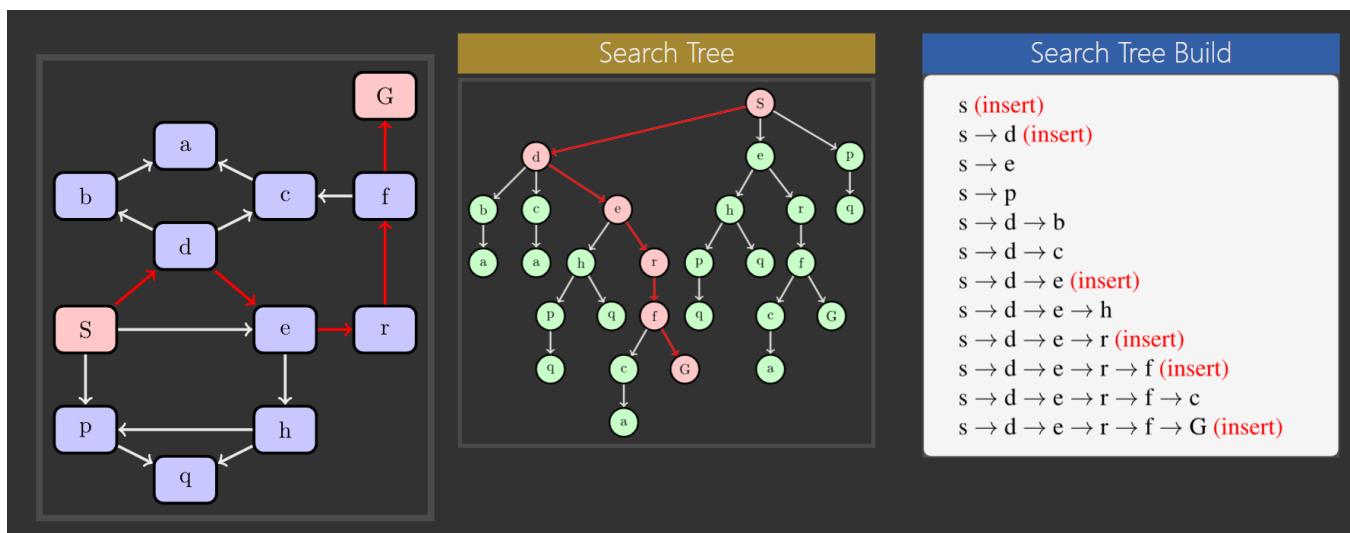
input : problem, strategy
output: solution or failure
initialize the search tree using initial state of problem;
while forever do
    if there are no candidates for expansion then return failure;
    choose a leaf node for expansion according to strategy;
    if node contains a goal state then
        | return the corresponding solution;
    else
        | expand the node and add the resulting nodes to the
          | search tree;
    end
end

```

1. Take initial state as root node. 以初始状态为根节点。

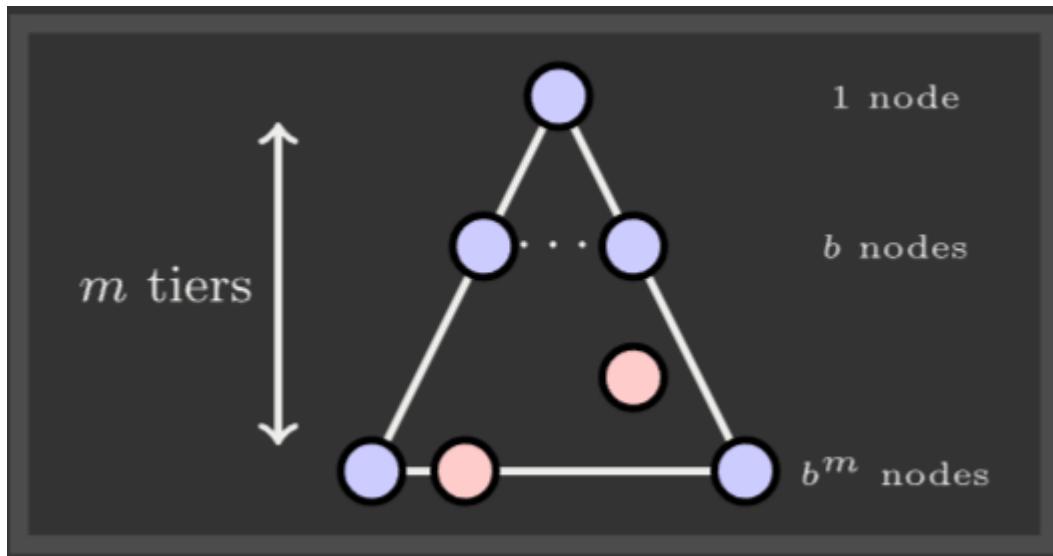
2. While:

1. check Fringe: if there are no candidates for expansion then return failure 检查边缘：如果没有候选扩展，则返回失败
2. choose expansion node: choose a leaf node for expansion according to strategy 选择扩展节点：根据策略选择一个叶节点进行扩展
3. goal test: if node contains a goal state then return the corresponding solution 目标测试：如果节点包含目标状态，则返回相应的解决方案
4. expand nodes: expand the node and add the resulting nodes to the search tree 扩展节点：扩展节点并将得到的节点添加到搜索树中



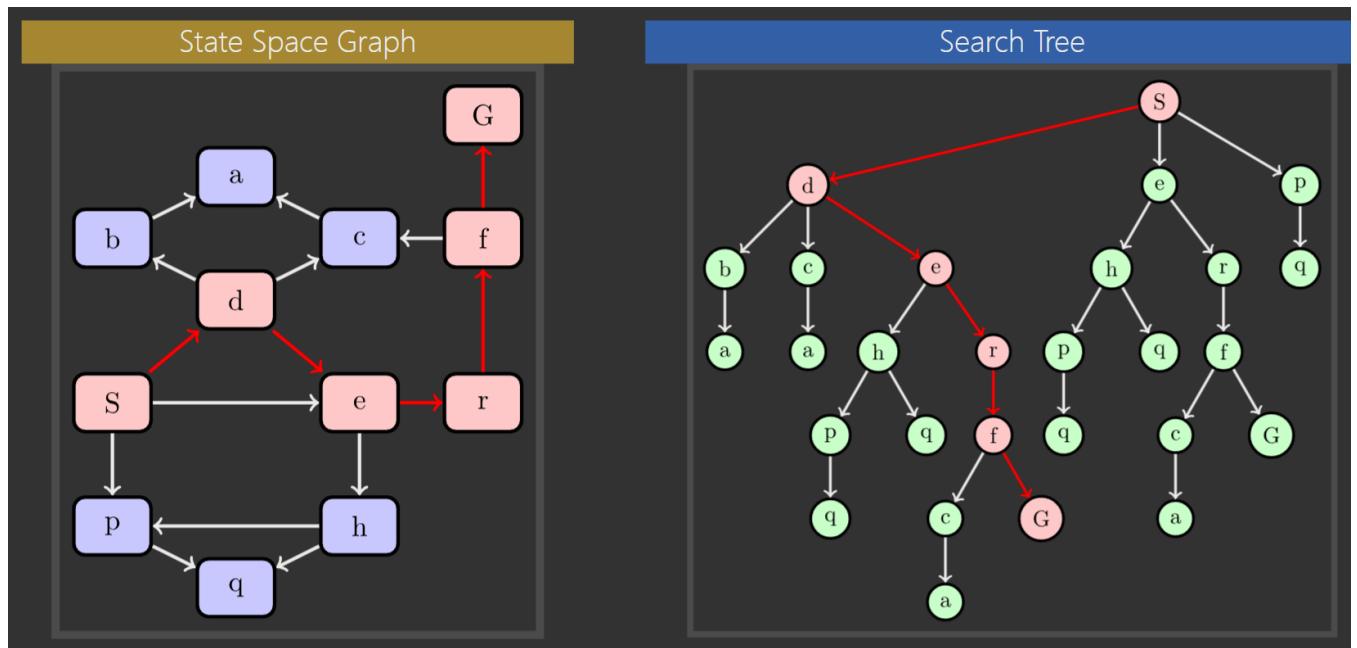
## Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists? 完整性：如果存在解决方案，能否保证找到？
- Optimal: Guaranteed to find the least cost path? 最优性：保证找到成本最低的路径？
- Time Complexity
- Space Complexity



- $b$ : the branching factor (Average number of child nodes generated per node)  $b$ : 分支因子（每个节点产生的子节点的平均数量）
- $m$ : the maximum depth (Number of steps from the root node to the deepest leaf node)  $m$ : 最大深度（从根节点到最深叶节点的步数）
- Number of nodes in a tree:  $1 + b + b^2 + \dots + b^m = O(b^m)$

## Depth First Search



DFS: Starting from the root node  $S$ , preferentially explore towards the deepest path (e.g.,  $S \rightarrow d \rightarrow e \rightarrow \dots$ ) until it encounters a dead end or a goal node  $G$ .

DFS: 从根节点 S 开始, 优先探索最深路径 (如  $S \rightarrow d \rightarrow e \rightarrow r \rightarrow f \rightarrow G$ ), 直到遇到死胡同或目标节点 G。

Fringe is a FILO or LIFO stack 边缘是 FILO 或 LIFO 栈

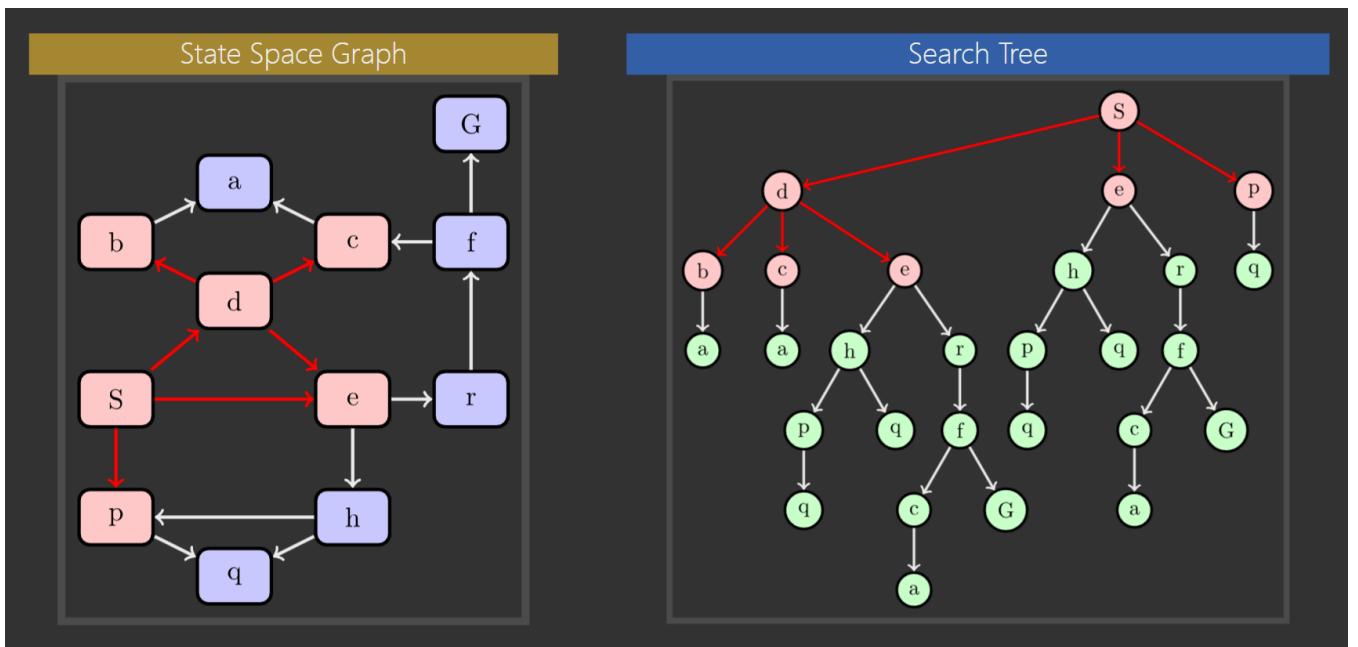
- Start from some left prefix of the tree. 从树的左前端开始。
- Could process the whole tree 可以处理整棵树
- If  $m$  is finite, takes time  $O(b^m)$

if  $m$  is finite, DFS is **complete**, but if there is cycles, it's not. 如果  $m$  是有限的, 则 DFS 是完整的, 但如果存在循环, 则不是。

DFS needs to store only the nodes on the current path and their unexplored siblings, so space complexity is  $O(bm)$  DFS 只需存储当前路径上的节点及其未探索的同级节点, 因此空间复杂度为  $O(bm)$

DFS is **not optimal**, it only finds the "leftmost" solution DFS 不是最优解, 它只能找到 "最左" 的解

## Breadth First Search



BFS: expand the shallowest node first (nodes with the smallest depth) BFS: 先扩展最浅的节点 (深度最小的节点)

fringe is a FIFO queue 边缘是一个先进先出队列

- Processes all nodes above shallowest solution 处理最浅解以上的所有节点
- Let depth of shallowest solution be  $s$ , time complexity is  $O(b^s)$

Space complexity is  $O(b^s)$ , usually higher than DFS

if  $s$  is finite, BFS is **complete**

Only when costs are all 1, BFS is optimal.

## Iterative Deepening

get the space advantage of DFS with the time / shallow-solution advantage of BFS. 既能获得 DFS 的空间优势，又能获得 BFS 的时间/浅解优势。

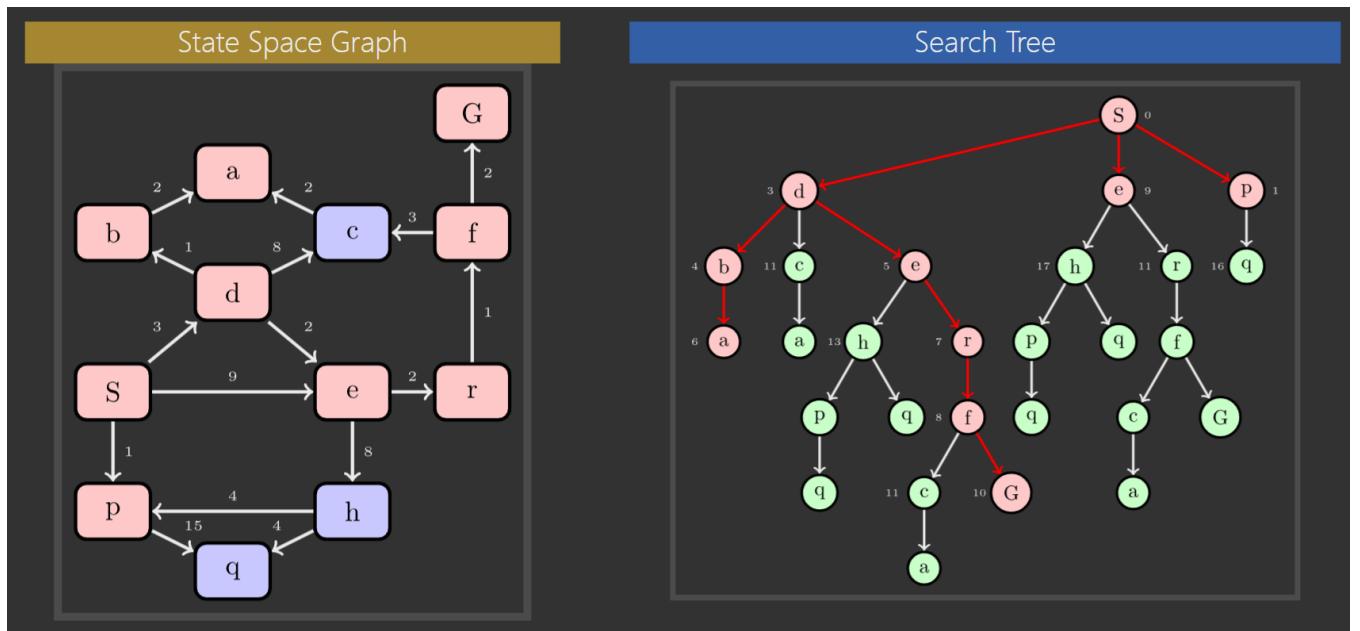
Run a DFS with depth limit 1. If no solution with depth limit 2.....

It's not wastefully redundant, because generally most work happens in the deepest level searched, so it is not so bad.

这并不是多余的浪费，因为一般来说，大部分工作都发生在最深层次的搜索中，所以并不那么糟糕。

Assume target at depth of d, time complexity is:  $O(b^1) + O(b^2) + \dots + O(b^{d-1}) + O(b^d) = O(b^d)$

## Uniform Cost Search (UCS)

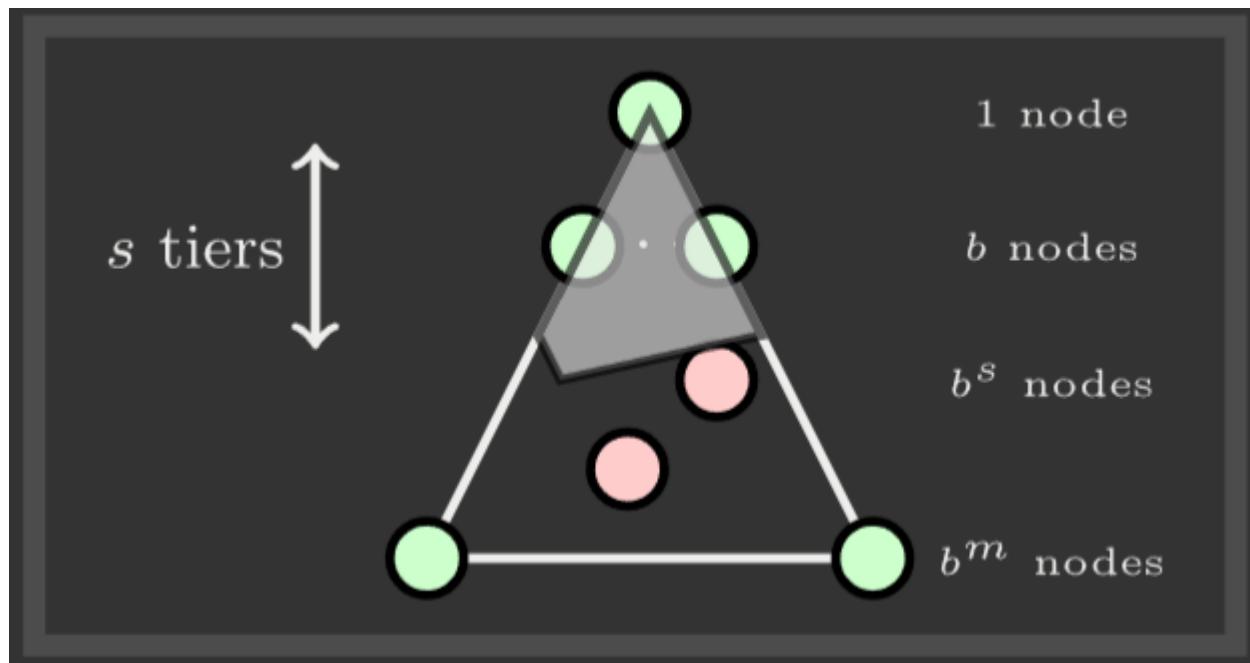


This algorithm tries to find the least-cost path.

UCS: expand a cheapest node first UCS: 先扩展最便宜的节点

Processing Path: S → p → d → b → e → a → r → f → e → G

fringe is a priority queue (priority: cumulative cost) 边缘是一个优先队列（优先级：累计成本）



- Processes all nodes with cost less than cheapest solution 处理所有成本低于最便宜解决方案的节点
- If the cheapest solution costs  $C^*$ , and arcs cost at least  $\epsilon$ , the "effective depth" is roughly  $(\frac{C^*}{\epsilon})$
- time complexity and space complexity are both  $O(b \frac{C^*}{\epsilon})$

If best solution has a finite cost and minimum arc cost is positive, UCS is **complete**

UCS explores increasing cost contours UCS 探索不断增长的成本轮廓

UCS is **optimal** and **complete**, but it explores options in every "direction" with no information about goal location

UCS 是**最优和完整**, 但它会在没有目标位置信息的情况下, 探索每个 "方向" 的选项

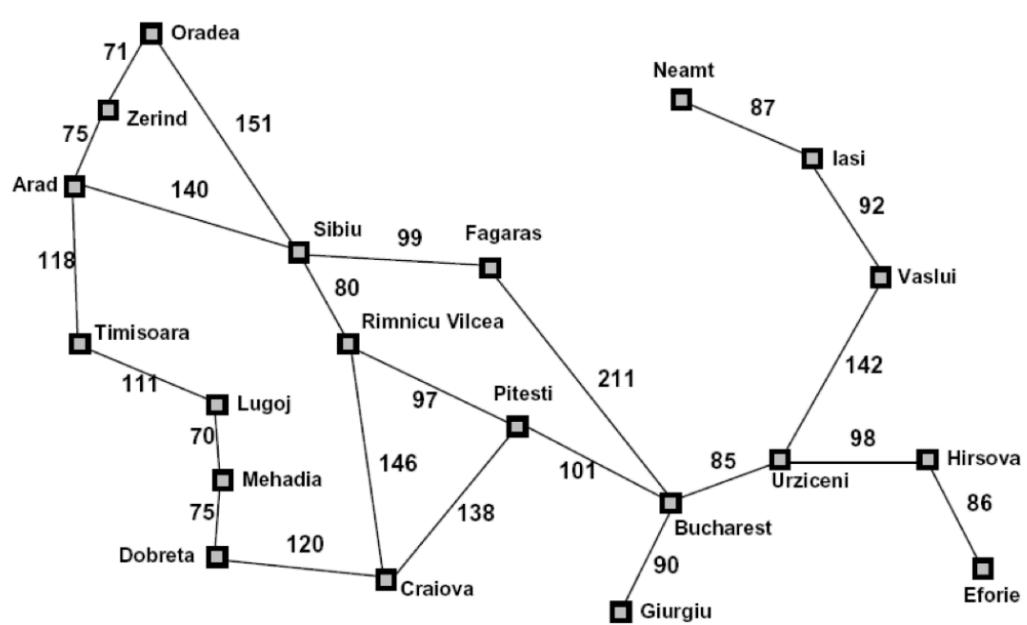
# 3.Informed Search

## Search Heuristics

A heuristic is: 启发式是：

- A function that **estimates** (not precise) how close a state is to a goal 估算（非精确值）状态与目标接近程度的函数
- Designed for a particular search problem 为特定搜索问题而设计

Like Manhattan distance, Euclidean distance

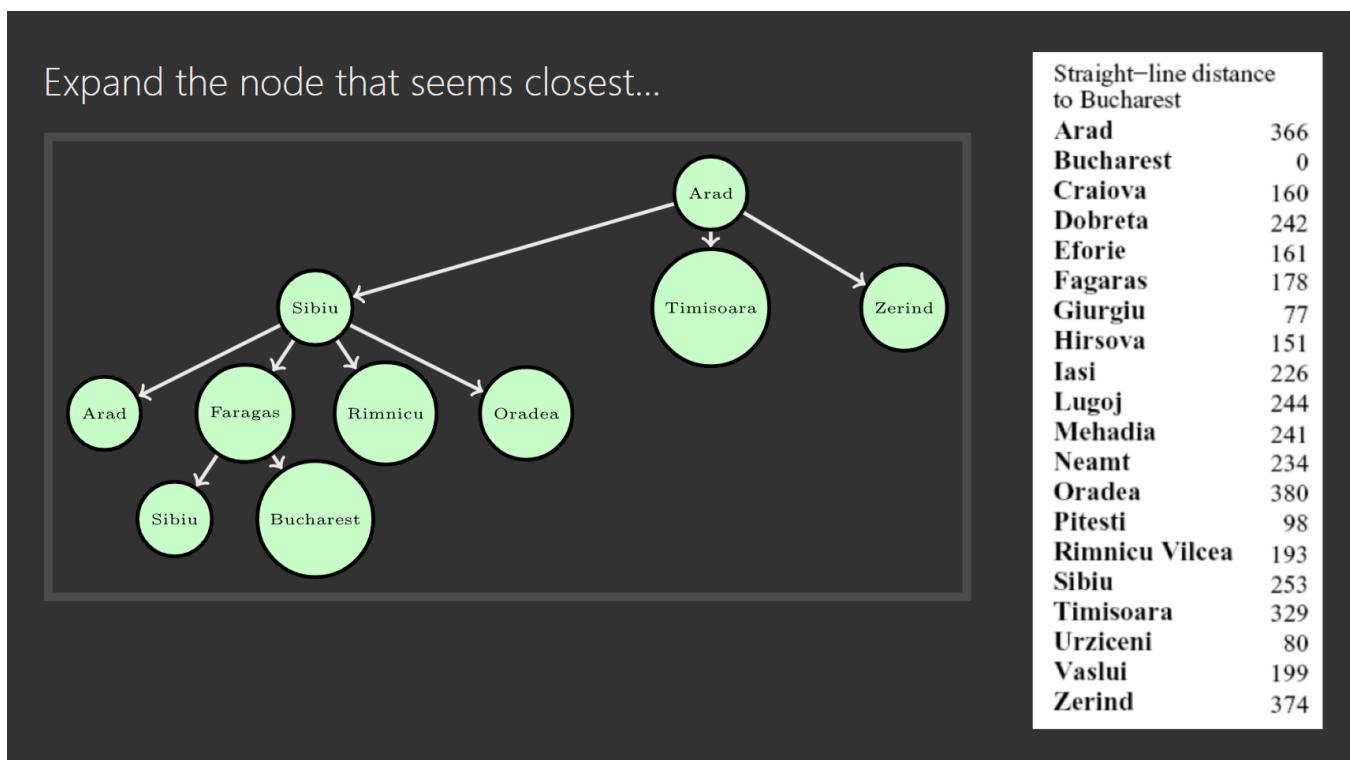


Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Straight-line distance could be seen as a heuristic 直线距离可视为一种启发式方法

## Greedy Search



Expand the node that seems closest 展开看起来最接近的节点

Heuristic: distance **estimate** to nearest goal for each state 启发函数：估计每个状态到最近目标的距离

But it only guarantees local optimal instead of global optimal 但它只能保证局部最优，而不是全局最优

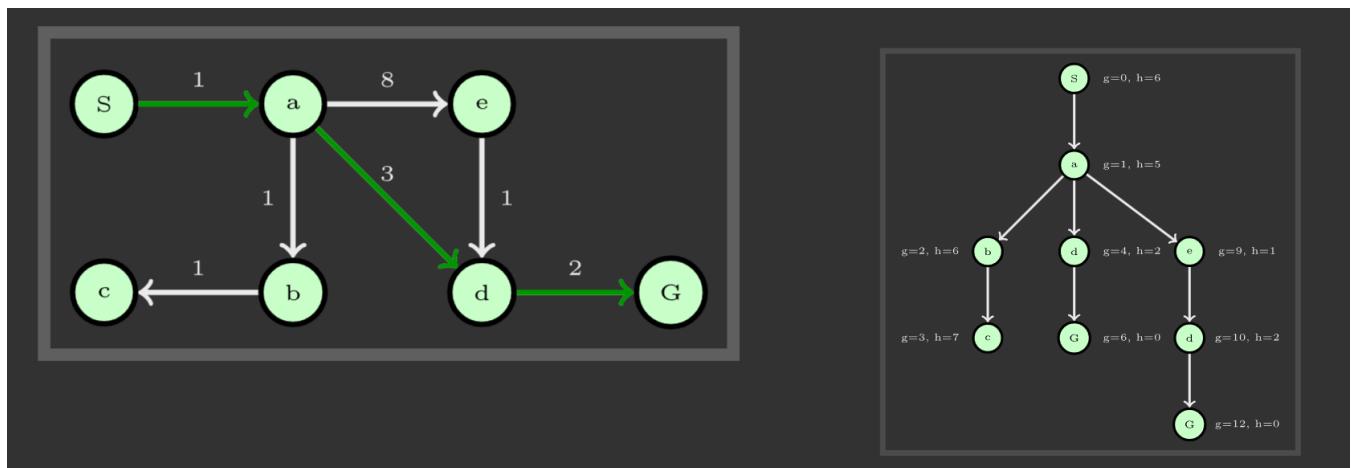
A common case: a best-first node takes you straight to the (wrong) goal 常见情况：最佳先行节点会让你直接到达（错误的）目标

Worst-case: like a badly-guided DFS 最坏的情况：就像指导不力的 DFS

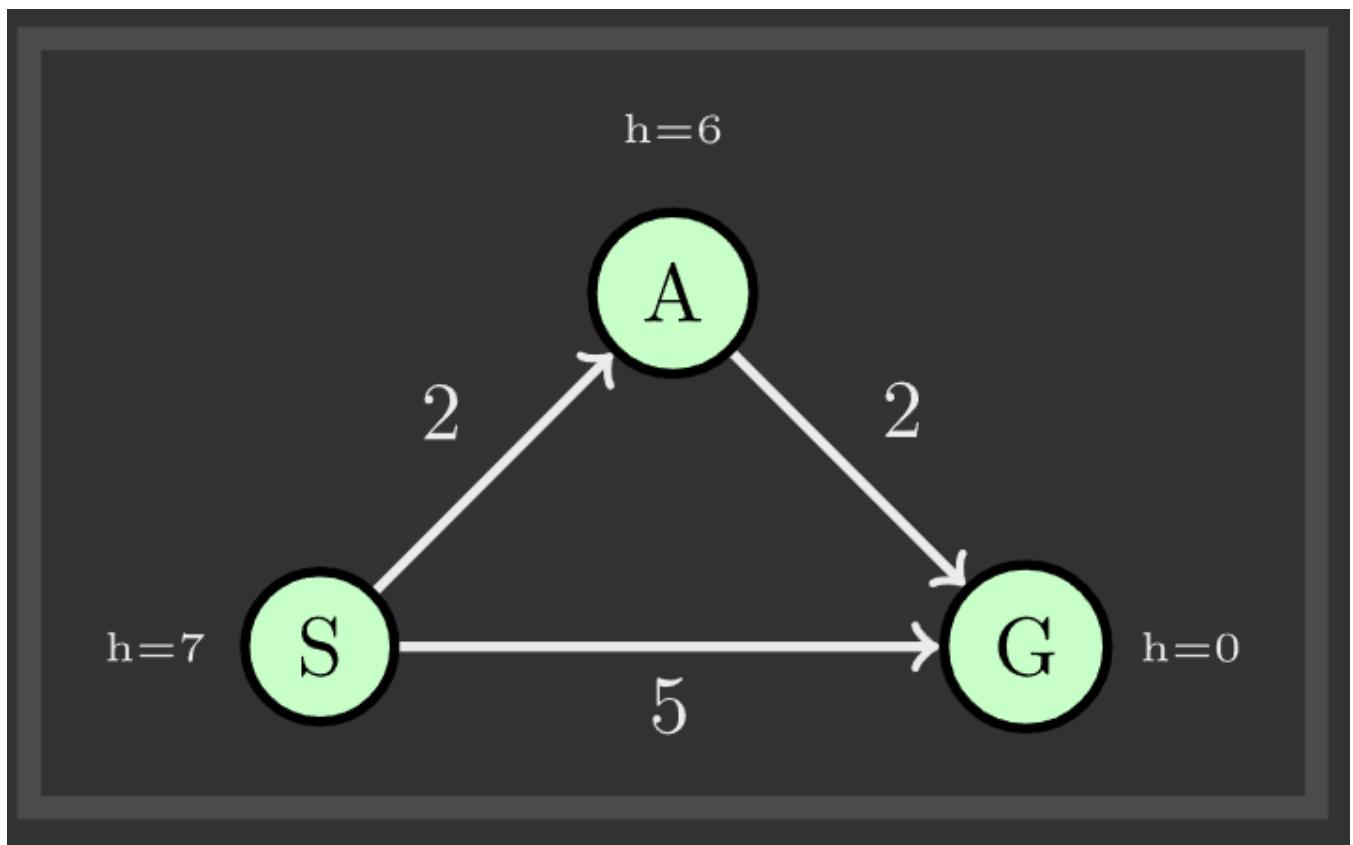
## A\* Search

It Combines UCS and Greedy.

- Uniform-cost orders by path cost, or backward cost  $g(n)$
- Greedy orders by goal proximity, or forward cost  $h(n)$
- A\* Search orders by the sum:  $f(n) = g(n) + h(n)$



Is A\* always optimal?



$f(A)=2+6=8$ ,  $f(G)=0+5=5$ , so we go  $S \rightarrow G$ , but the actually path should be  $S \rightarrow A \rightarrow G$

We wrongly estimate  $h$ . Heuristic design must be conservative

A\* is optimal when estimated goal cost  $\leq$  Actual goal cost (Admissible heuristics) 当估计目标成本 $\leq$ 实际目标成本时, A\*为最优 (可接受的启发式方法)

Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs 可接受的 (乐观的) 启发式方法会减缓糟糕计划的速度, 但永远不会超过真正的成本

# 4.Constraint Satisfaction

---

## What is Search For

---

Given assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space

给出对世界的假设：单一代理、确定性行动、完全观察状态、离散状态空间

- Planning: sequences of actions (care about path) 规划：行动序列（关心路径）
  - The path to the goal is the important thing 通往目标的道路是重要的
  - Paths have various costs, depths 路径有不同的成本、深度
  - Heuristics give problem-specific guidance 启发式方法针对具体问题提供指导
- Identification: assignments to variables (care about state itself) 识别：变量赋值（关心状态本身）
  - The goal itself is important, not the path 目标本身很重要，路径并不重要
  - All paths at the same depth (for some formulations) 所有路径深度相同（对于某些配方而言）
  - CSPs are a specialized class of identification problems 约束满足问题是一类专门的识别问题

## Constraint Satisfaction Problems

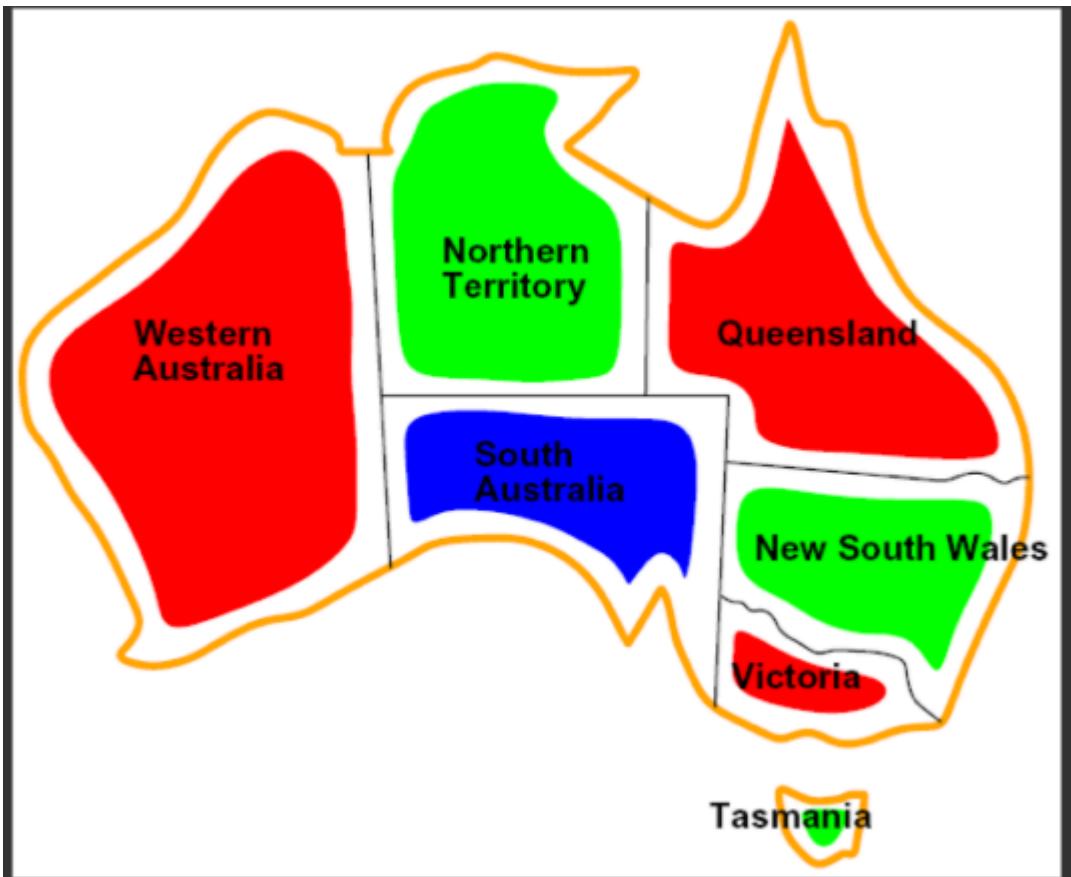
---

Standard search problems: 标准搜索问题：

- State is a "black box": arbitrary data structure 状态是一个 "黑盒子"：任意数据结构
- Goal test can be any function over states 目标测试可以是状态上的任何函数
- Successor function can also be anything 后续函数也可以是任何函数

Constraint satisfaction problems (CSPs): 约束满足问题 (CSP)：

- A special subset of search problems 搜索问题的一个特殊子集
- State is defined by variables  $X_i$  with values from a domain  $D$  (sometimes  $D$  depends on  $i$ ) 状态由变量  $X_i$  定义，其值来自一个域  $D$ （有时  $D$  取决于  $i$ ）。
- Goal test is a set of constraints specifying allowable combinations of values for subsets of variables 目标测试是一组约束条件，规定了变量子集的允许值组合 Like ( $X_1 \neq X_2$ )
- Allows useful general-purpose algorithms with more power than standard search algorithms 允许使用比标准搜索算法更强大的实用通用算法



- Variables: WA, NT, Q, NSW, V, SA, T
- Domains:  $D = \{red, green, blue\}$
- Constraints: adjacent regions must have different colors
  - Implicit: WA ≠ NT
  - Explicit: (WA, NT) ∈ {(red, green), (red, blue), ...}
- Solutions: Assignments that satisfy all constraints, e.g.:
  $\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

Variety of CSPs: 各种 CSP:

- Discrete Variables: 离散变量:
  - Finite domains: Size d means  $O(d^n)$  complete assignments 有限域：大小 d 意味着  $O(d^n)$  完成任务
  - Infinite domains (integers, strings, etc.): E.g., job scheduling, variables are start/end times for each job 无限域（整数、字符串等）：例如，作业调度，变量是每个作业的开始/结束时间
- Continuous variables 连续变量:
  - E.g., start/end times for Hubble Telescope observations 例如，哈勃望远镜观测的开始/结束时间
- Hard Constraints: 硬约束:
  - Unary constraints involve a single variable (equivalent to reducing domains), e.g.: SA ≠ green 一元约束涉及单一变量（相当于缩小域），例如 SA ≠ 绿色
  - Binary constraints involve pairs of variables, e.g.: SA ≠ WA 二元约束涉及成对的变量，例如 SA ≠ WA
  - Higher-order constraints involve 3 or more variables: e.g., cryptarithmetic column constraints 高阶约束涉及 3 个或更多变量：如加密列约束

- Preferences (soft constraints): 偏好 (软约束):
  - E.g., red is better than green 例如, 红色比绿色好
  - Often representable by a cost for each variable assignment 通常可以用每个变量分配的成本来表示
  - Leads to constrained optimization problems 导致受约束的优化问题

## Solving CSPs

---

Modeling CSPs into normal search problems 将 CSP 建模为普通搜索问题

States defined by the values assigned so far (partial assignments) 状态由目前分配的值来定义

- Initial state: the empty assignment, {} 初始状态: 空赋值 {}
- Successor function: assign a value to an unassigned variable 继承函数: 为未赋值变量赋值
- Goal test: the current assignment is complete and satisfies all constraints 目标测试: 当前任务已完成并满足所有限制条件

# 5. Backtracking Search

---

main ideas:

1. One variable at a time: process variables based on fixed order. i.e., [WA = red then NT = green] same as [NT = green then WA = red] 一次一个变量：根据固定顺序处理变量。即 [WA = 红色，则 NT = 绿色] 与 [NT = 绿色，则 WA = 红色] 相同
2. Check constraints as you go: after assigning variables, check the constraints. if conflicts happen, back to previous state. 边操作边检查约束条件：分配变量后，检查约束条件。如果发生冲突，则返回之前的状态

An improved DFS algorithm. 改进的 DFS 算法

3 improvement directions: Ordering, Filtering, Structure 3 个改进方向：排序、筛选、结构

## Ordering

---

Which variable should be assigned next? 接下来应该分配哪个变量？

In what order should its values be tried? 应按什么顺序尝试其值？

**Minimum Remaining Values (MRV):** Choose the variable with the fewest legal values left in its domain

**最低剩余价值 (MRV):** 选择其域中合法剩余值最少的变量

**Least Constraining Values (LCV):** Given a choice of variable, choose the least constraining value

**最小限制值 (LCV):** 给定变量选择，选择最小约束的值

**MRV+LCV:** Choose variable by MRV, then assign its value by LCV  
**MRV+LCV:** 通过 MRV 选择变量，然后通过 LCV 赋值

## Filtering

---

Can we detect inevitable failure early?

我们能否及早发现不可避免的故障？

**Filtering:** Keep track of domains for unassigned variables and cross off bad options. 过滤：跟踪未分配变量的域，划掉不良选项。

**过滤：**跟踪未分配变量的域，划掉不良选项

**Forward checking:** Immediately after each assignment, the domains of neighboring variables are checked to remove values that conflict with the current assignment.

**前向检查：**每次赋值后，立即检查相邻变量的域，删除与当前赋值冲突的值。

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures

前向检查将信息从已分配变量传播到未分配变量，但并不能及早发现所有故障

**Consistency of A Single Arc:** An arc  $X \rightarrow Y$  is consistent iff for every  $x$  in the tail there is some  $y$  in the head which could be assigned without violating a constraint.

**单一弧线的一致性：**弧  $X \rightarrow Y$  是一致的，如果尾部的每个  $x$  都可以在不违反约束条件的情况下分配给头部的某个  $y$ 。

**Arc Consistency:** process all arcs, check consistency of all arcs. If a value of a variable has no legal counterpart in a neighboring variable, the value is deleted.

**弧一致性：**处理所有弧，检查所有弧的一致性。如果一个变量的值在相邻变量中没有合法的对应变量，则删除该值。If  $X$  loses a value, neighbors of  $X$  need to be rechecked. 如果  $X$  丢失了一个值，则需要重新检查  $X$  的邻域。

## Problem Structure

---

Can we exploit the problem structure?

我们能否利用问题结构？

The exponential complexity problem is transformed into a linear complexity problem by identifying independent subproblems (connected components) through constraint graphs.

通过约束图确定独立的子问题（连接部分），将指数复杂度问题转化为线性复杂度问题。

# 6.Adversarial Game and Search

Zero-Sum Games: 零和游戏

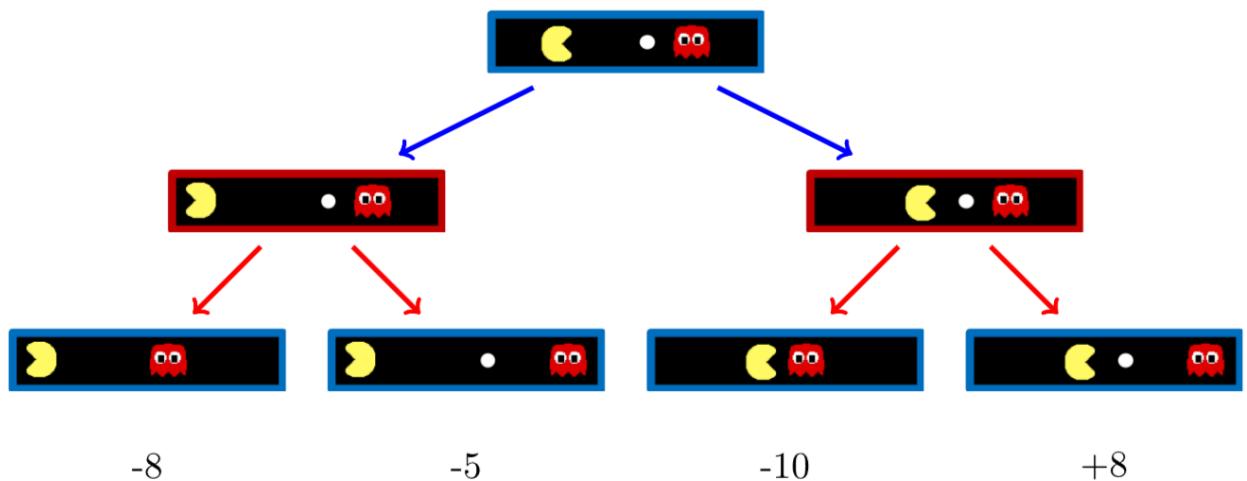
- Agents have opposite utilities 代理具有相反的效用
- A single value that one maximizes and the other minimizes 一个值最大化，另一个值最小化
- Adversarial, pure competition 对抗性、纯竞争

General Games

- Agents have independent utilities 代理拥有独立的效用
- Cooperation, indifference, competition, and more are all possible 合作、冷漠、竞争等等都有可能发生

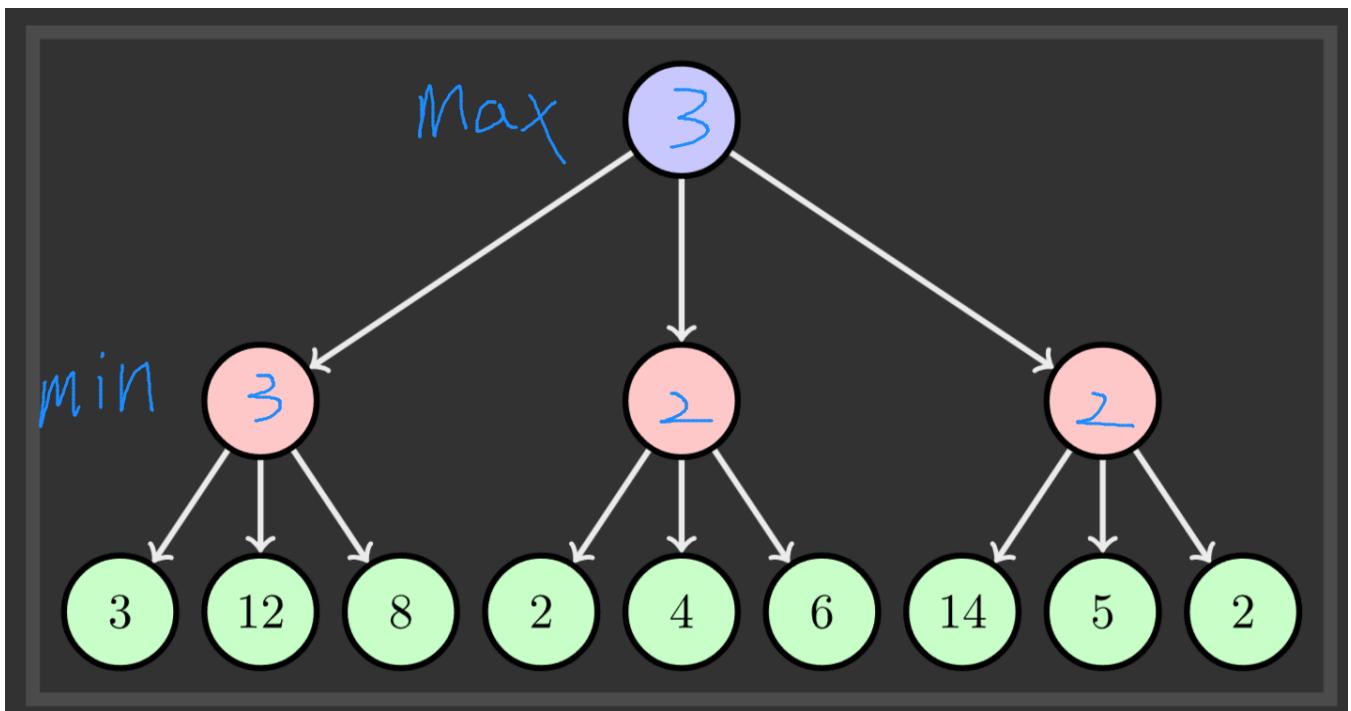
## Adversarial Search

Minimax Values:



States Under Agent's Control:  $V(s) = \max_{s' \in \text{successors}(s)} V(s')$

States Under Opponent's Control:  $V(s) = \min_{s' \in \text{successors}(s)} V(s')$



- A state-space search tree 状态空间搜索树
- Players alternate turns 玩家交替轮流
- Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary 计算每个节点的最小值：面对理性（最优）对手时可实现的最佳效用

Just like (exhaustive) DFS, Time:  $O(b^m)$ , Space:  $O(bm)$

## A-B Pruning

1. Set layer rule (MAX or Min), except leaf nodes. 设置层规则 (MAX 或 Min)，叶节点除外。
2. Set initial value of  $\alpha = -\infty$  and  $\beta = +\infty$  for all nodes 对所有节点设置初始值  $\alpha = -\infty$  和  $\beta = +\infty$
3. DFS process (left and down) DFS流程 (向左和向下)
4. for Min layer, it only changes  $\beta$  as  $\text{MIN}(\text{itself}, \text{downer } \alpha, \text{downer } \beta)$  对于最小层，它只改变  $\beta$ ，即 MIN (自身, 下层  $\alpha$ , 下层  $\beta$ )  
for Max layer, it only changes  $\alpha$  as  $\text{MAX}(\text{itself}, \text{downer } \alpha, \text{downer } \beta)$  对于最大层，它只改变  $\alpha$ ，即 MAX (自身, 下层  $\alpha$ , 下层  $\beta$ )
5. After value update, backtrack to its parent node with value transfer. 左子节点值更新后，回溯到其父节点并传递值。
6. Then parent node transfer its  $\alpha$  and  $\beta$  to its right child node. 然后，父节点将其  $\alpha$  和  $\beta$  转移到右侧子节点。
7. When right child node is processed over, it also need to backtrack to parent node and update parent value. 当处理完右子节点后，还需要回溯到父节点并更新父节点的值。
8. Keep the loop of step 3-7 保持步骤 3-7 的循环
9. Whenever  $\alpha \geq \beta$ , prune 只要  $\alpha \geq \beta$ ，就剪枝

# $\alpha-\beta$ 剪枝算法.

Max

Min

Max

Min

$\alpha = -\infty$  7  
 $\beta = +\infty$

{ Max 层只改变  $\alpha$ , 取  $\max(\text{自己}, \text{下层} \alpha, \text{下层} \beta)$ 。  
 Min 层只改变  $\beta$ , 取  $\min(\text{自己}, \text{下层} \alpha, \text{下层} \beta)$ 。  
 $\alpha$  和  $\beta$  的值的传递: 先左子树, 后返回父节点, 再右子树。  
 当  $\alpha \geq \beta$  时进行剪枝。

$\alpha$  初值  $-\infty$ 。  
 $\beta$  初值  $+\infty$ 。

$\alpha = -\infty$  5 7  
 $\beta = +\infty$

$\alpha = +\infty$  11  
 $\beta = +\infty$

$\alpha = 7$   
 $\beta = +\infty$  11

$\alpha = -\infty$   
 $\beta = +\infty$

$\alpha = -\infty$   
 $\beta = 7$

$\alpha = +\infty$   
 $\beta = +\infty$

$\alpha = +\infty$   
 $\beta = 11$

$\alpha = -\infty$   
 $\beta = +\infty$

$\alpha = -\infty$   
 $\beta = 7$

$\alpha = +\infty$   
 $\beta = +\infty$

$\alpha = +\infty$   
 $\beta = 11$

$\alpha = -\infty$   
 $\beta = +\infty$

$\alpha = -\infty$   
 $\beta = 7$

$\alpha = +\infty$   
 $\beta = +\infty$

$\alpha = +\infty$   
 $\beta = 11$

$\alpha = -\infty$   
 $\beta = +\infty$

$\alpha = -\infty$   
 $\beta = 7$

$\alpha = +\infty$   
 $\beta = +\infty$

$\alpha = +\infty$   
 $\beta = 11$

After using  $A-B$  Pruning, in perfect ordering, time complexity drops to  $O(b^{\frac{m}{2}})$

# 7. Probabilities and Utilities

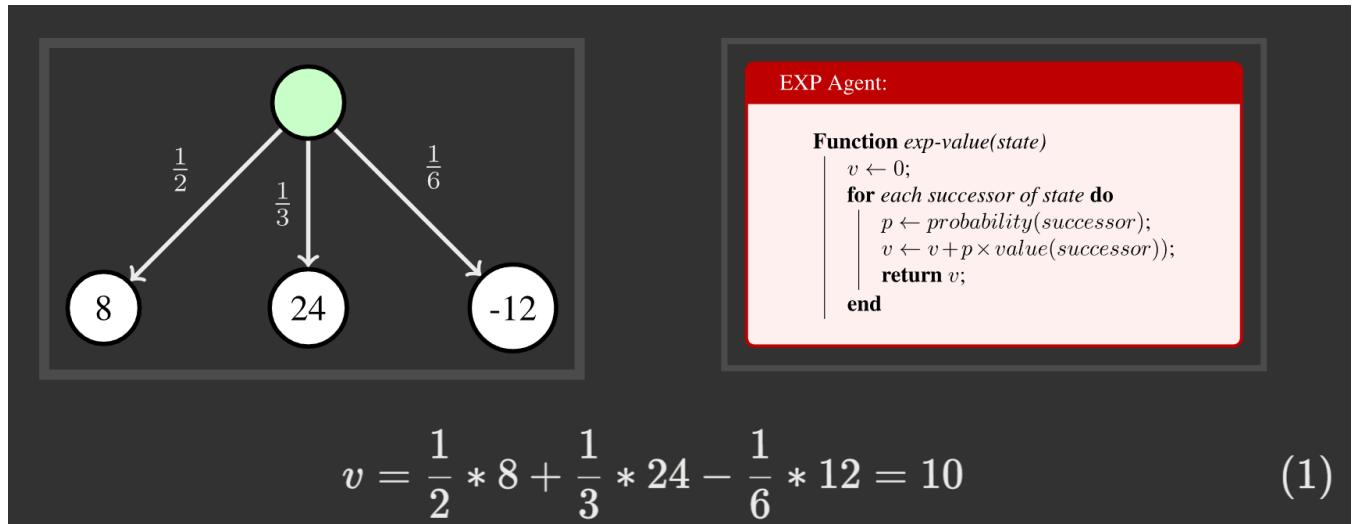
Values should reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes, because opponents are unpredictable.

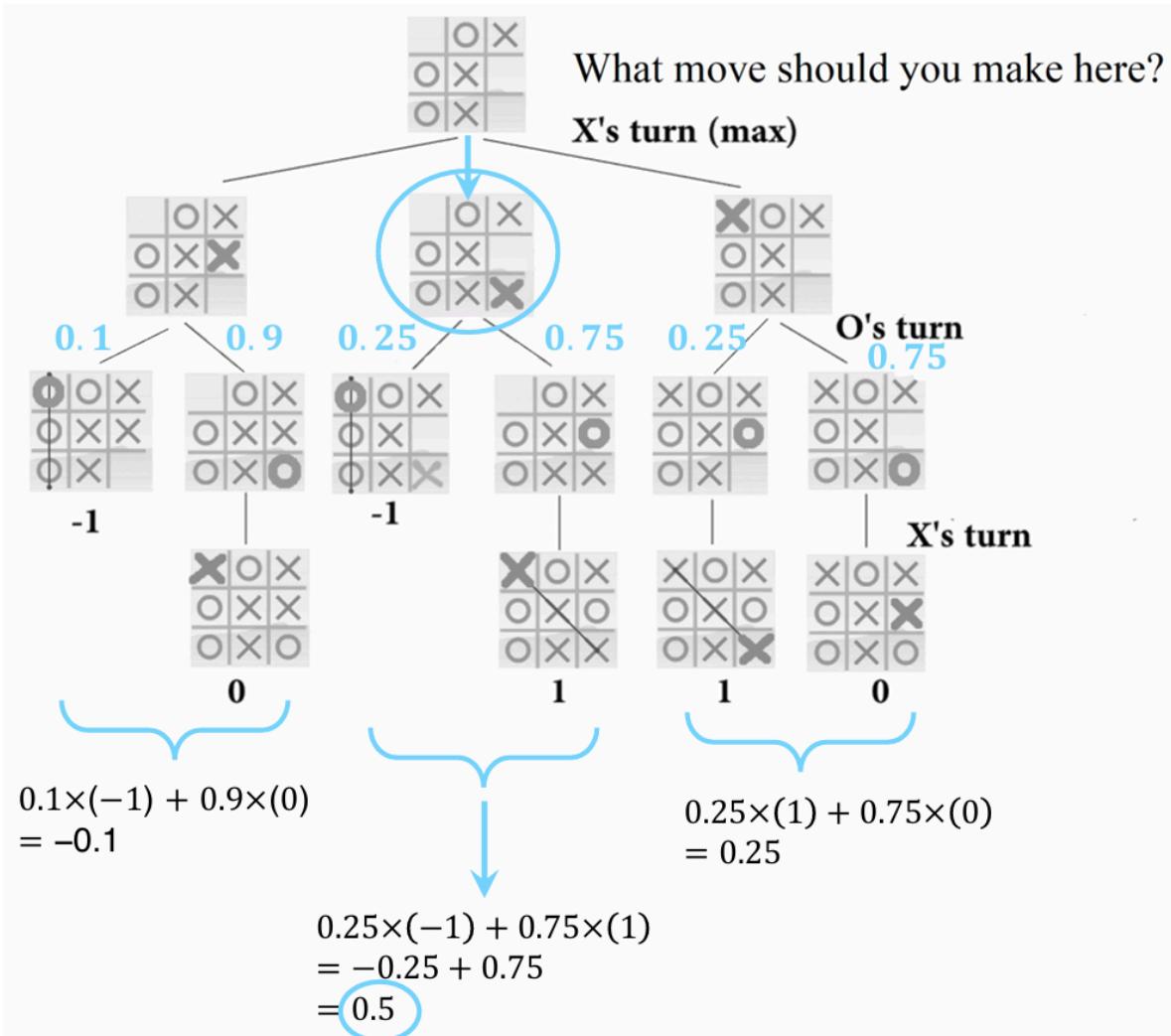
数值应反映平均情况（期望最大值）结果，而不是最坏情况（最小值）结果，因为对手是不可预测的。

## Expectimax Search

compute the average score under optimal play 计算最佳打法下的平均得分

- Max nodes as in minimax search 最大节点与最小搜索相同
- Chance nodes are like min nodes but the outcome is uncertain 机会节点与最小节点类似，但结果不确定
- Calculate their **expected utilities** 计算他们的预期效用
- i.e., take weighted average (expectation) of children 即，取子节点的加权平均数（期望值）





## 8. Linear Programming

Considering trying healthy by finding the optimal amount of food to purchase.

考虑通过寻找最佳购买量来尝试健康食品。

We can choose the amount of stir-fry (ounce) and boba (fluid ounces).

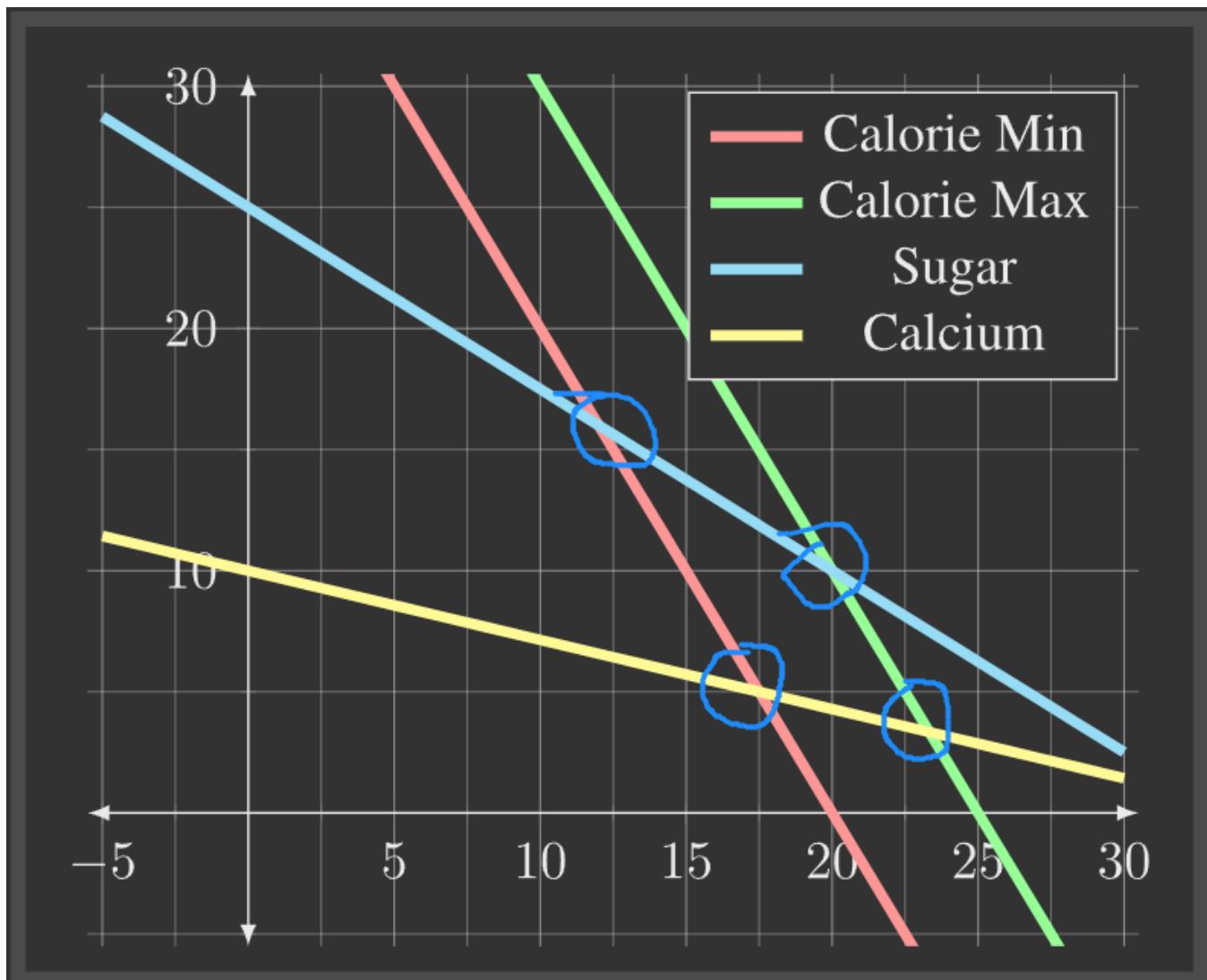
我们可以选择炒菜（盎司）和荞麦面（液体盎司）的量。

Food	Cost	Calories	Sugar	Calcium
<b>stir-fry (per oz)</b>	1	100	3	20
<b>Boba (per fl oz)</b>	0.5	50	4	70

- $2000 \leq \text{Calories} \leq 2500$
- $\text{Sugar} \leq 100\text{g}$
- $\text{Calcium} \geq 700\text{mg}$

We need to min  $(1x_1 + 0.5x_2)$  satisfying:

- $100x_1 + 50x_2 \geq 2000$
- $100x_1 + 50x_2 \leq 2500$
- $3x_1 + 4x_2 \leq 100$
- $20x_1 + 70x_2 \geq 700$



Solutions are at feasible intersections of constraint boundaries!! 解决方案位于约束边界的可行交叉点!

# 9. Integer Programming

---

How to solve a LP?

- Check objective at all feasible intersections. 检查所有可行路口的目标。
- Simplex 单纯形
- Interior Point 内点

IP is similar to LP, with one more constraint:  $x \in Z^N$

For the result, result of LP is usually better than that of IP, because LP has a much looser constraint.

就结果而言，LP 的结果通常优于 IP 的结果，因为 LP 的约束要宽松得多。

# 10.Optimization

---

Optimization Problem: Determine value of optimization variable within feasible region/set to optimize  
optimization objective 优化问题：在可行区域/集合内确定优化变量的值，以优化优化目标

- Optimization variable:  $x \in R^n$  variable waiting for determined. 等待确定的变量
- Feasible region/set:  $F \subseteq R^n$  set of all solutions that satisfy the constraints 所有满足约束条件的解的集合
- Optimization objective:  $f : F \rightarrow R$  f is a function to estimate whether the solution is good or not. f 是一个函数，用于估算解决方案的优劣

Optimal solution 最佳解:  $x^* = \operatorname{argmin} f(x), x \subseteq F$

Optimal objective value 最佳目标值:  $f^* = \min f(x) = f(x^*)$

How to solve?

No general way to solve

# 11. Convex Optimization - I

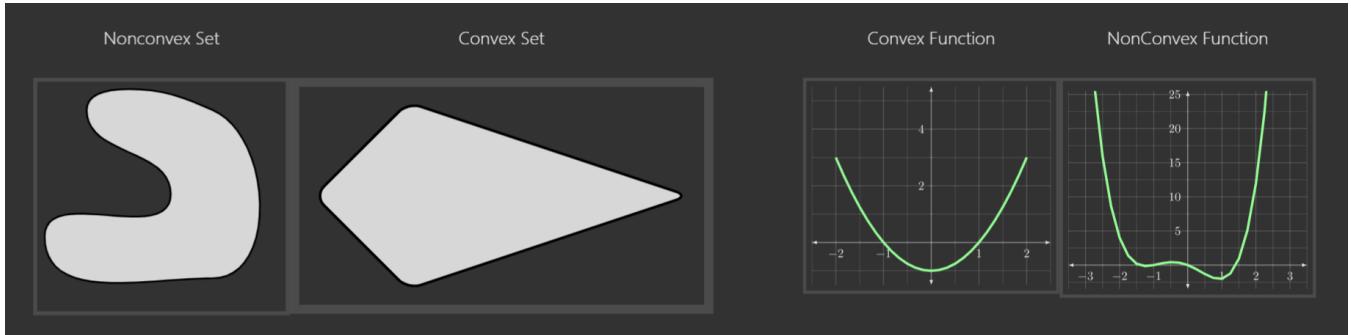
Convex Optimization Problem:

$$\min f(x)$$

$$s.t. x \in F$$

A special class of optimization problem, optimization objective  $f$  is a convex function and feasible region  $F$  is a convex set.

一类特殊的优化问题，优化目标  $f$  是一个凸函数，可行区域  $F$  是一个凸集。



## Convex Combination

**Convex Combination:** Given  $x, y \in R^n$ , a convex combination of them is any point of the form  $z = \theta x + (1 - \theta)y$  where  $\theta \in [0, 1]$ . When  $\theta \in (0, 1)$ ,  $z$  is called a strict convex combination of  $x, y$ .

**凸组合：**给定  $x, y \in R^n$ , 它们的凸组合是形式为  $z = \theta x + (1 - \theta)y$  的任意一点，其中  $\theta \in [0, 1]$ 。当  $\theta \in (0, 1)$  时， $z$  称为  $x, y$  的严格凸组合。

## Convex Set

### Convex Set: 凸集合

- Conceptually: Any convex combination of two points in the set is also in the set 概念上：集合中任意两点的凸组合也在集合中
- Mathematically: A set  $F$  is convex if  $\forall x, y \in F, \forall \theta \in [0, 1], z = \theta x + (1 - \theta)y \in F$  数学上：如果  $\forall x, y \in F, \forall \theta \in [0, 1], z = \theta x + (1 - \theta)y \in F$ , 则集合  $F$  是凸的。

## Convex Function

**Convex Function:** Value in the middle point is lower than average value **凸函数：**中间点的值低于平均值

Mathematically: Let  $F$  be a convex set. A function  $f : F \rightarrow R$  is convex in  $F$  if  $\forall x, y \in F, \forall \theta \in [0, 1]$ ,

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y)$$

If  $F = R^n$ , we simply say  $f$  is convex.

How to determine if a functions is convex?

1. Sum of convex functions is convex 凸函数之和是凸函数

$$\text{if } f(x) = \sum_i w_i f_i(x), w_i \geq 0, f_i(x) \text{ convex, then } f(x) \text{ is convex.}$$

2. Convexity is preserved under a linear transformation 在线性变换下保持凸性

$$f(x) = g(Ax + b), \text{ if } g(x) \text{ is convex, then } f(x) \text{ is convex.}$$

3. If  $f$  is a twice differentiable function of one variable,  $f$  is convex on  $[a, b] \in R$  iff(if and only if) its second derivative  $f'' \geq 0$  in  $[a, b]$ .

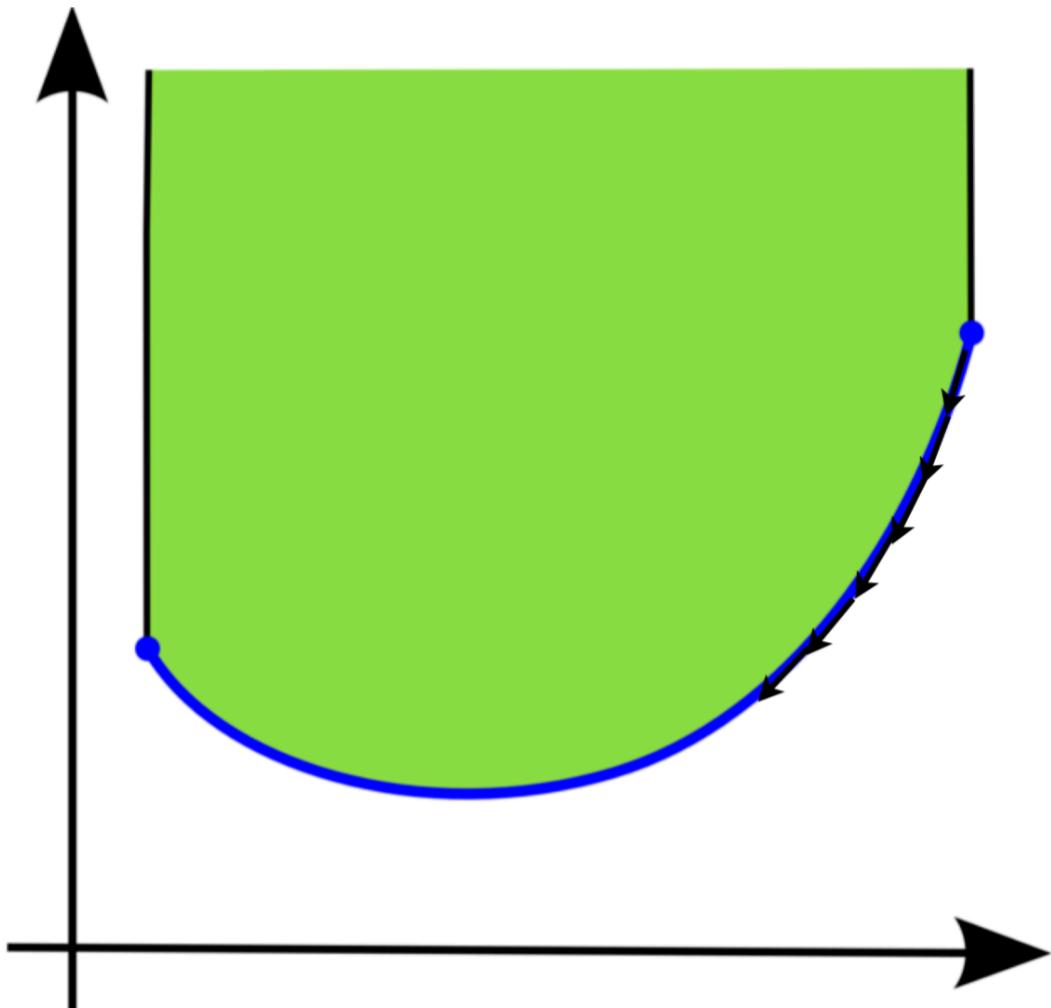
如果  $f$  是单变量的二次可微分函数，那么  $f$  在  $[a, b] \in R$  上是凸的，如果（当且仅当）它在  $[a, b]$  的二阶导数  $f'' \geq 0$ 。

For convex function, Local Optima=Global Optima 对于凸函数，局部最优值=全局最优值

## 12. Convex Optimization - II

How to solve?

Gradient descent: iteratively update the value of  $x$  梯度下降：迭代更新  $x$  值



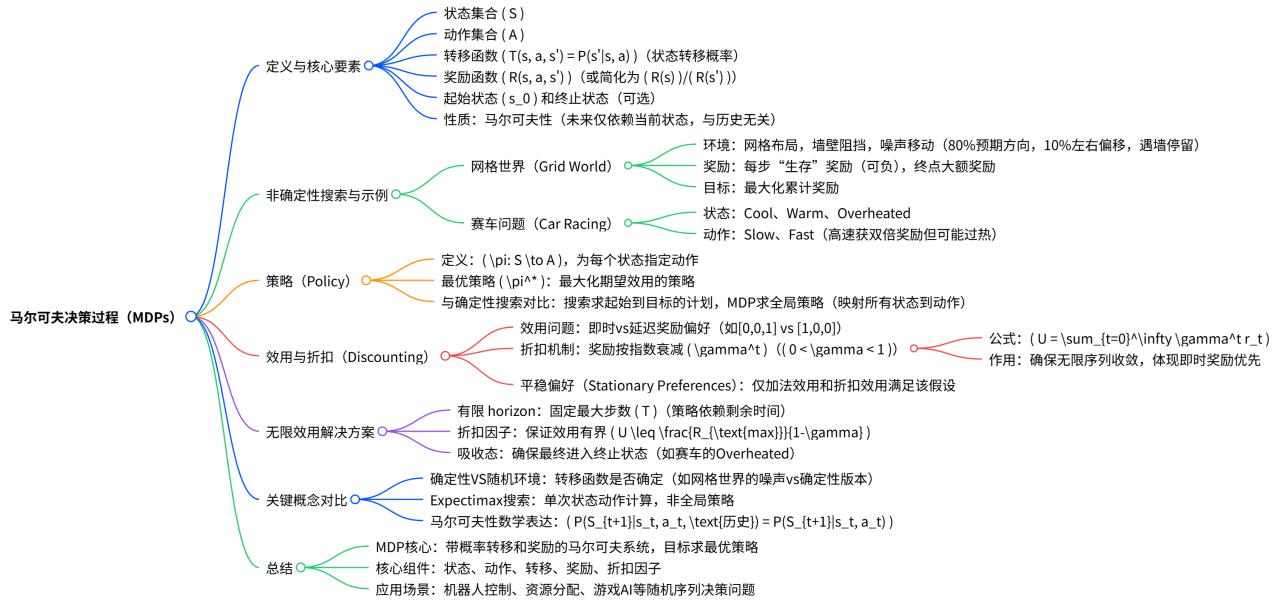
1. Choose an initial point  $x_0$

2. Iteratively update:

$$x^{k+1} = x^k - \alpha \nabla f(x^k)$$

3. Stop when convergence 收敛时停止

# 13. Markov Decision Process - I



豆包 你的 AI 助手, 助力每日工作学习

## Non-Deterministic Search

考虑如下的场景:



### 环境设定:

- 智能体 (agent) 在网格中活动, 墙壁 (walls) 会阻挡其路径。

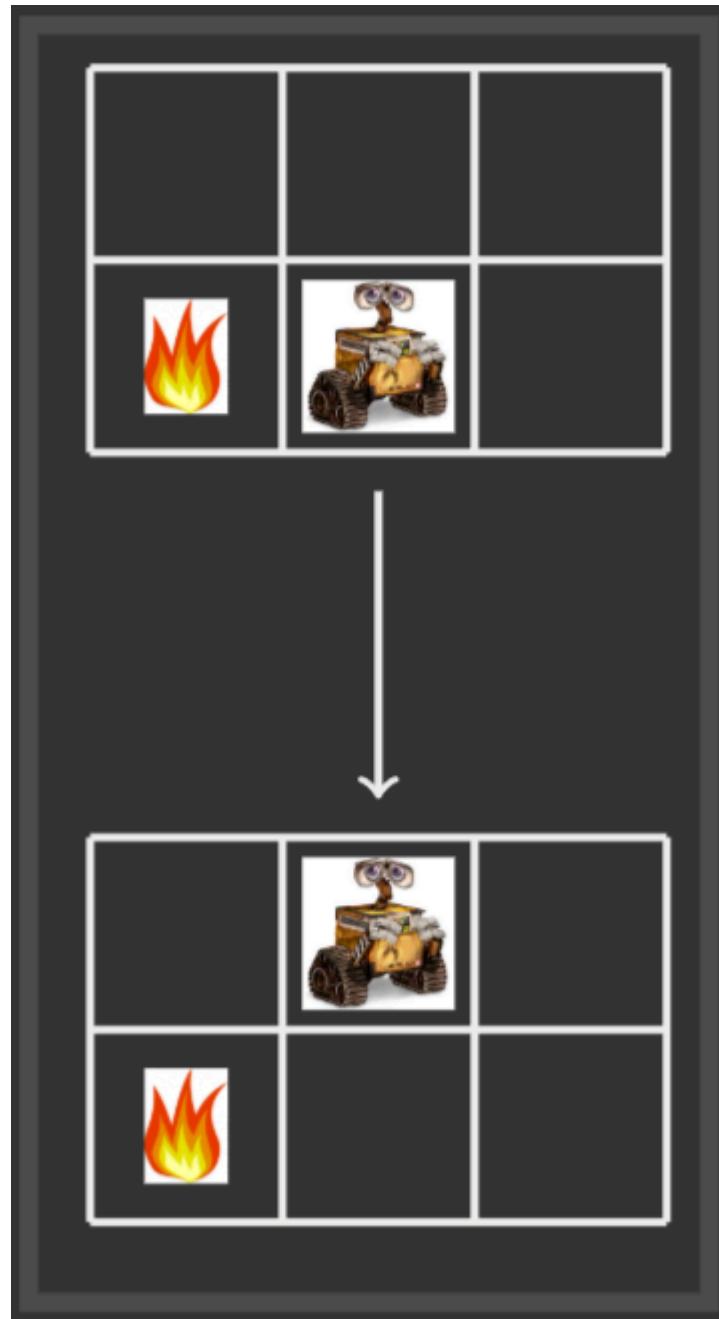
- 移动具有“噪音 noisy”：例如向北的动作，80% 概率实际向北（若无墙），10% 概率向西，10% 概率向东；若目标方向有墙，智能体保持不动。

### 奖励机制：

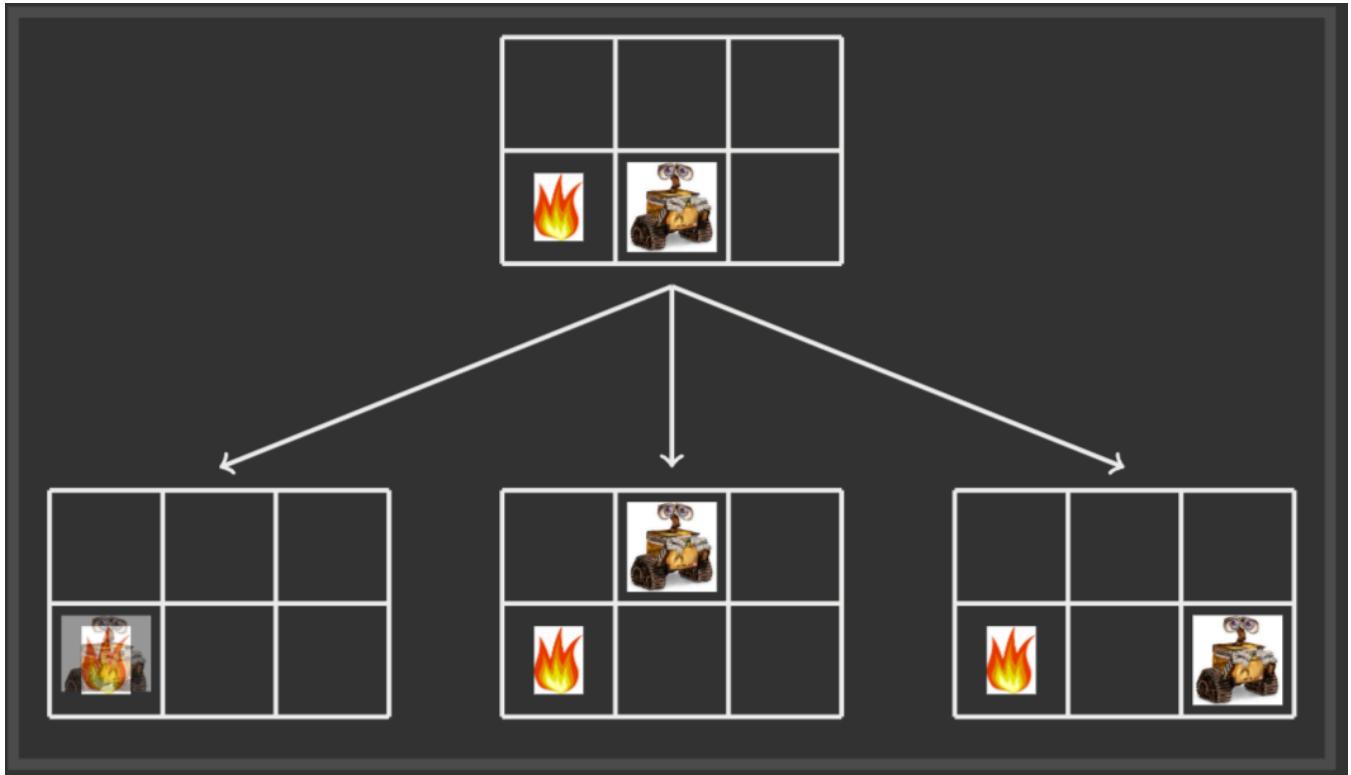
- 每一步获得小额“生存”奖励（可能为负）。
- 到达终点时获得大额奖励（可能是好的，如钻石；或坏的，如火焰）。

**目标：**最大化累计奖励总和。

In **Deterministic Grid World**, agents perform actions with uniquely determined outcomes 在**确定性网格世界**，智能体执行动作后，结果唯一确定



In the Stochastic Grid World (SGW), on the other hand, the outcome of an action performed by an agent is random in nature, with a variety of possibilities 而在随机网格世界 (Stochastic Grid World)，智能体执行动作后，结果具有随机性，存在多种可能



## MDP

### Definition

- 状态集合 (States S)**: The set of all possible states 所有可能状态的集合, 用 ( $s \in S$ ) 表示。例如网格世界中 智能体的位置。
- 动作集合 (Actions A)**: The set of all actions that an agent can perform 智能体可执行的所有动作的集合, 用 ( $a \in A$ ) 表示 (如上下左右移动)。
- 转移函数 (Transition Function ( $T(s, a, s')$ ))**: Describe the probability of moving from state  $s$  to state  $(s')$  after performing action  $a$  ( $P(s'|s, a)$ ) 描述从状态  $s$  执行动作  $a$  后转移到状态  $(s')$  的概率 ( $P(s'|s, a)$ ), 也称为**模型the model** 或**动态 the dynamics**特性, 体现了环境的不确定性。
- 奖励函数 (Reward Function ( $R(s, a, s')$ ))**: Reward for performing action  $a$  to transfer state from  $s$  to  $(s')$  执行动作  $a$  使状态从  $s$  转移到  $(s')$  时获得的奖励, 有时简化为 ( $R(s)$ ) 或 ( $R(s')$ )。
- 起始状态 (Start State ( $s_0$ ))**: Initial state at the beginning of the agent 智能体开始的初始状态。
- 终止状态 (Terminal State, 可选)**: 如到达目标 (如钻石) 或陷阱 (如火焰) 时的结束状态。

MDP problem is a non-deterministic search problem, i.e., the outcome of the action is probabilistic (e.g., "noisy movement" in a grid world)

MDP问题 属于**非确定性搜索问题**, 即动作结果具有概率性 (如网格世界中的“噪音移动”).

一种求解方法是**期望最大化搜索 (expectimax search)**

"Markov" generally means that given the present state, the future and the past are independent

“马尔可夫”意味着给定当前状态，未来与过去相互独立。对于马尔可夫决策过程，其“马尔可夫性”指动作的结果仅取决于当前状态

$$p(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \dots, S_0 = s_0) = p(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

这类似于搜索中的后继函数，仅依赖当前状态，而非历史信息

MDP中的关键量：

**策略 (Policy)**：为每个状态选择动作的规则，即 (Policy = Choice of action for each state)。

**效用 (Utility)**：(折扣) 奖励的总和，即 (Utility = sum of (discounted) rewards)，用于衡量决策序列的价值。

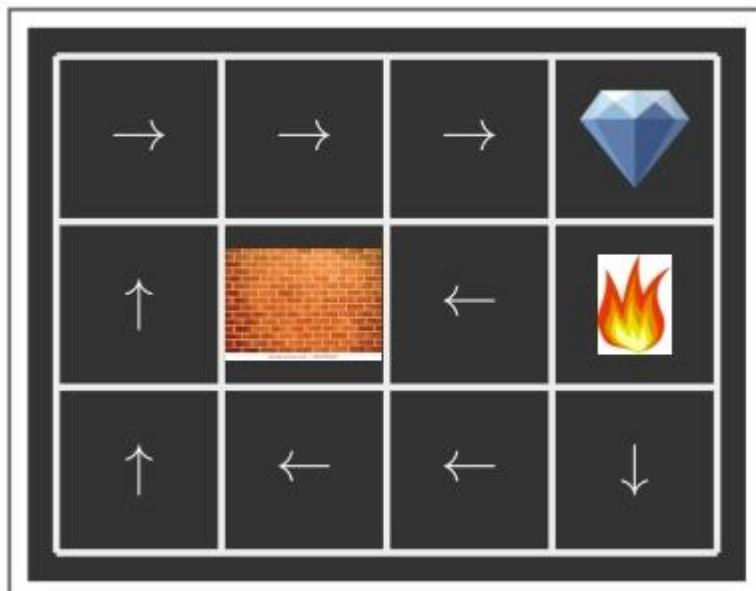
## Polices

在确定性单智能体搜索问题中，目标是找到从起始到目标的最优计划 optimal plan (即动作序列 sequence of actions)

而在 MDP 中，目标是找到一个**最优策略 the optimal policy** ( $\pi^* : S \rightarrow A$ )，它将每个状态映射到一个动作

- Strategy ( $\pi$ ) Assign an action to each state 策略 ( $\pi$ ) 为每个状态指定一个动作。
- The optimal strategy is the one that maximises expected utility when followed 最优策略是指遵循后能最大化期望效用的策略。
- An explicit policy defines a reflex agent, i.e. a reflex agent that directly executes the action specified by the policy based on the current state. 显式策略定义了一个反射型智能体，即根据当前状态直接执行策略指定的动作。

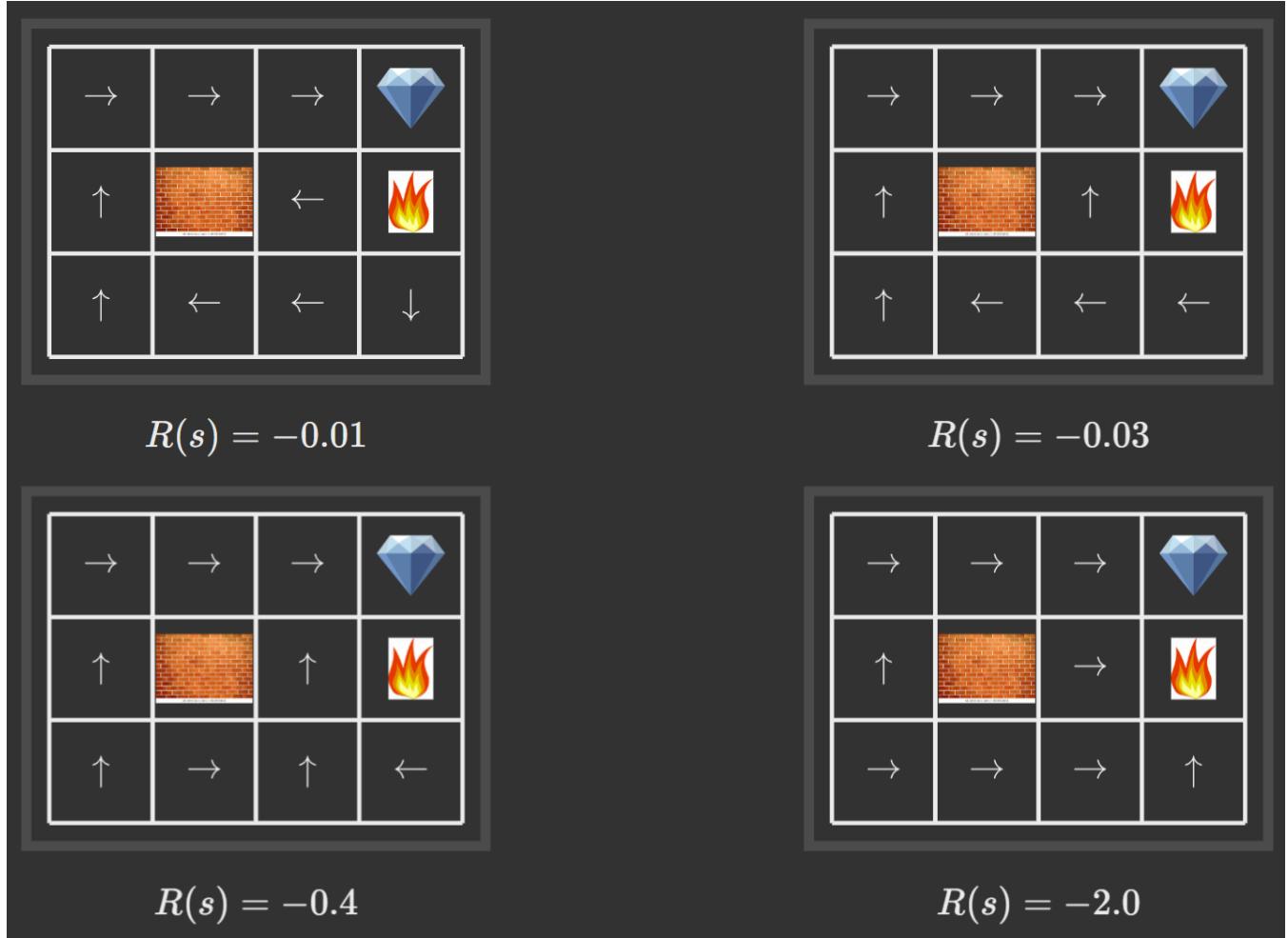
一个最优策略的例子：当设置所有非终止状态的奖励 ( $R(s, a, s') = -0.01$ ) 时，智能体在不同状态下应执行的动作



In contrast the Expectimax algorithm does not compute the full policy, it only computes actions for individual states rather than generating a global policy for all states

相比之下Expectimax 算法并不计算完整的策略，它仅为单个状态计算动作，而非为所有状态生成全局策略

当采取不同策略时，智能体采取的动作也相应改变



## Example: Car Racing

**目标：**机器人赛车希望快速地行驶更远

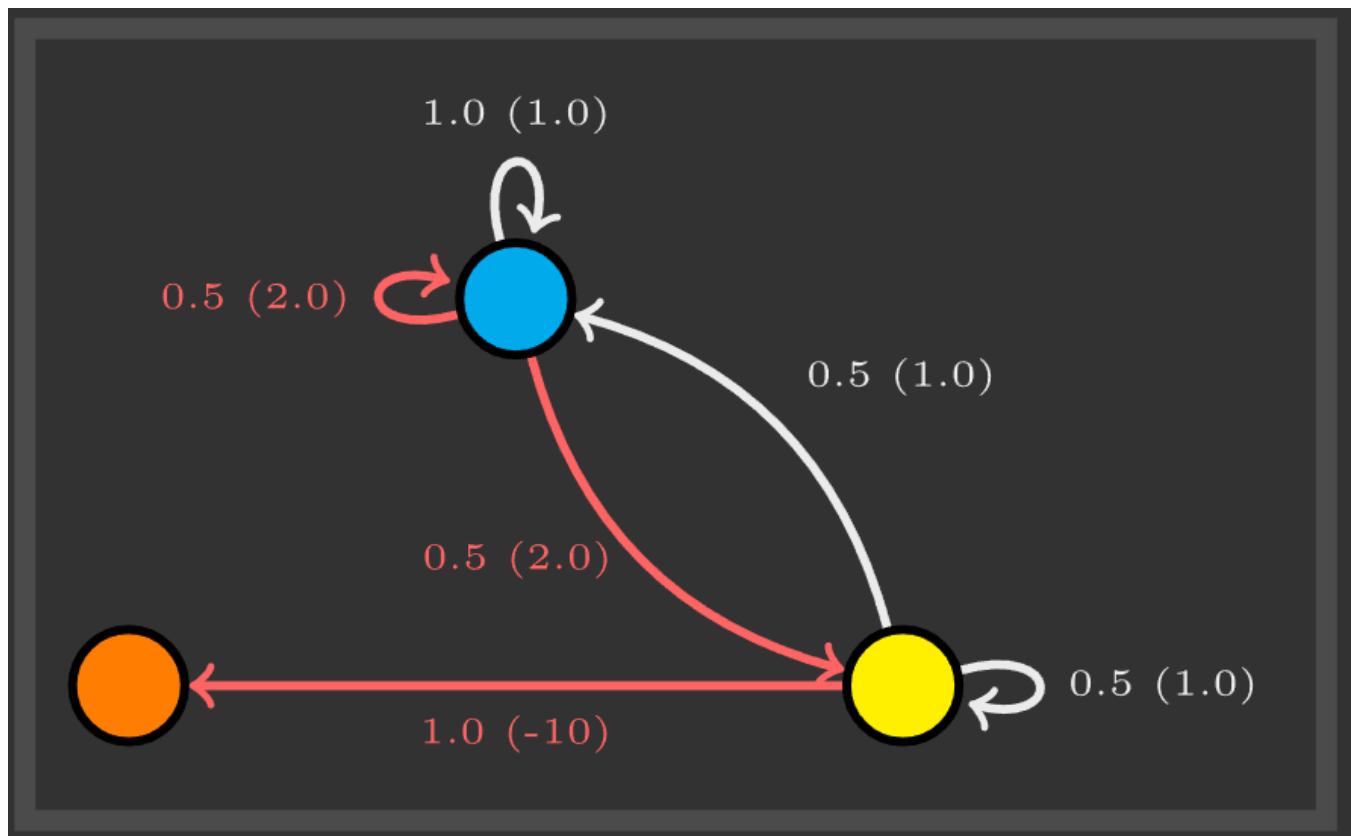
**状态：**包含三种状态，分别为 (Cool) (冷)、(Warm) (温)、(Overheated) (过热)

**动作：**包含两种动作，分别为 (Slow) (慢)、(Fast) (快)，其中快速行驶可获得双倍奖励

**具体：**

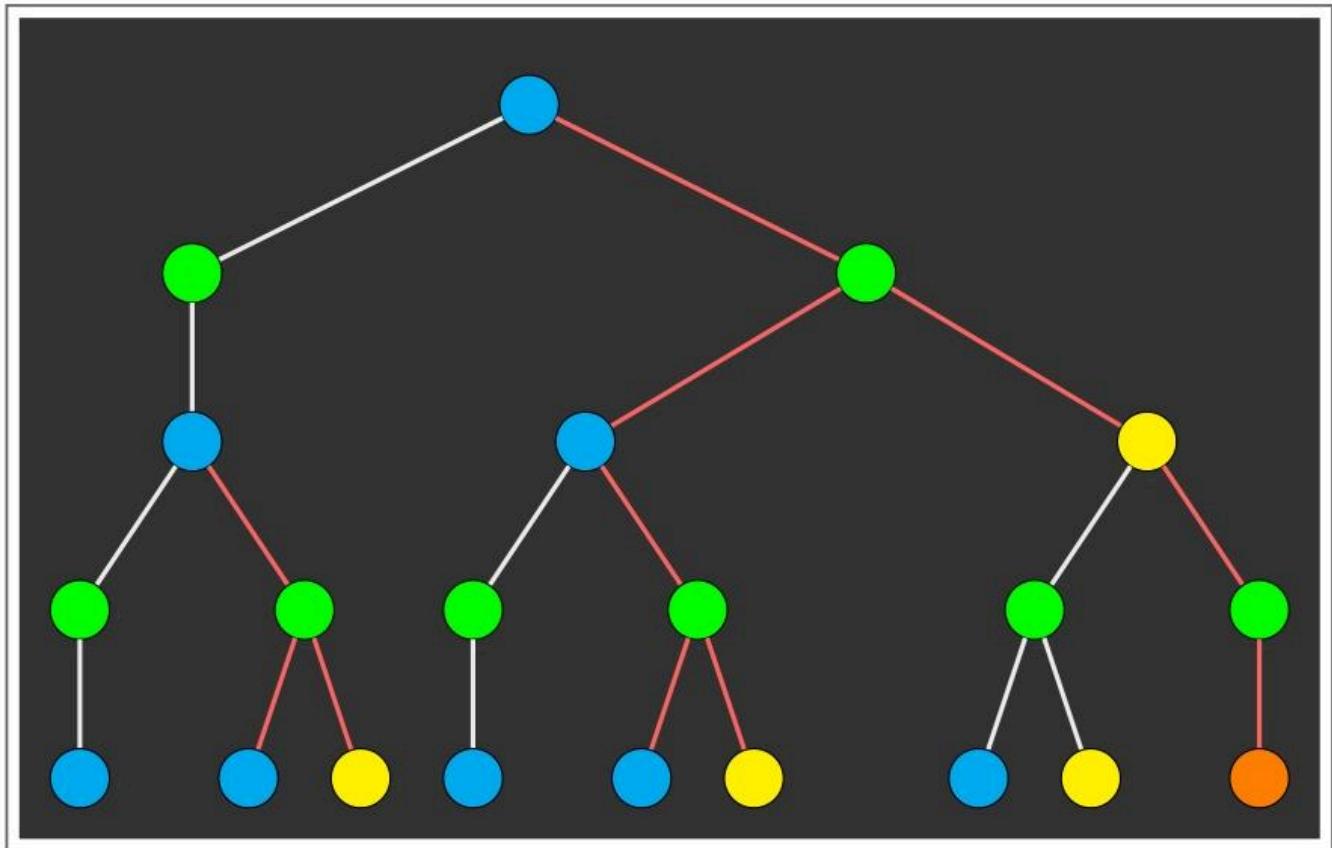
- (Cool) (蓝色)：
  - 执行 (Slow) (白色箭头)：以概率 (1.0) 保持 (Cool)，奖励为 (1.0)。
  - 执行 (Fast) (红色箭头)：以 (0.5) 概率保持 (Cool)，奖励 (2.0)；以 (0.5) 概率转移到 (Warm) (黄色)，奖励 (2.0)。
- (Warm) (黄色)：
  - 执行 (Slow) (白色箭头)：以 (0.5) 概率保持 (Warm)，奖励 (1.0)；以 (0.5) 概率转移回 (Cool)，奖励 (1.0)。

- 执行 (Fast) (红色箭头): 以概率 (1.0) 转移到 (Overheated) (橙色), 奖励 (-10)。
- (Overheated) (橙色): 图中未明确展示该状态下的动作与转移, 但其通过 (Fast) 动作从 (Warm) 转移而来, 且奖励为 (-10), 体现了过热的负面后果。



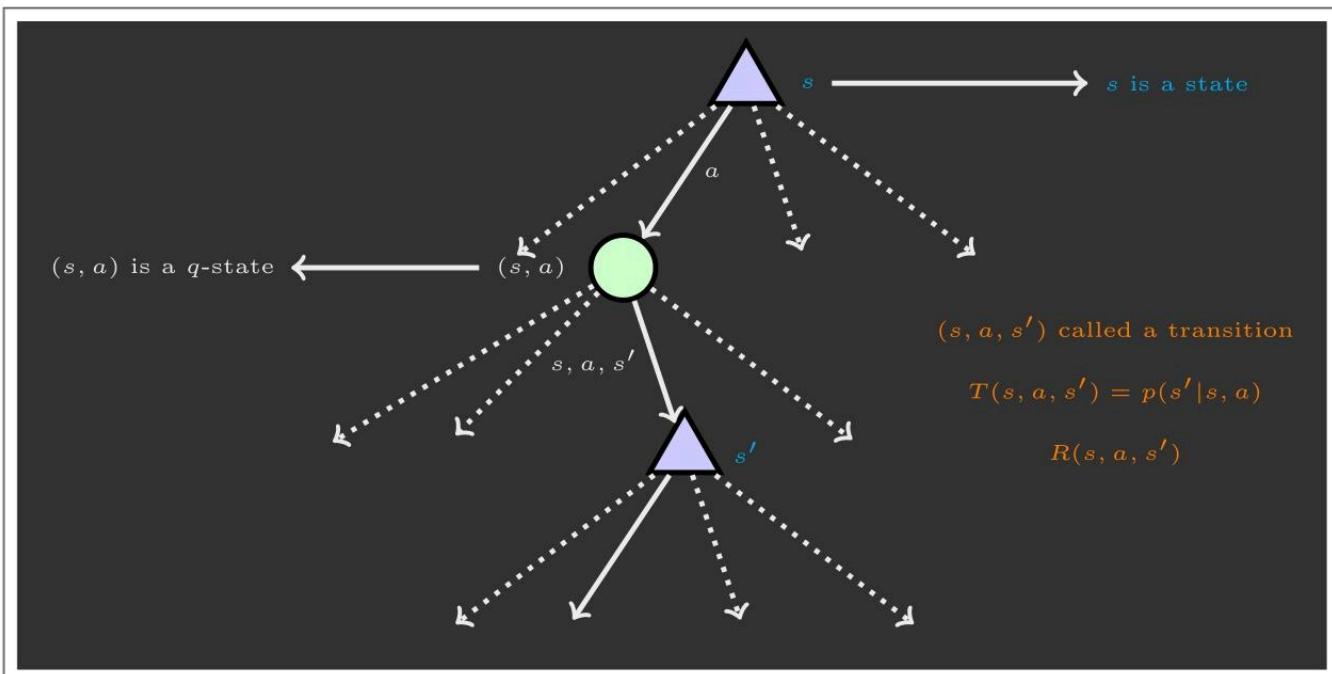
对以上过程, 有搜索树, 其中红色代表加速行驶, 白色代表缓速行驶

绿色代表概率点



更普遍地，有以下抽象MDP普遍搜索树：

- **三角形节点 ( $s$ )**: 代表一个状态 (state)，是智能体在环境中所处的情境。
- 圆形节点( $(s, a)$ ): 称为 q-state，代表 “状态 - 动作” 对，即智能体在状态  $s$  下执行动作  $a$
- **转移( $s, a, s'$ )**: 从  $(s, a)$  到下一个状态  $s'$  的过程称为转移。
  - **转移函数  $T(s, a, s') = p(s'|s, a)$** : 表示在状态  $s$  执行动作  $a$  后，转移到状态  $(s')$  的概率。
  - **奖励函数  $R(s, a, s')$** : 表示在状态  $s$  执行动作  $a$  并转移到  $(s')$  时获得的奖励



## Utility of Sequences

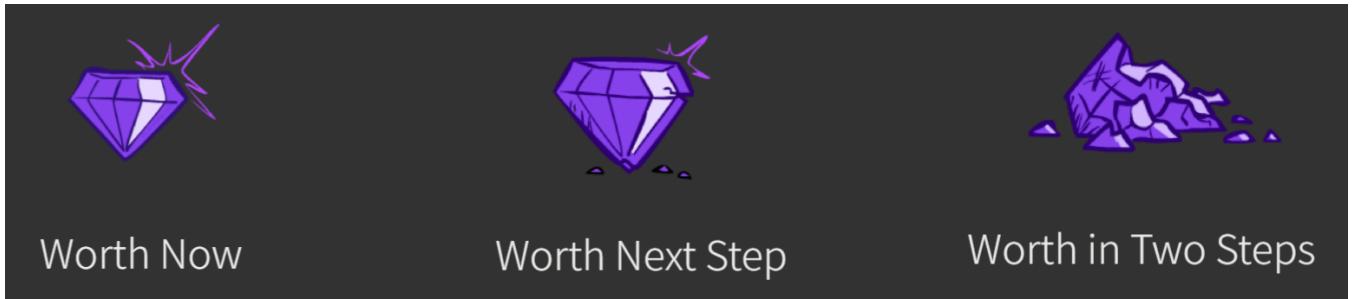
Reward sequence metrics are also important 奖励序列衡量标准也很重要

奖励多少？比较不同奖励序列的总量，如  $[1,2,2]$  与  $[2,3,4]$ ，后者总和更大

奖励时机？奖励集中在前期还是后期，如  $[0,0,1]$  是后期获得奖励， $[1,0,0]$  是前期获得奖励

最大化奖励总和是合理的目标，但我们通常希望更偏好当前奖励（即“即时奖励优于未来奖励”），也就是说会更偏好  $[1,0,0]$  而不是  $[0,0,1]$

解决这一矛盾的方法是让奖励价值呈指数衰减 **decay exponentially**，即随着时间推移，未来奖励的实际价值逐渐降低



- **折扣方式 (How to discount?)**: 每推进一个时间步（每降一层），奖励乘以一次折扣因子，实现奖励价值的衰减。
- **折扣原因 (Why discount?)**
  - 即时奖励通常比未来奖励具有更高的效用。
  - 折扣机制有助于算法收敛，避免因无限期未来奖励导致计算不收敛。

以折扣因子 (0.5) 为例：

- 计算序列  $([1, 2, 3])$  的效用： $(U([1, 2, 3])) = 1 \times 1 + 0.5 \times 2 + 0.25 \times 3)$ 。
- 比较  $(U([1, 2, 3]))$  与  $(U([3, 2, 1]))$ ，由于折扣因子的作用，将较大奖励置于前期的序列  $([3, 2, 1])$  效用更高，即  $(U([1, 2, 3]) < U([3, 2, 1]))$  Sooner rewards probably do have higher utility than later

### 平稳偏好 (Stationary Preferences)

若奖励序列  $([a_1, a_2, \dots])$  优于  $([b_1, b_2, \dots])$ ，则在两个序列前添加相同奖励  $r$  后，新序列  $([r, a_1, a_2, \dots])$  仍优于  $([r, b_1, b_2, \dots])$

对以下两种效用定义方式都适用

1. **加法效用 (Additive utility)**：直接将所有奖励相加，公式为  $(U([r_0, r_1, r_2, \dots])) = r_0 + r_1 + r_2 + \dots)$ 。
2. **折扣效用 (Discounted utility)**：对未来奖励按指数衰减计算，公式为  $(U([r_0, r_1, r_2, \dots])) = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots)$ ，其中  $(\gamma)$  为折扣因子，体现对即时奖励的偏好。

### 无限效用 (INFINITE UTILITIES) 问题

若决策过程无限持续，是否会导致奖励无限大

- 解决方案：

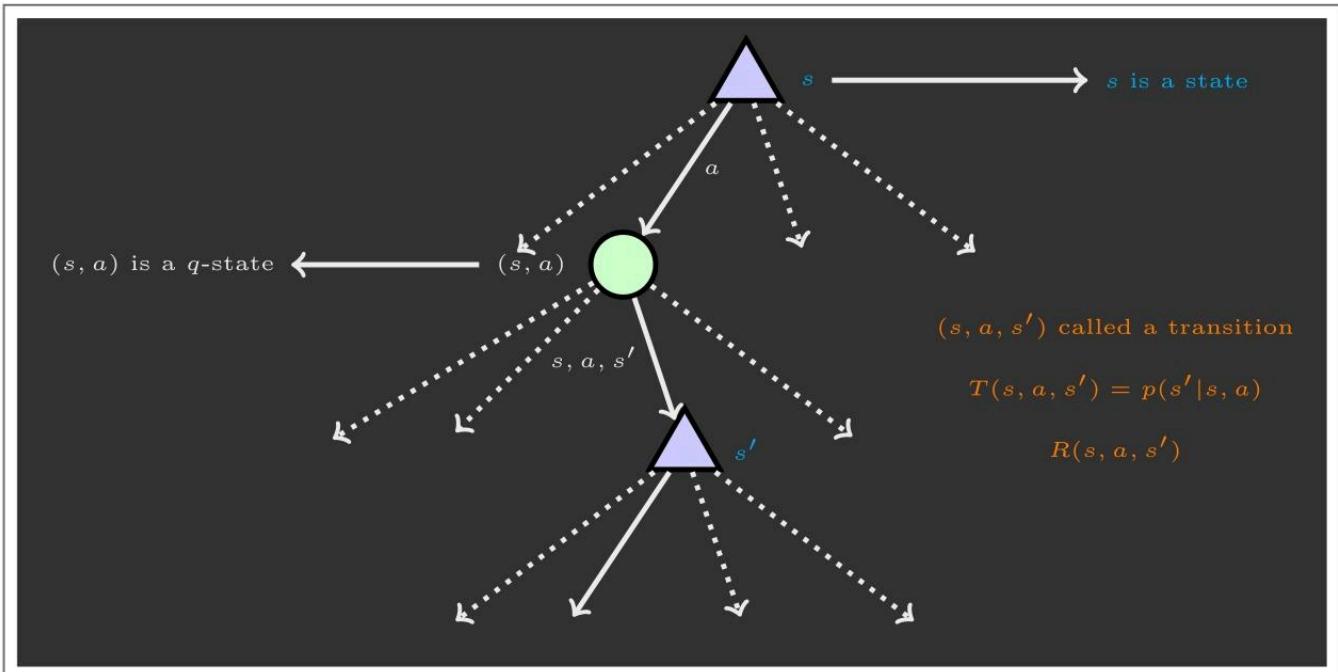
1. **有限视野 (Finite horizon)**: 类似深度受限搜索，在固定步数  $T$  后终止（如生命有期限）。此方法会产生非平稳策略（策略  $\pi$  依赖剩余时间）。
2. **折扣 (Discounting)**: 引入折扣因子 ( $0 < \gamma < 1$ )，计算效用 ( $U([r_0, \dots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{R_{\max}}{1-\gamma}$ )，确保效用有界。 $(\gamma)$  越小，“视野”越小，更关注短期奖励。
3. **吸收态 (Absorbing state)**: 保证任何策略最终都会到达终止状态（如赛车问题中的“过热”状态），避免无限循环。

# 14. Markov Decision Process - II



豆包 你的 AI 助手，助力每日工作学习

## Solving MDPs



马尔可夫决策过程 (MDP) 中三个关键的最优量：

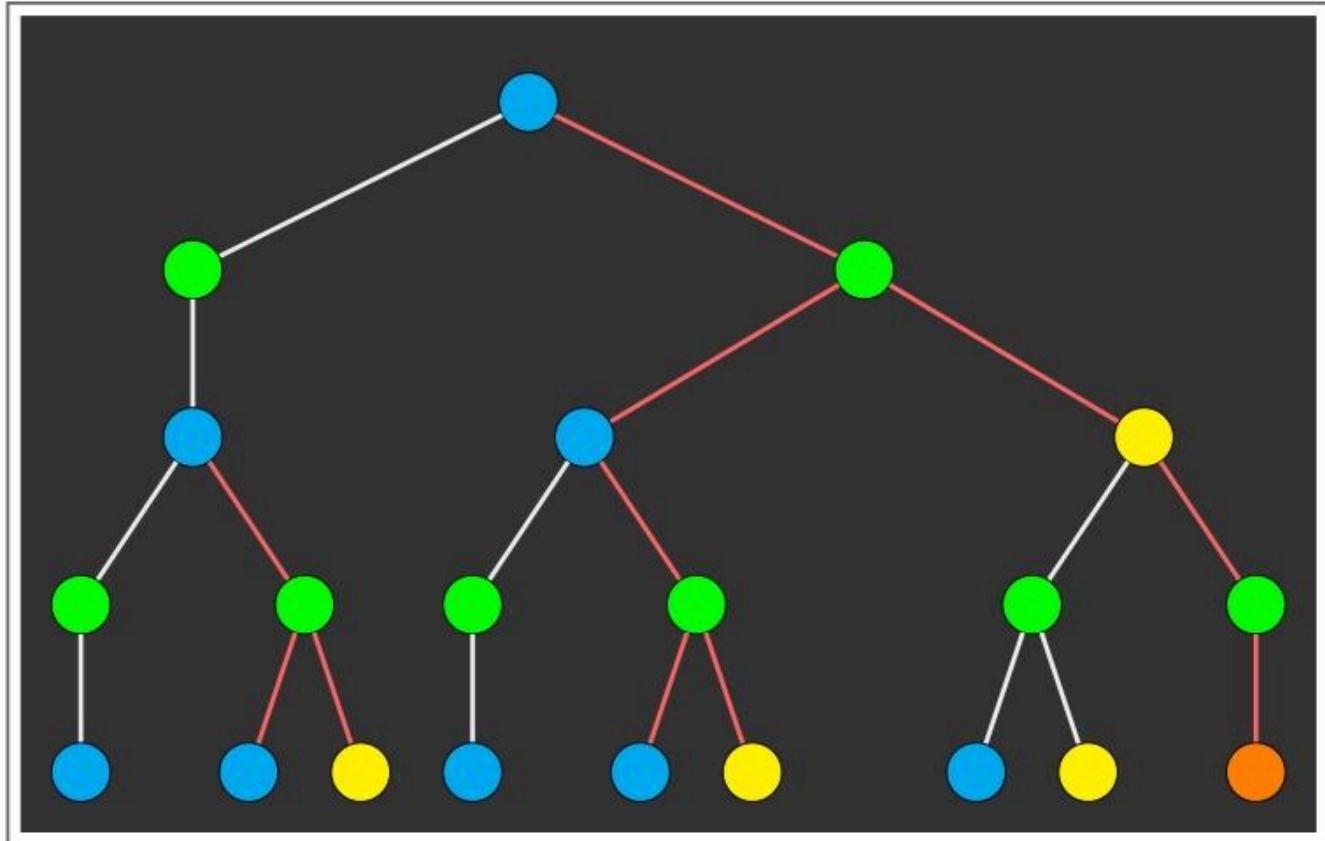
1. **状态的最优值函数  $V^*(s)$** : 表示从状态  $s$  出发, 采取最优策略 (acting optimally) 时的**期望效用** (expected utility)。数学上, 它刻画了状态  $s$  的“长期价值”, 是后续最优决策的综合体现
2. **状态 - 动作对的最优值函数  $Q^*(s, a)$** : 表示从状态  $s$  执行动作  $a$  后, 再采取最优策略时的期望效用。它评估了在状态  $s$  下选择动作  $a$  的“即时 + 长期”价值
3. **最优策略  $\pi^*(s)$** : 定义为在状态  $s$  下应采取的最优动作 (optimal action), 即通过  $\pi^*(s)$  的选择, 最大化  $V^*(s)$  或  $Q^*(s, a)$

这些概念是求解 MDP 最优决策的基础, 其中  $V^*(s)$  和  $Q^*(s, a)$  用于量化状态和动作的价值, 而  $\pi^*(s)$  则直接指导决策行为

$$\begin{aligned}
 (1) \quad & V^*(s) = \max_a Q^*(s, a) \\
 (2) \quad & Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')] \\
 & \text{or } Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s' | s, a) V^*(s') \\
 (3) \quad & V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')] \\
 & \text{or } V^*(s) = \max_a [R(s, a, s') + \gamma \sum_{s'} P(s' | s, a) V^*(s')]
 \end{aligned}$$

- (1) 状态  $s$  的最优值函数  $V^*(s)$  是其所有动作  $a$  对应的最优动作 - 状态值函数  $Q^*(s, a)$  的最大值, 建立了  $V^*(s)$  与  $Q^*(s, a)$  的关系
- (2) 动作 - 状态值函数  $Q^*(s, a)$  表示在状态  $s$  执行动作  $a$  后, 转移到下一状态  $s'$  的期望奖励  $R(s, a, s')$  与折扣后后续状态值  $\gamma V^*(s')$  的总和, 其中  $T(s, a, s')$  是状态转移概率。相比  $V^*(s)$ , 更强调了动作的作用, 更适合动作决策
- (3) **Bellman 最优方程**, 将  $Q^*(s, a)$  代入  $V^*(s)$  的定义, 通过最大化所有可能动作  $a$  的期望未来奖励 (含折扣), 递归地定义了状态  $s$  的最优值函数, 体现了当前决策对未来状态值的影响

## Racing Search Tree



两个问题及解决思路：

### 问题 1：状态重复 (States are repeated)

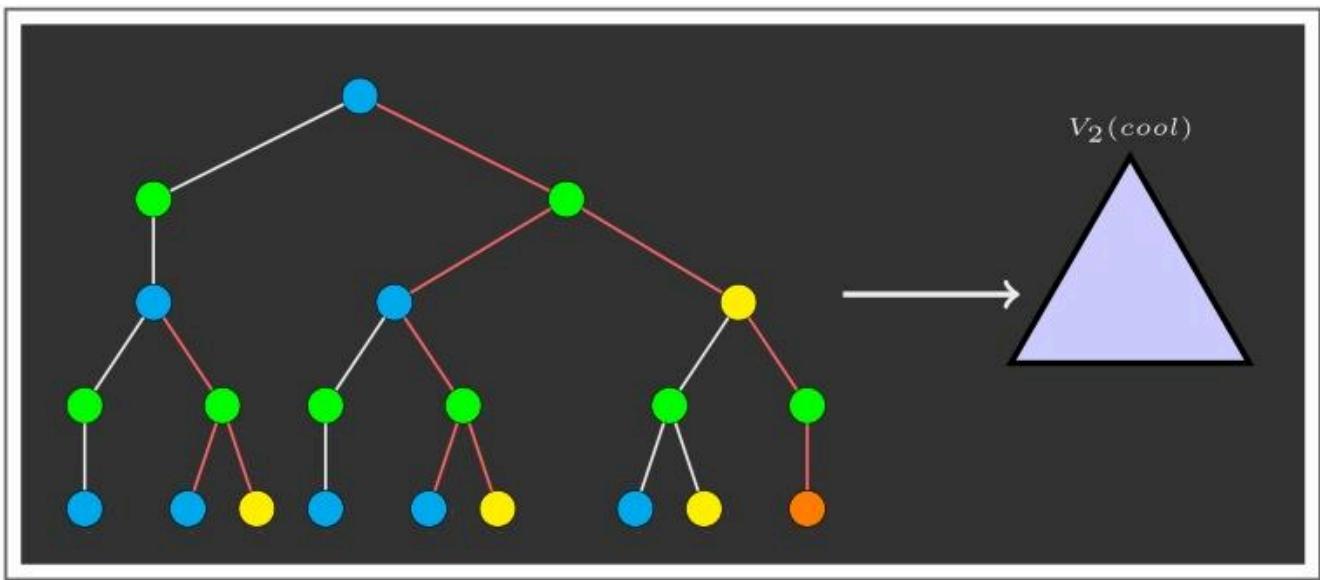
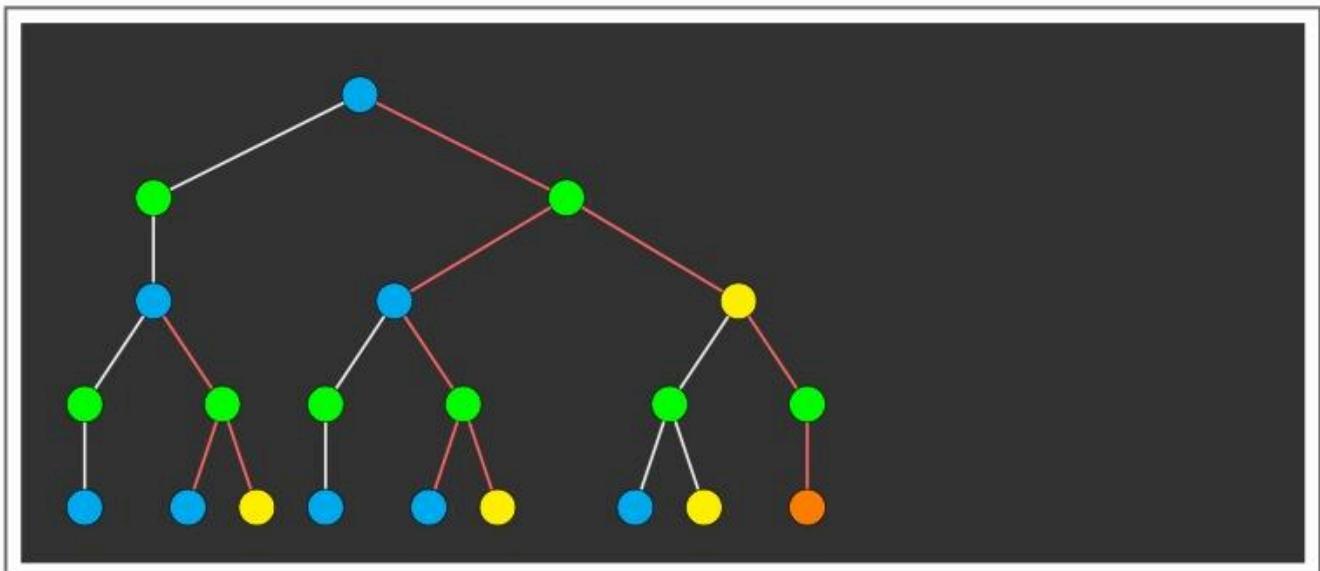
- 现象：计算过程中存在大量重复状态的计算，导致冗余。
- 解决思路：仅计算一次所需量 (Only compute needed quantities once)，避免重复计算以提高效率。

### 问题 2：搜索树无限延伸 (Tree goes on forever)

- 现象：搜索树若不限制深度，会无限扩展，难以终止计算。
- 解决思路：进行深度有限的计算，但逐步增加深度，直至计算结果变化微小。此外，若折扣因子 ( $\gamma < 1$ )，树的深层部分对结果的影响最终会变得很小，因此无需过度深入计算。、

引入“时间有限值 time-limited values”，定义  $V_k(s)$  为若游戏在接下来  $k$  个时间步结束时，状态  $s$  的最优值

$V_k(s)$  等同于从状态  $s$  进行深度为  $k$  的期望最大化 (expectimax) 计算所得的值。这一方法通过限制搜索深度，避免搜索树无限延伸，从而在有限计算资源下近似求解状态的最优值，提升计算效率



## Value Iteration 重要

- **初始化:** 从  $V_0(s) = 0$  开始, 意味着没有剩余时间步时, 期望奖励总和为零。这是迭代的起点, 为每个状态赋予初始值
- **迭代更新:** 给定  $V_k(s)$ , 对每个状态执行一次期望最大化 (expectimax) 计算, 更新公式为:

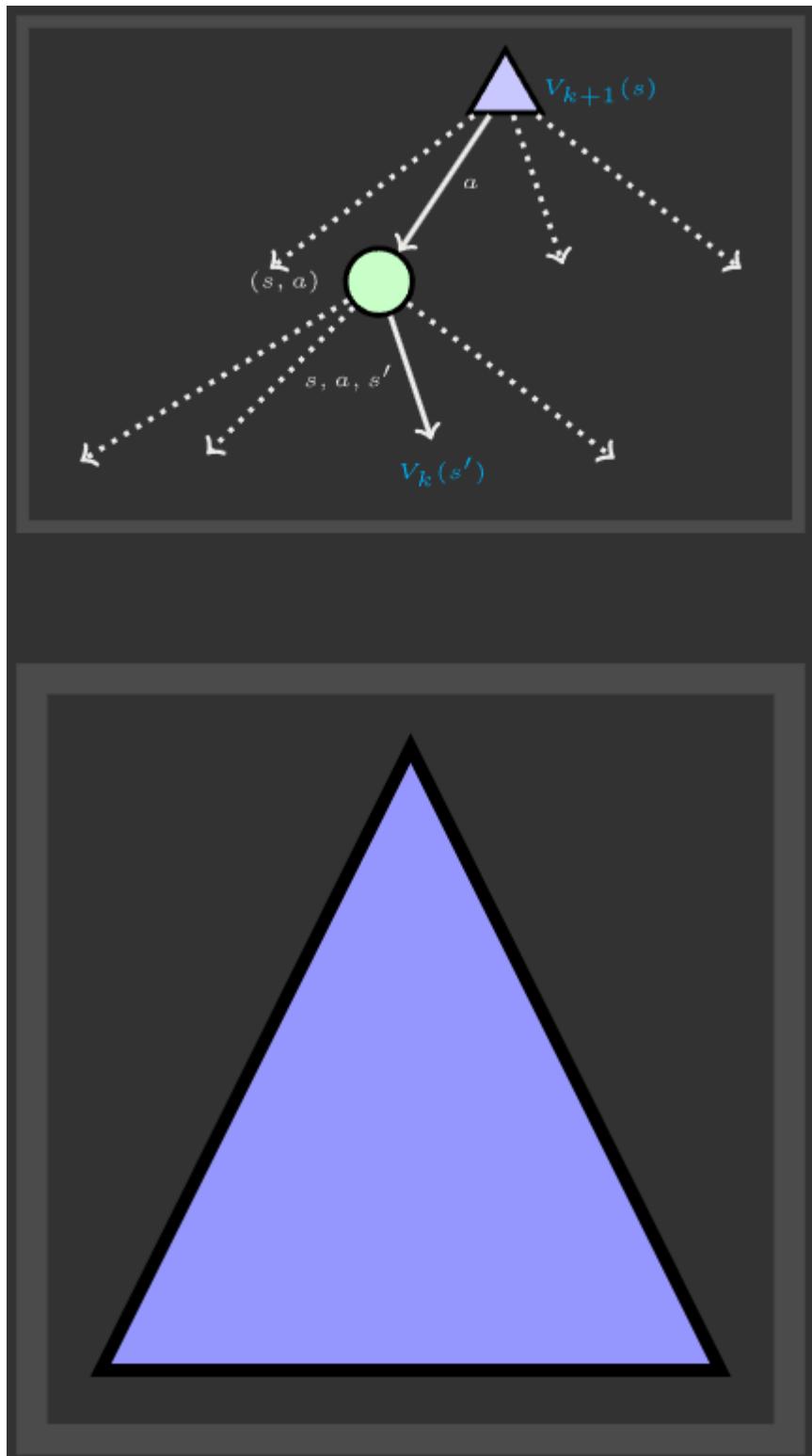
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

$$V_{k+1}(s) = \max_a [R(s, a, s') + \gamma \sum_{s'} P(s' | s, a) V_k(s')]$$

- **终止条件:** 重复上述更新过程, 直至收敛

每次迭代的时间复杂度为  $\mathcal{O}(S^2 A)$ , 其中  $S$  为状态数,  $A$  为动作数

该算法的基本思想是通过迭代逐步细化近似值, 使其趋近于最优值, 但策略 (Policy) 可能在值完全收敛之前就已收敛

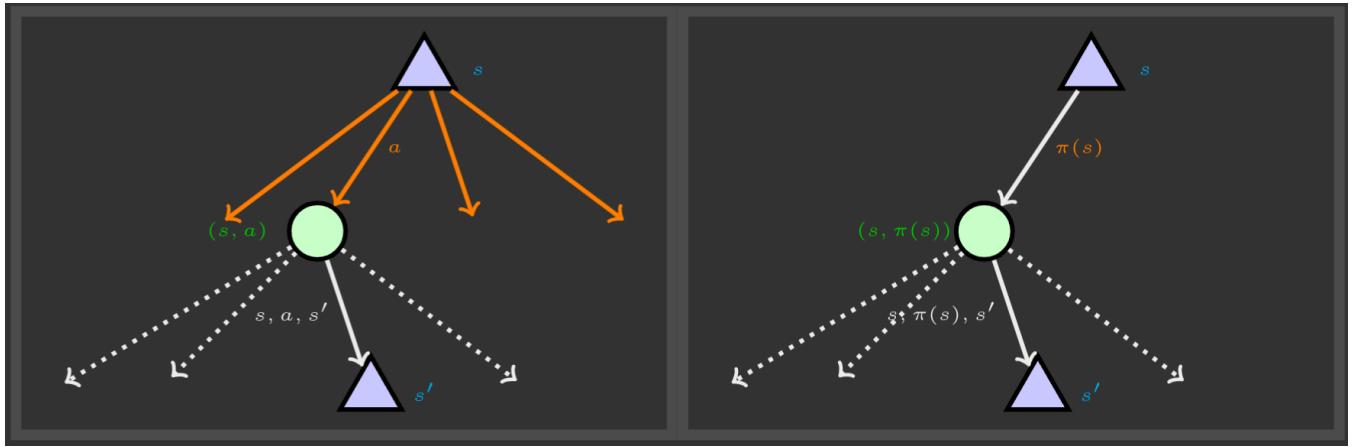


如何收敛?

- Case 1: 搜索树有最大深度 M
- Case 2: 折扣因子  $\gamma < 1$

## Policy Evaluation

若固定策略  $\pi(s)$ , 每个状态只对应一个动作, 树结构会简化, 但树的值依赖于所固定的策略 (右图)



为评估策略，需计算固定策略下的效用

在固定策略  $\pi$  下，状态  $s$  的效用  $V^\pi(s)$  定义为：从状态  $s$  出发并遵循策略  $\pi$  时，期望的总折扣奖励

$$V^\pi(s) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^\pi(s')]$$

两种解法：

1. **迭代更新法（类似值迭代）**：每次迭代时间复杂度为  $\mathcal{O}(S^2)$ ，由于无需对动作取最大值（固定策略），计算量简化

2. **线性系统求解法**：可借助 Matlab 或其他线性系统求解器直接求解该线性方程组

## Policy Extraction

已知最优状态值函数  $V^*(s)$ ，探讨如何据此确定最优行动策略

通过“一步迷你期望最大化（mini - expectimax）”计算

$$\pi^*(s) = \operatorname{argmax}_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

这一过程从最优值函数  $V^*(s)$  中提取出隐含的最优策略  $\pi^*(s)$

以上过程有简化形式，即设状态-动作价值函数：

$$Q^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

有：

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$$

从  $Q$  值选择动作比从状态值 ( $V$  值) 选择动作更容易，因为  $Q^*(s, a)$  直接刻画了在状态  $s$  下执行动作  $a$  的期望效用，可直接通过比较  $Q$  值大小做出决策；而从  $V$  值选择动作，需通过

$\pi^*(s) = \arg \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$  进行一步期望计算，过程相对间接

# Policy Iteration

---

值迭代解法具有局限性：

- **计算速度慢**：每次迭代的时间复杂度为  $\mathcal{O}(S^2 A)$ ，效率低
- “**最大化**”操作更新冗余：在迭代过程中，每个状态下使值函数最大化的动作（“max”操作结果）往往很早就趋于稳定，后续仍持续更新值函数，造成不必要的计算浪费
- **策略收敛早于值函数**：实际应用中，最优策略（决策规则）通常在值函数完全收敛之前就已确定，但值迭代仍会继续执行直到值函数收敛，延长了计算时间

考虑**策略迭代 (Policy Iteration)**，其核心是通过交替进行**策略评估**和**策略改进**来求解最优策略

## 1. 策略评估 (Policy evaluation)

对某个固定策略（非最优策略）计算状态效用  $V^\pi(s)$ ，直到其收敛，即求解

$$V^\pi(s) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^\pi(s')]$$

其中  $a = \pi(s)$

评估当前策略下每个状态的长期期望折扣奖励

## 2. 策略改进 (Policy improvement)

利用收敛后的（但非最优的）效用值进行一步前瞻，更新策略

对每个状态  $s$ ，选择使  $\sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^\pi(s')]$  最行动  $a$ ，即

$$\pi_{new}(s) = \arg \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^\pi(s')]$$

不断重复上述两步，直至策略收敛（不再变化），此时得到最优策略

# Comparison

---

值迭代 (Value Iteration) 和策略迭代 (Policy Iteration)

**共同点：**两者的目标一致，均用于计算马尔可夫决策过程 (MDP) 中的所有最优值（如最优状态值函数  $V^*(s)$ ）

**不同点：**

- 值迭代：
  - 每次迭代同时更新值函数和（隐含地）策略
  - 不显性跟踪策略，而是通过对动作取最大值（max 操作）隐含地重新计算策略。例如：在值迭代公式  $V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')]$  中，每次对动作  $a$  取最大值的过程，实际上在更新策略，但未单独记录策略的变化
- 策略迭代：
  - 先进行多次策略评估，即用固定策略更新效用值（每次评估较快，因为只需考虑该策略指定的动作，而非所有动作）
  - 策略评估完成后，选择新策略（此步骤类似值迭代的一次迭代，速度较慢）

- 新策略必定更优（或已达到最优，迭代结束）

## Summary

---

马尔可夫决策过程（MDP）算法

### 目标与对应算法

- **计算最优点：**使用 **值迭代** 或 **策略迭代**。两者均致力于求解 MDP 中的最优点函数（如  $V^*(s)$ ）。
- **计算特定策略的值：**采用 **策略评估**。固定某一策略，计算该策略下各状态的效用值  $V^\pi(s)$ 。
- **将值转化为策略：**通过 **策略提取**（一步前瞻）。基于已有值函数，计算每个状态下的最优动作，从而形成策略。

## 算法的共性与差异

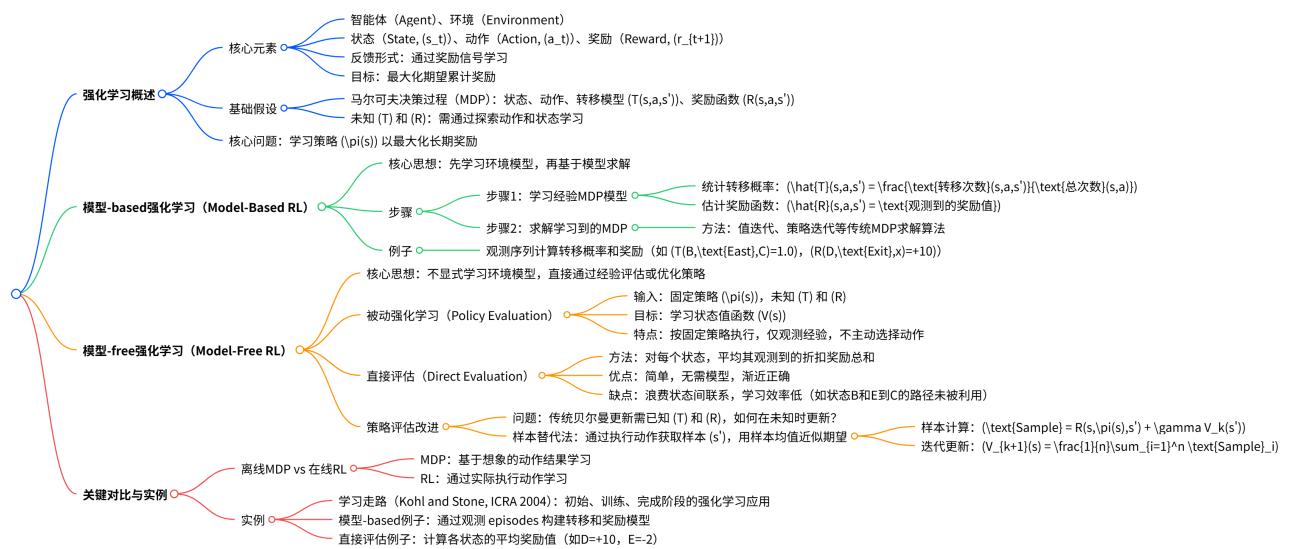
---

### • 共性

- 本质上都是 **贝尔曼更新** 的变体，核心均围绕贝尔曼方程展开。
- 都运用 **一步前瞻的期望最大化**，即在计算中考虑下一步的状态转移与奖励。

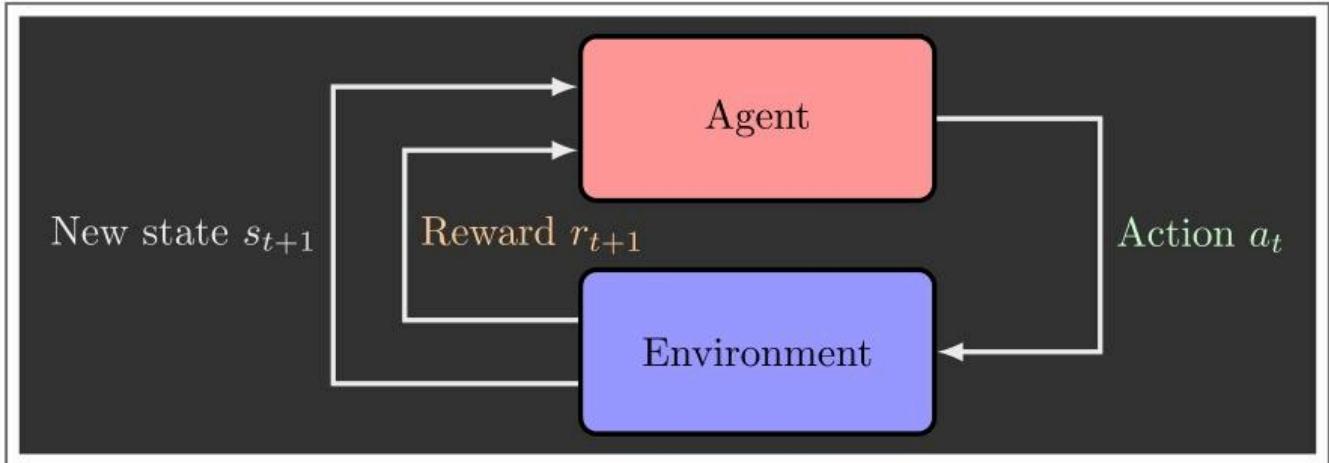
- **差异：**主要在于是否引入固定策略或对动作取最大值。例如，值迭代每次对动作执行 max 操作以更新值函数；策略评估则固定策略，不进行 max 操作，直接计算固定策略下的状态值

# 15. Reinforcement Learning - I



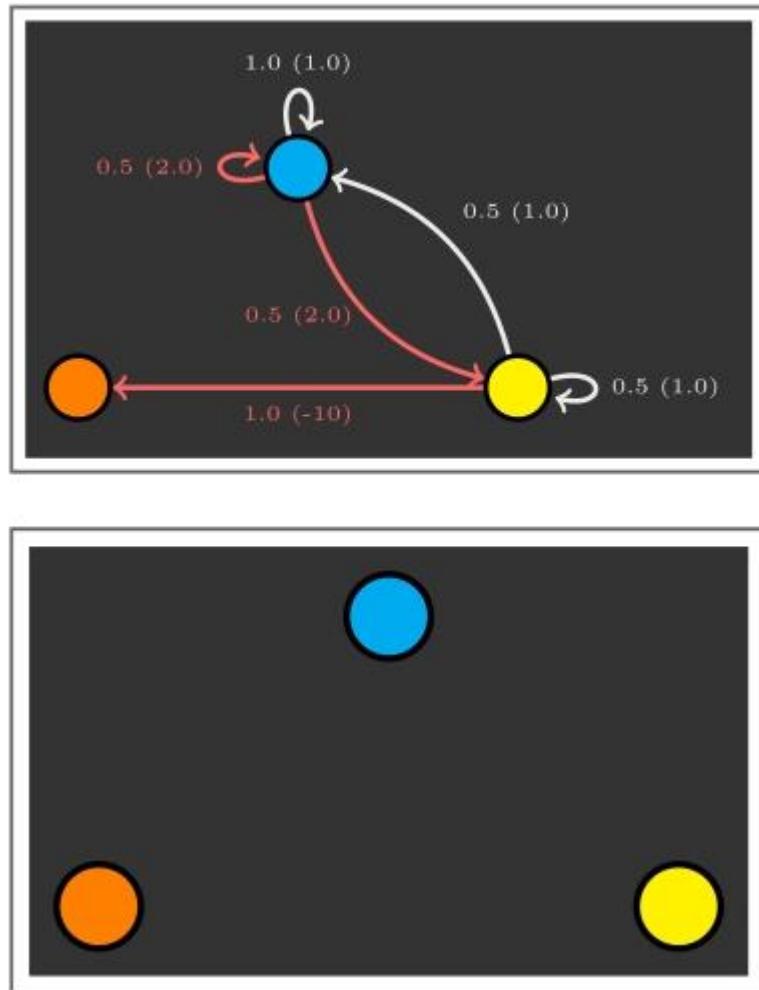
豆包  
你的 AI 助手，助力每日工作学习

## Reinforcement Learning



智能体向环境执行动作  $a_t$ , 环境则返回新状态  $s_{t+1}$  和奖励  $r_{t+1}$ , 形成交互循环

- 智能体通过 **奖励 (Reward)** 形式接收环境反馈
- 智能体的效用由奖励函数决定, 其行为目标受奖励机制引导
- 智能体必须 (学习) 采取行动, 以最大化 **期望累计奖励**, 这是强化学习的核心目标
- 所有学习过程都基于对结果样本的观察, 强调了数据驱动的学习特性



相较于MDP，强化学习未知T和R，需要通过 **尝试动作和状态** 来学习，探索环境以获取信息

MDP	RL
"Imagine what it is to take an action and learn", 即基于已知模型，通过 <b>想象或模拟</b> 动作的后果来学习和规划，无需与真实环境交互	"Actually take an action and learn", 即智能体需与 <b>真实环境实际交互</b> ，通过执行动作、观察环境反馈（如奖励、新状态）来学习，而非依赖预先设定的模型

## Model-Based Learning

核心思想 (Model - Based Idea)

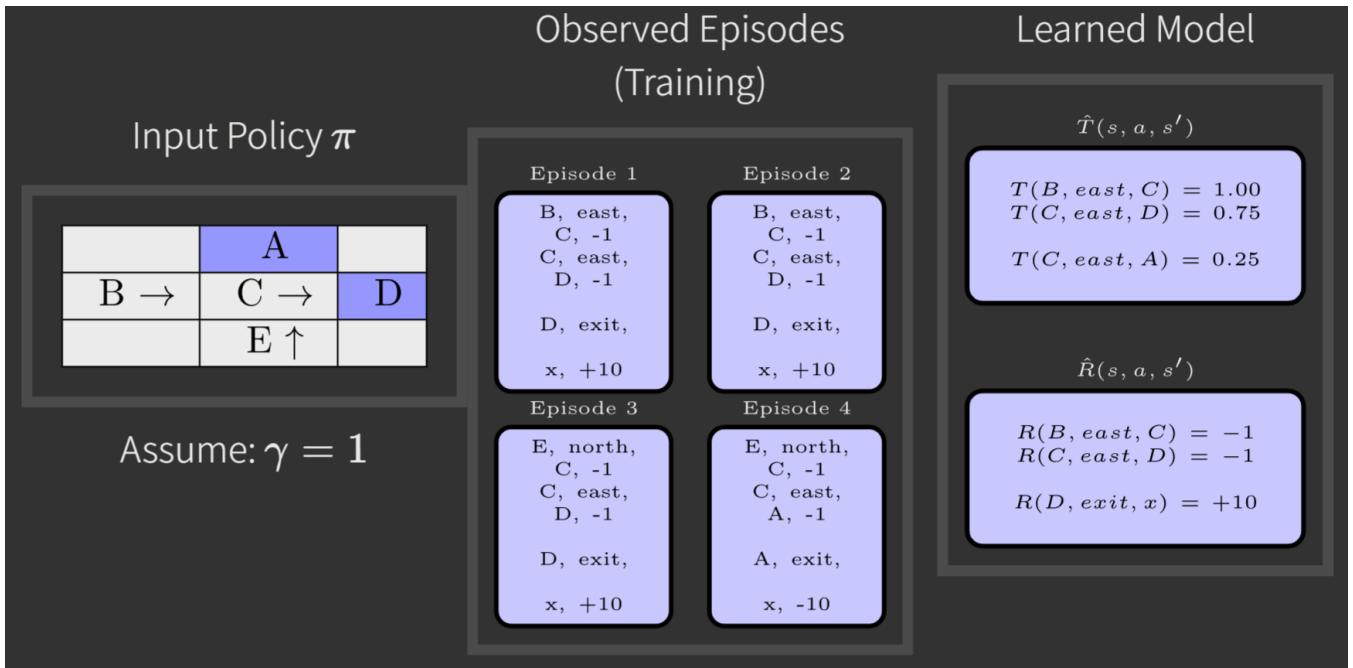
- Learning an approximate model based on experience 基于经验学习一个近似的环境模型
- 假设学习到的模型是正确的，在此基础上求解值函数

Step 1 学习经验 MDP 模型 (Learn empirical MDP model) :

- 对每一个状态  $s$  和动作  $a$ ，统计执行动作  $a$  后转移到后续状态  $(s')$  的次数。
- 对统计结果进行归一化，得到状态转移概率的**估计**  $\hat{T}(s, a, s')$ 。
- 在经历状态 - 动作 - 后续状态  $(s, a, s')$  的过程中，确定奖励函数的**估计**  $\hat{R}(s, a, s')$

## Step 2 求解学习到的 MDP (Solve the learned MDP) :

例如，使用之前的值迭代 (value iteration) 等方法，基于学习到的  $\hat{T}$  和  $\hat{R}$  求解最优策略或值函数



## Model-Free Learning

被动强化学习 (Passive Reinforcement Learning)，可简化为 策略评估 (policy evaluation)

仅给出一个固定的策略  $\pi(s)$ ，状态转移概率  $T(s, a, s')$  未知，奖励函数  $R(s, a, s')$  未知

**目标：**学习 状态值 (state values)，即评估在固定策略下各状态的价值

在这个过程中，学习者只能“跟随策略”，**不能自主选择动作**，仅执行给定策略并从经验中学习

这一过程不是离线规划 (offline planning)，需要在真实环境中实际执行动作

### 评估方法——直接评估 (Direct Evaluation)

**目标 (Goal):** 计算在固定策略  $\pi$  下每个状态的值

**核心思想 (Idea):** 对观察到的样本值进行平均

1. 按照策略  $\pi$  执行动作与环境交互
2. 每次访问一个状态时，记录该状态下 **折扣奖励总和**（即从该状态开始，后续获得的奖励经折扣后的累加值）
3. 对同一个状态的多次记录值进行平均，得到该状态的最终评估值

### 优点

- **易于理解：**直接评估的逻辑简单直观，没有复杂的计算过程。
- **无需模型知识：**不依赖状态转移概率  $T(s, a, s')$  和奖励函数  $R(s, a, s')$  的先验知识，仅通过样本转移学习。
- **渐近正确性：**通过足够多的样本转移，最终能计算出每个状态正确的平均价值。

### 缺点

- **浪费状态联系信息**: 忽略状态之间的潜在联系 (如下图中, 状态 B 和 E 都能转移到 C, 但直接评估未利用这种关联)。
- **独立学习状态**: 每个状态的价值单独学习, 未共享或利用其他状态的信息。
- **学习效率低**: 由于上述两点, 学习每个状态都需大量独立样本, 导致整体学习时间漫长。

# Output Values

	A (-10)	
B (+8) →	C (+4) →	D (+10)
	E (-2)↑	

B 和 E 在该策略下都转移到 C, 它们的值为何不同?

答: 直接评估中, 每个状态的价值是其自身经历的折扣奖励总和的平均, B 和 E 各自的奖励历史不同 (即使都到 C), 因此计算出的平均值不同, 体现了直接评估未利用状态联系、独立学习的特点

为何不直接使用策略评估 Policy Evaluation?

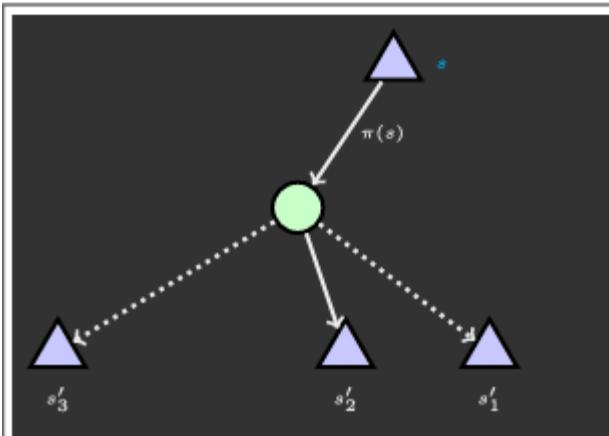
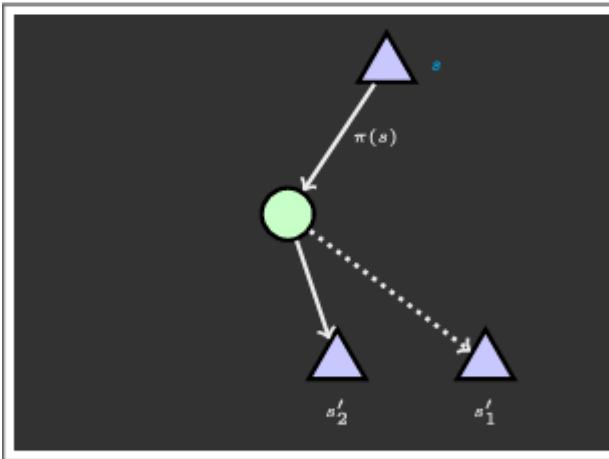
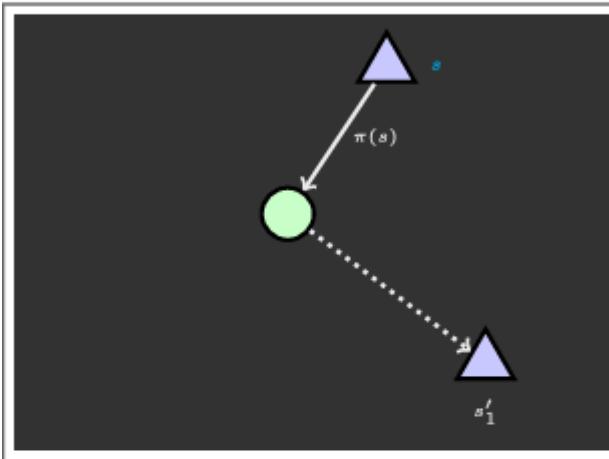
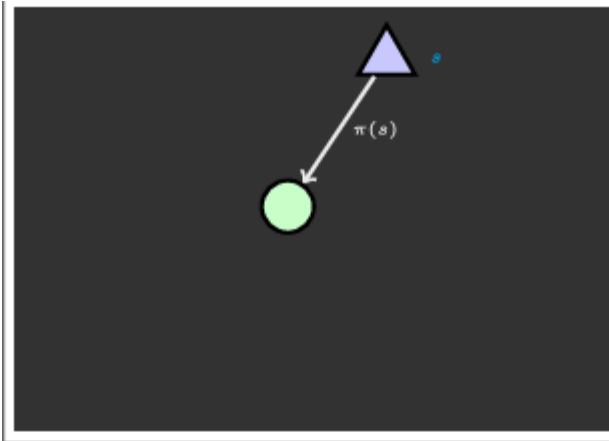
因为策略评估必须已知状态转移概率  $T(s, \pi(s), s')$  和奖励函数  $R(s, \pi(s), s')$ , 而在RL中这两者通常未知

### 评估方法——基于样本的策略评估 (Sample - Based Policy Evaluation)

传统贝尔曼更新公式为  $V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$ , 但该公式依赖已知的 T 和 R

**核心思想**: 通过执行动作获取后续状态  $s'$  的样本, 对样本值进行平均, 以此近似上述公式

1. 执行策略  $\pi(s)$  的动作, 得到多个样本  $sample_i = R(s, \pi(s), s'_i) + \gamma V_k^{\pi}(s'_i)$  ( $i = 1, 2, \dots, n$ )
2. 对样本取平均更新 V:  $V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n} \sum_i sample_i$



# 16. Reinforcement Learning - II



豆包  
你的AI助手, 助力每日工作学习

## Temporal-Difference Learning

是一种 Model-free 的强化学习方法, 通过时序差分更新值函数  $V$

从每次经验转移  $(s, a, s', r)$  中学习, 每次经历该转移时更新状态值函数  $V(s)$ 。

相较于其他后续状态, 更可能出现的后续状态  $s'$  会更频繁地影响  $V(s)$  的更新

**策略固定, 仍在评估:** 在更新过程中, 策略  $\pi$  保持不变, 仍处于对当前策略的值评估阶段

**向后继状态值移动:** 通过移动平均 (running average) 的方式, 将  $V(s)$  的值向实际出现的后继状态  $s'$  的值靠拢  
Move values toward value of whatever successor occurs

公式:

样本估计:  $\text{sample} = R(s, \pi(s), s') + \gamma V^\pi(s')$

指数移动平均更新:  $V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha \cdot \text{sample}$ , 其中  $\alpha$  是学习率

等价形式:  $V^\pi(s) = V^\pi(s) + \alpha(\text{sample} - V^\pi(s))$ , 直接体现了  $V^\pi(s)$  的更新量为学习率  $\alpha$  乘以样本值与当前值的差值

**指数移动平均 (Exponential Moving Average)** 是一种连续插值更新:

$$\bar{x}_n = (1 - \alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$$

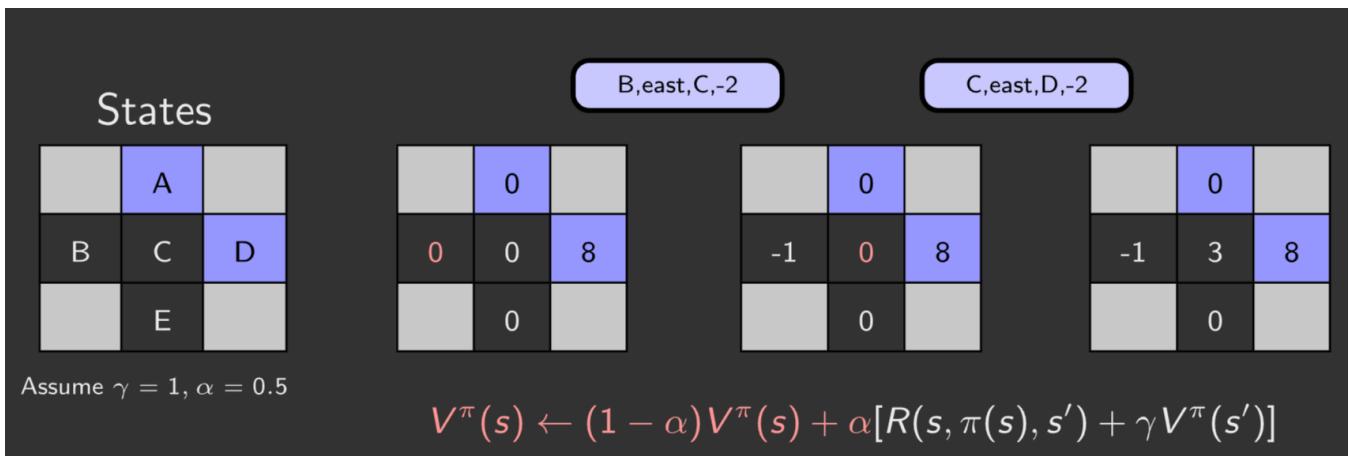
其中  $\bar{x}_n$  是当前的平均值,  $\bar{x}_{n-1}$  是上一次的平均值,  $x_n$  是当前样本值,  $\alpha$  是学习率 (权重系数)

展开后有:

$$\bar{x}_n = \frac{x_n + (1 - \alpha) \cdot x_{n-1} + (1 - \alpha)^2 \cdot x_{n-2} + \dots}{1 + (1 - \alpha) + (1 - \alpha)^2 + \dots}$$

可见越近期的样本 (如  $x_n$ ) 权重越大, 越远期的样本 (如  $x_{n-2}$  等) 权重呈指数级递减, 突出了近期数据的影响

该方法会“遗忘”遥远过去的值 (因这些值可能本身就不准确)。同时, 递减的学习率  $\alpha$  能使平均值趋于收敛, 避免因持续更新导致结果不稳定



图中  $V(B) = (1 - 0.5) \times 0 + 0.5 \times (-2 + V(C))$

TD学习是一种无模型的策略评估方法, 通过连续样本均值模拟贝尔曼更新, 用于评估策略的值函数  $V(s)$

但若想将学习到的 ( $V(s)$ ) 转化为新策略  $\pi(s) = \arg \max_a Q(s, a)$ , 会面临问题。

因为  $Q(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V(s')]$ , 而计算  $Q(s, a)$  需依赖转移模型  $T(s, a, s')$ , 这打破了无模型的特性, 无法直接从  $V(s)$  无模型地生成新策略

**改进思路:** 直接学习动作值函数  $Q(s, a)$  (而非  $V(s)$ )。这样一来, 动作选择  $\pi(s) = \arg \max_a Q(s, a)$  也能实现无模型, 避免了借助  $T(s, a, s')$  从  $V(s)$  计算  $Q(s, a)$  的复杂过程, 使策略生成更直接且保持无模型特性

## Active Reinforcement Learning

策略评估, TD都是被动强化学习, 策略是固定的

**主动强化学习 (Active Reinforcement Learning)** 需要自主选择动作, 目标是学习到最优策略或值函数

在此过程中同样无模型, 既不知道状态转移函数  $T(s, a, s')$  (从状态  $s$  执行动作  $a$  转移到  $(s')$  的规律), 也不知道奖励函数  $R(s, a, s')$  (执行动作  $a$  从  $s$  转移到  $(s')$  获得的奖励)

这一过程中, 智能体面临“探索与利用”的根本权衡: “探索”指尝试新动作以获取更多环境信息; “利用”指选择当前已知最优动作以最大化奖励

这不是离线规划，而是需要智能体在实际环境中执行动作，观察结果并学习，属于在线学习、

### Q - 值迭代 (Q - Value Iteration)

值迭代中，目标是寻找连续（深度有限）的状态值函数  $V(s)$

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')]$$

而 Q - 值（动作值函数  $Q(s, a)$ ）直接关联动作选择，比状态值函数  $V(s)$  更有用

1. 初始化  $Q_0(s, a) = 0$
2. 给定  $Q_k$ ，对所有状态 - 动作对计算深度  $k + 1$  的 Q - 值：

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma \max_{a'} Q_k(s', a')]$$

即对每个状态 - 动作对  $(s, a)$ ，利用转移模型 T、奖励 R，结合后续状态  $(s')$  的最大 Q - 值 ( $\max_{a'} Q_k(s', a')$ ) 和折扣因子  $\gamma$  进行更新

### Q - Learning

TD Learning的具体方法，active

基于样本更新  $Q(s, a)$ ，而非依赖完整的模型（如转移概率 T）

1. 获取一个经验样本  $(s, a, s', r)$ ，即从状态  $s$  执行动作  $a$  转移到  $(s')$ ，并获得奖励  $r$
2. 提取当前对  $Q(s, a)$  的旧估计值
3. 计算新样本估计：

$$\text{sample} = R(s, a, s') + \gamma \max_{a'} Q(s', a')$$

其中  $R(s, a, s')$  是即时奖励， $\gamma$  是折扣因子， $\max_{a'} Q(s', a')$  是后继状态  $(s')$  下所有可能动作的最大 Q 值，体现对未来奖励的预期

4. 将新样本估计融入滑动平均

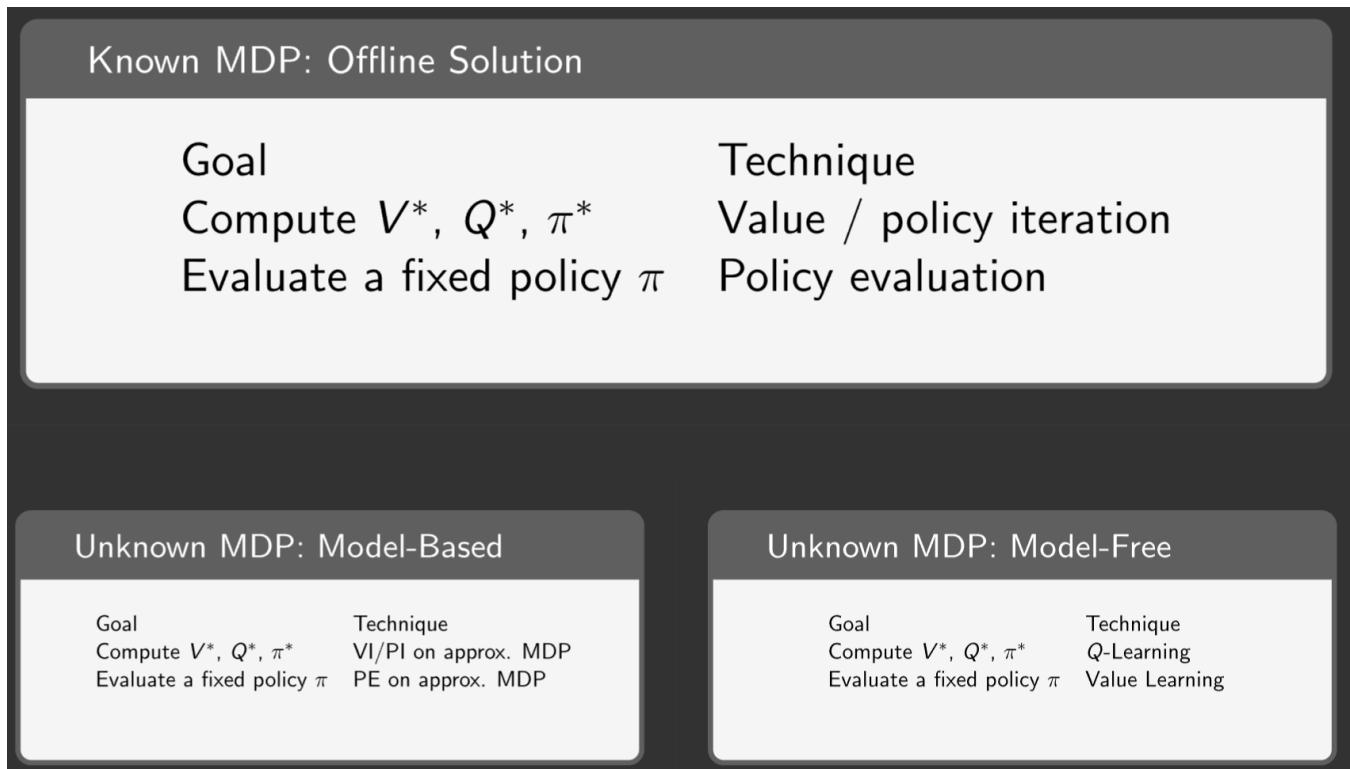
$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \cdot \text{sample}$$

通过这种方式， $Q(s, a)$  逐步逼近真实的动作值函数，支持智能体做出更优的动作决策

Q - Learning能够收敛到最优策略，即使智能体当前执行的策略并非最优

这种特性被称为**离线策略学习 (off - policy learning)**，意味着 Q - 学习不依赖于当前正在执行的策略进行学习，可从多种来源的经验中获取知识

## The Story So Far: MDPs vs RL



方案	已知 MDP：离线offline解决方案	未知 MDP：基于模型 model-based 方法	未知 MDP：无模型 model-free 方法
目标	最优值函数 $V^*$ , 最优动作值函数 $Q^*$ 、最优策略 $\pi^*$ ，并评估固定策略 $\pi$	同已知 MDP，计算 $V^*$ 、 $Q^*$ 、 $\pi^*$ 并评估固定策略 $\pi$	同样是计算 $V^*$ 、 $Q^*$ 、 $\pi^*$ 并评估固定策略 $\pi$
技术	通过值迭代 / 策略迭代 (Value /policy iteration) 求解最优解，利用策略评估 (Policy evaluation) 分析固定策略	先构建近似 MDP 模型，再在其上执行值迭代 / 策略迭代和策略评估。通过估计环境模型来间接求解	直接通过与环境交互学习，如 Q - 学习 (Q - Learning) 直接优化动作值函数，值学习 (Value Learning) 更新状态值函数，无需显式构建 MDP 模型

## Exploration vs Exploitation

探索策略

最简单的方法是采用随机动作( $\epsilon$  - greedy)。每一步通过“抛硬币”决定行动：

- 以较小概率  $\epsilon$  随机选择动作（探索新行为）；
- 以较大概率  $1 - \epsilon$  按当前策略选择动作（利用已有经验）

学习完成后，仍会因随机动作持续产生无意义的变动，无法稳定在最优策略

**优化思路：**探索“不好”(价值未明确) 的区域，而非盲目探索，最终在价值明确后停止探索

探索函数：输入值估计  $u$  和状态 - 动作对的访问次数  $n$ ，输出乐观效用

$$f(u, n) = u + \frac{k}{n+1}$$

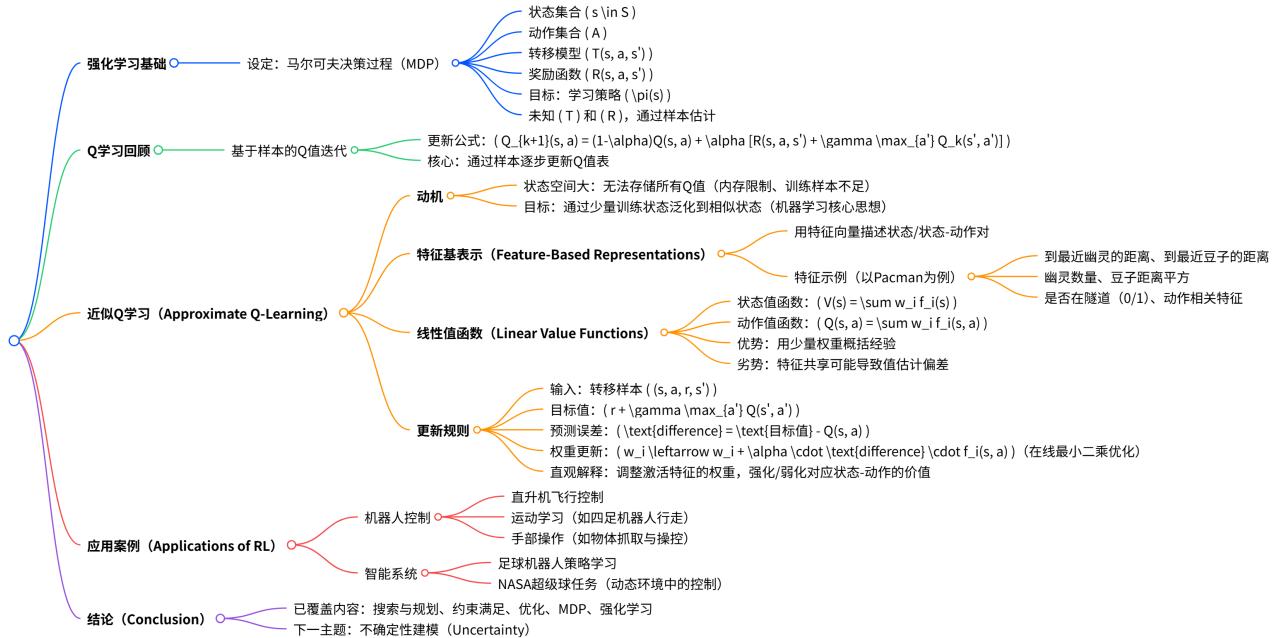
当访问次数  $n$  较少时， $\frac{k}{n+1}$  较大，激励智能体探索未充分访问的状态

使用该探索函数，可将原有Q-Update： $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[R(s, a, s') + \gamma \max_{a'} Q(s', a')]$

替换成  $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[R(s, a, s') + \gamma \max_{a'} f(Q(s', a'), N(s'))]$

将后继状态的 Q 值替换为探索函数 f 的输出（ $N(s')$  为状态  $(s')$  的访问次数），使“探索奖励（bonus）”能反向传播到导致未知状态的前置状态，引导智能体更高效地探索有价值的未知区域

# 17. Reinforcement Learning - III



 豆包  
你的 AI 助手，助力每日工作学习

强化学习：无T和R，使用样本计算关于转移模型T的所有平均值来寻找最佳策略

## Approximate Q-Learning 重要

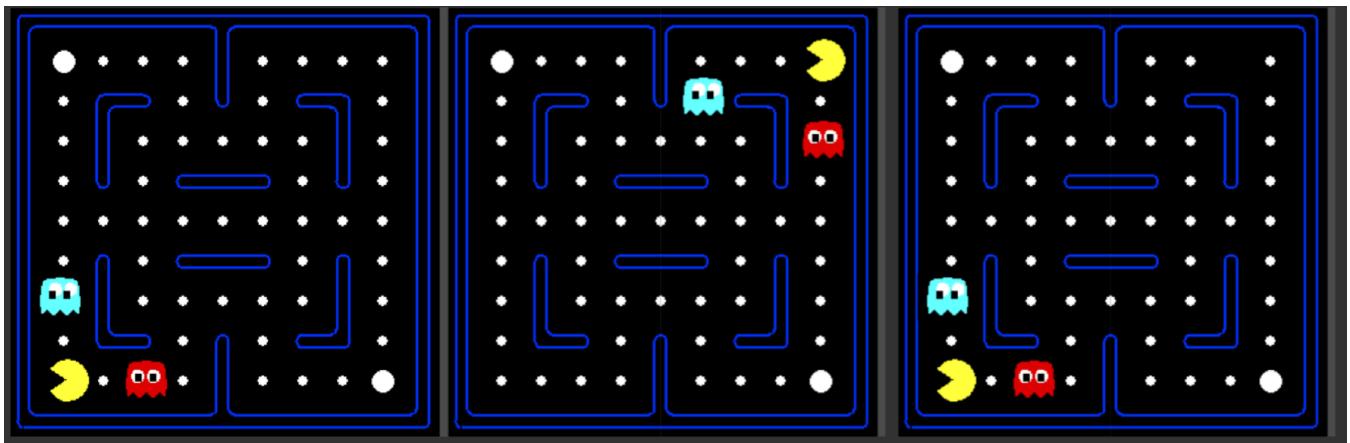
基础的 Q 学习会维护一个包含所有 Q 值的表格，但在实际场景中，完全学习每一个单独的状态是不可行的，原因有二：

- 训练时状态数量过多，无法全部遍历
- 状态数量过多，内存无法存储所有 Q 值表

因此考虑“泛化”(generalize)：

1. 从经验中学习少量的训练状态 (**Learn about some small number of training states from experience**)
2. 将这些经验推广到新的、类似的情况中 (**Generalize that experience to new, similar situations**)

这是机器学习的一个基本思想，会在许多场景中反复出现



如吃豆人例子中，如果状态太过庞大，未经历过的状态（如中间和右侧画面展示的状态）就难以学习其价值，体现了基础 Q 学习在状态过多时无法泛化的问题

### 基于特征的表示 (FEATURE - BASED REPRESENTATIONS)

**解决方案：**使用**特征（属性）向量**描述状态。特征是从状态映射到实数（常为 (0/1)）的函数，用于捕捉状态的重要属性

#### 特征示例

- 到最近幽灵的距离 (Distance to closest ghost)；
- 到最近豆子的距离 (Distance to closest dot)；
- 幽灵数量 (Number of ghosts)；
- $\frac{1}{(\text{dist to dots})^2}$  (到豆子距离的平方倒数)；
- 吃豆人是否在隧道 (0/1, Is Pacman in a tunnel?)；
- 等等

特征不仅能描述状态，还能用特征描述 Q 状态  $(s, a)$ ，例如“动作是否使吃豆人更接近食物”这样的特征

这种表示方法通过提取状态的关键特征，为解决状态空间过大时的泛化问题提供了有效途径，避免了对每个具体状态单独存储和学习的局限性

### 线性值函数 (LINEAR VALUE FUNCTIONS)

利用特征表示，通过少量权重描述状态值函数  $V(s)$  或动作值函数  $Q(s, a)$ 。

- 状态值函数： $V(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$ 。
- 动作值函数： $Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$

**优势：**经验可以通过少量权重进行概括，避免了对每个状态单独存储大量值，提高了效率

**劣势：**不同状态可能共享某些特征，但实际价值差异较大。例如，两个状态有相似的“到幽灵距离”特征，但一个可能因周围豆子多而价值高，另一个因幽灵即将攻击而价值低，线性模型可能因特征共享无法准确区分这种差异，导致值估计偏差

### 近似 Q 学习 (Approximate Q - Learning) 重要

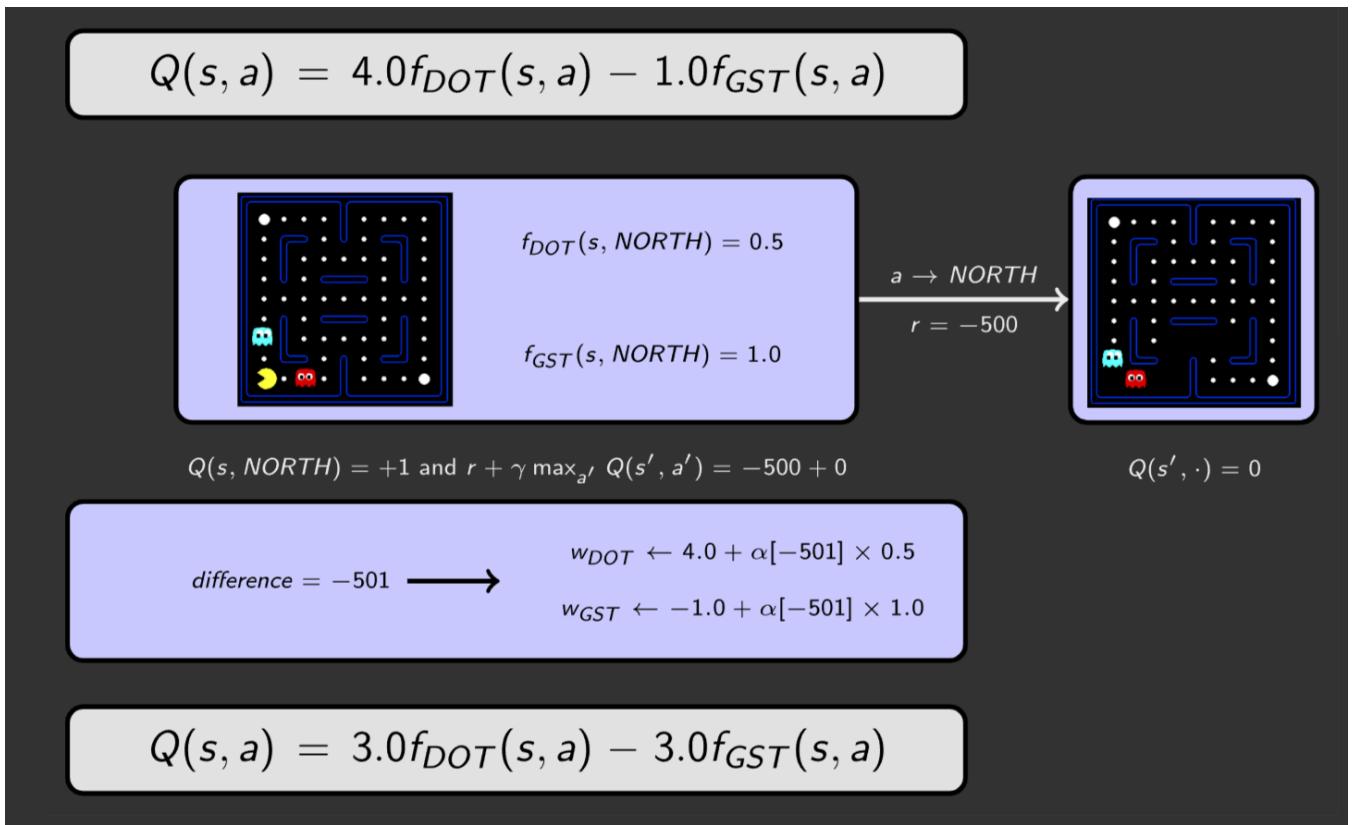
**线性 Q 函数定义:**  $Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$ , 通过特征  $f_i(s, a)$  与权重  $w_i$  的线性组合近似表示动作值函数

**线性版本 Q 学习更新步骤:**

1. 给定转移样本  $\text{transition} = (s, a, r, s')$  (在状态  $s$  执行动作  $a$ , 获得奖励  $r$ , 转移到新状态  $s'$ )。
2. 计算目标值与当前 Q 值的差异:  $\text{difference} = [r + \gamma \max_{a'} Q(s', a')] - Q(s, a)$ , 其中  $\gamma$  是折扣因子, 体现未来奖励的重要性。
3. 更新权重  $w_i$ :  $w_i \leftarrow w_i + \alpha[\text{difference}]f_i(s, a)$ , 通过学习率  $\alpha$  调整与当前状态 - 动作对激活特征  $f_i(s, a)$  对应的权重, 强化或弱化该特征对 Q 值的影响。

可动态调整激活特征的权重。例如, 若发生意外的负面结果, 就“责怪”当前激活的特征, 降低对具有这些特征的状态 - 动作对的偏好, 反之亦然

这种更新方式基于**在线最小二乘法**, 通过最小化预测误差 (目标值与当前估计值的差) 来优化权重

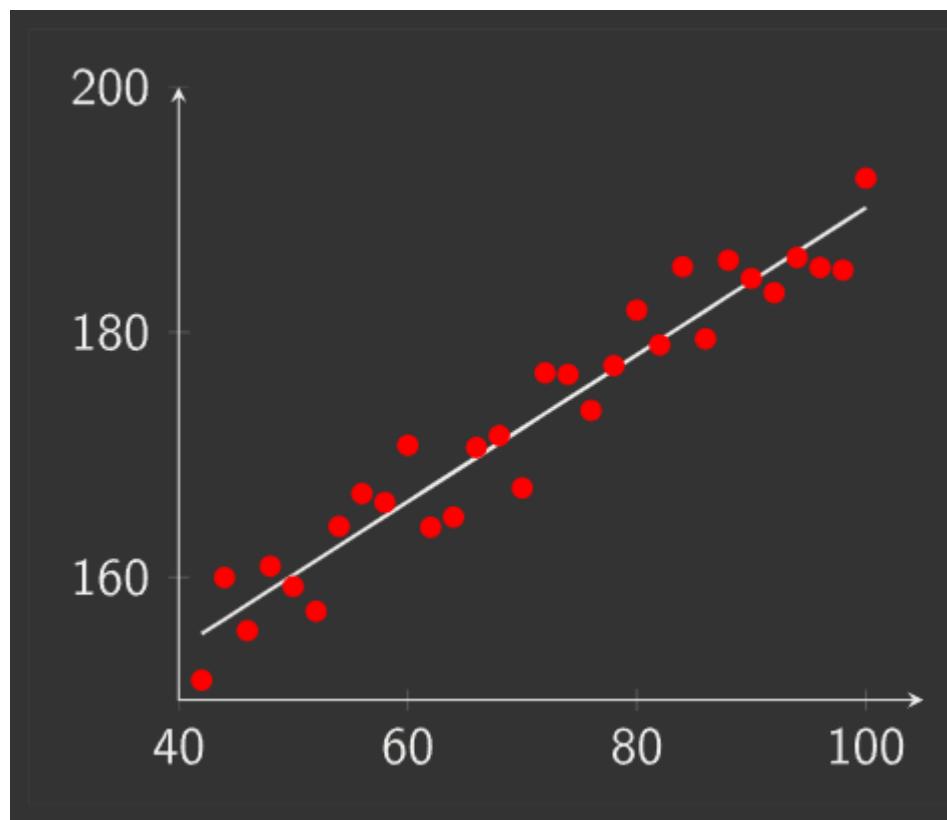


## Q-Learning and Least Squares

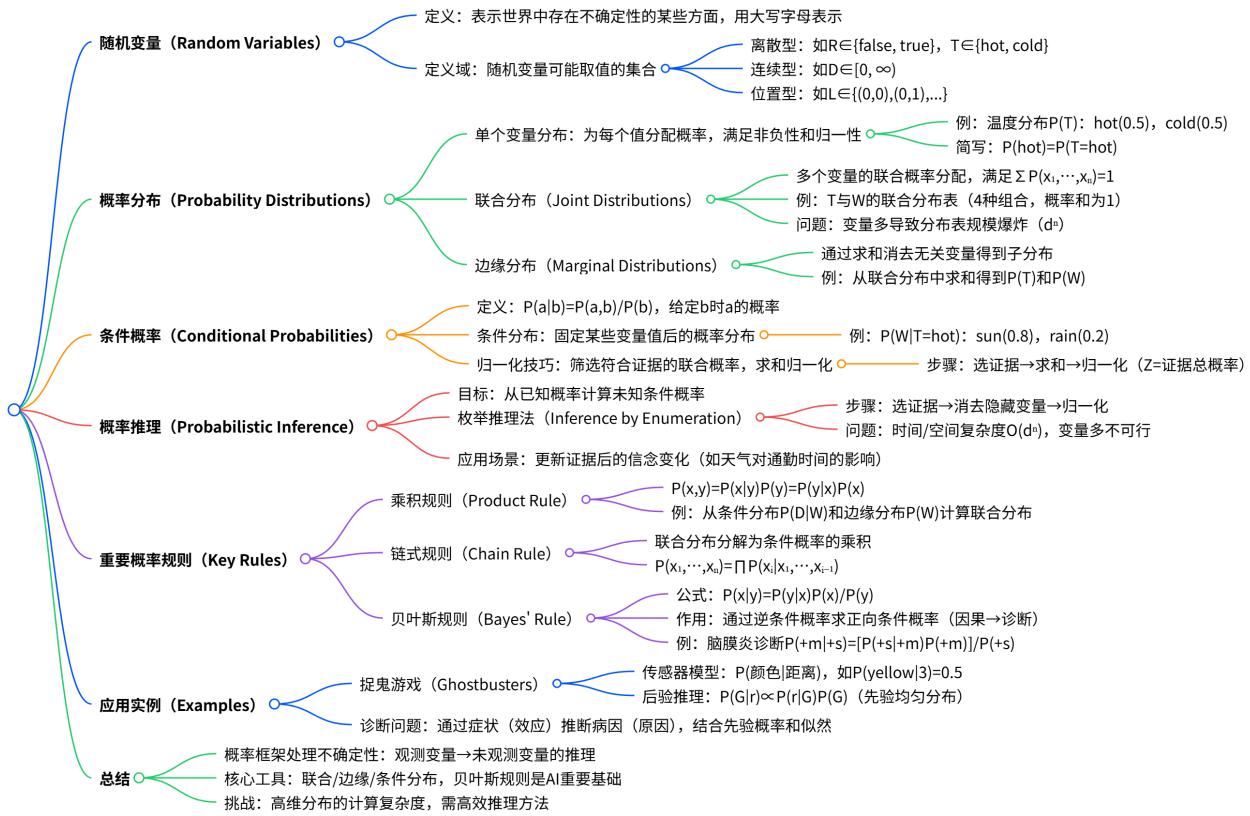
**最小二乘法 (Least Squares)** 中, 总误差 (total error) 定义为真实值  $y_i$  与预测值  $\hat{y}_i$  之差的平方和, 即:

$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left( y_i - \sum_k w_k f_k(x_i) \right)^2$$

其中  $w_k$  是权重,  $f_k(x_i)$  是特征函数。通过调整  $w_k$ , 最小化该误差, 使预测值  $\hat{y}_i$  尽可能接近真实值  $y_i$



# 18. Probability



豆包

你的 AI 助手, 助力每日工作学习

概率推理可用于不确定环境中

概率推理中主要包含:

- 观察到的变量 (证据):** 智能体已知关于世界状态的某些信息 (如传感器读数或症状), 这些信息作为推理的证据。
- 未观察到的变量:** 智能体需要推理的其他方面 (如物体的位置或存在的疾病), 这些是未知且需推断的内容。
- 模型:** 智能体知晓已知变量与未知变量之间的关联方式。

概率推理为管理知识提供了一个框架, 用于在不确定情况下进行推理和判断

## Random Variables

我们通常使用大写字母来表示随机变量, 用小数表示该变量取值发生的概率

Weather $P(W)$	
$W$	$P$
sun	0.6
rain	0.1
fog	0.3
meteor	0.0

1. **非负性**: 对于任意取值  $x$ , 有  $\forall x, P(X = x) \geq 0$ 。
2. **归一性**: 所有取值的概率之和为 1, 即  $\sum_x P(X = x) = 1$ 。

## Joint Distributions

联合分布是针对一组随机变量  $X_1, X_2, \dots, X_n$  的分布, 为每个变量取值组合 (或结果) 指定一个实数, 记作  $P(X_1, \dots, X_n)$  或  $P(X_1 = x_1, \dots, X_n = x_n)$

$T$	$W$	$P$
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

$$P(T = \text{hot}, W = \text{sun}) = 0.4$$

仍满足非负性和归一性

**规模问题：**若有  $n$  个变量，每个变量的定义域大小为  $d$ ，则联合分布包含  $d^n$  种取值组合

除了极简单的情况，完整写出联合分布表往往不切实际（如  $n$  较大时，组合数呈指数级增长）

## Probabilistic Models

---

**定义：**概率模型是一组随机变量的联合分布。

核心特征：

1. **带定义域的（随机）变量：**如  $T$ （定义域为  $\{\text{hot}, \text{cold}\}$ ）和  $W$ （定义域为  $\{\text{sun}, \text{rain}\}$ ）。
2. **结果（Outcomes）：**变量的赋值组合称为结果，例如  $(T = \text{hot}, W = \text{sun})$ 。
3. **联合分布：**描述结果发生的可能性，如右侧第一个表格中  $(P(T = \text{hot}, W = \text{sun}) = 0.4)$ 。
4. **归一化：**所有结果的概率之和为  $(1.0)$ ，如  $(0.4 + 0.1 + 0.2 + 0.3 = 1)$ 。
5. **变量交互：**理想情况下，仅特定变量直接交互。

$T$	$W$	$P$
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

约束满足问题（CSP）

核心特征

1. **带定义域的变量：**与概率模型类似。
2. **约束：**说明赋值是否可能（而非可能性大小）。
3. **变量交互：**理想情况下，仅特定变量直接交互。

$T$	$W$	$P$
hot	sun	T
hot	rain	F
cold	sun	F
cold	rain	T

二者都涉及带定义域的变量和变量间的交互，但概率模型通过联合分布量化结果的可能性，CSP 通过约束判断结果是否可行

## Marginal Distributions

定义：边缘分布是通过消除联合分布中的某些变量得到的子分布，其核心操作是 **边缘化**（marginalization，也称求和消去 summing out），即将被消除变量的所有取值情况对应的概率相加

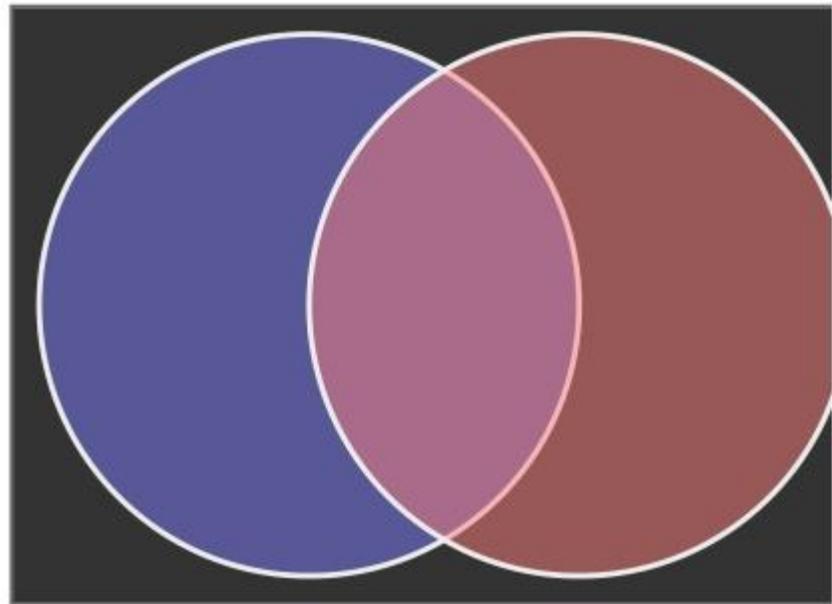
$T$	$W$	$P$	$T$	$P$
hot	sun	0.4	hot	0.5
hot	rain	0.1	cold	0.5
cold	sun	0.2	$W$	$P$
cold	rain	0.3	sun	0.6
			rain	0.4

例如： $P(T = \text{hot}) = P(T = \text{hot}, W = \text{sun}) + P(T = \text{hot}, W = \text{rain}) = 0.4 + 0.1 = 0.5$

## Conditional Distributions

表示在事件 b 发生的条件下，事件 a 发生的概率，等于 a 和 b 同时发生的联合概率  $P(a, b)$  除以 b 发生的概率  $P(b)$

$$P(a|b) = \frac{P(a,b)}{P(b)}$$



$T$	$W$	$P$
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

$$P(W = s|T = c) = \frac{P(W = s, T = c)}{P(T = c)} = \frac{P(W = s, T = c)}{P(W = s, T = c) + P(W = r, T = c)} = \frac{0.2}{0.2 + 0.3} = 0.4$$

在给定变量固定值时，计算条件概率很方便

$$P(W|T = \text{hot})$$

$W$	$P$
sun	0.8
rain	0.2

$$P(W|T = \text{cold})$$

$W$	$P$
sun	0.4
rain	0.6

$T$	$W$	$P$
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

其中在  $T = \text{hot}$  时,  $T = \text{hot}$  的总概率为  $0.4 + 0.1 = 0.5$

$$P(W = \text{sun}|T = \text{hot}) = \frac{0.4}{0.5} = 0.8$$

$$P(W = \text{rain}|T = \text{hot}) = \frac{0.1}{0.5} = 0.2$$

在以上计算中, 涉及到归一化过程, 也就是  $\div 0.5$  的过程

1. 计算选中部分的概率总和  $Z$  (即  $P(\text{evidence})$ , 此处为  $P(T = \text{cold}) = 0.2 + 0.3 = 0.5$ )

2. 将选中的每个概率除以  $Z$ , 使其总和为 1。例如,  $P(W = \text{sun}|T = \text{cold}) = \frac{0.2}{0.5} = 0.4$ ,

$$P(W = \text{rain}|T = \text{cold}) = \frac{0.3}{0.5} = 0.6$$

$W$	$P$	$\xrightarrow[\mathcal{Z} = 0.5]{\text{NORMALIZE}}$	$W$	$P$	$\xrightarrow[\mathcal{Z} = 50]{\text{NORMALIZE}}$	$X$	$Y$	$P$
sun	0.2		sun	0.4		hot	sun	0.4
rain	0.3		rain	0.6		hot	rain	0.1

## Probabilistic Inference

概率推理是从其他已知概率 (如联合概率) 中计算所需概率 (如条件概率) 的过程

通常计算条件概率, 例如  $P(\text{on time}|\text{no reported accidents}) = 0.90$ , 表示在 “无事故报告” 这一证据下, “准时”的概率为 0.90

概率会随新证据的出现而变化:  $P(\text{on time} | \text{no accidents}, 5 \text{ a.m.}) = 0.95$

- **证据变量 (Evidence variables):**  $E_1, \dots, E_k$ , 取值为  $e_1, \dots, e_k$ 。
- **查询变量 (Query variables):**  $Q$ , 即我们关心其概率的变量。
- **隐藏变量 (Hidden variables):**  $H_1, \dots, H_r$ , 这些变量未被观察, 但影响推理结果。
- **目标:** 计算条件概率  $P(Q | e_1, \dots, e_k)$

## 步骤

1. **选择与证据一致的条目:** 从联合分布中筛选出所有与证据  $e_1, \dots, e_k$  匹配的项。

2. **对隐藏变量求和:** 计算查询变量  $Q$  与证据  $e_1, \dots, e_k$  的联合概率, 公式为:

$$P(Q, e_1, \dots, e_k) = \sum_{h_1, \dots, h_r} P(Q, e_1, \dots, e_k, h_1, \dots, h_r) \text{ 即对隐藏变量 } H_1, \dots, H_r \text{ 的所有可能取值求和。}$$

3. **归一化**

- 计算归一化常数  $Z = P(e_1, \dots, e_k) = \sum_Q P(Q, e_1, \dots, e_k)$ 。
- 最终条件概率为:  $P(Q | e_1, \dots, e_k) = \frac{1}{Z} P(Q, e_1, \dots, e_k)$

summer	hot	sun	0.30
summer	hot	rain	0.05
summer	cold	sun	0.10
summer	cold	rain	0.05
winter	hot	sun	0.10
winter	hot	rain	0.05
winter	cold	sun	0.15
winter	cold	rain	0.20

例如用该表格求  $P(W|winter, hot)$

以上计算最坏时间复杂度为  $\mathcal{O}(d^n)$ , 存储联合分布的空间复杂度为  $\mathcal{O}(d^n)$

## Product Rule

用于从条件分布推导联合分布, 公式为  $P(y)P(x|y) = P(x, y)$ , 也可变形为条件概率公式  $P(x|y) = \frac{P(x,y)}{P(y)}$

如图中即为使用  $P(y)P(x|y) = P(x, y)$

$W$	$P$
sun	0.8
rain	0.2

$D$	$W$	$P$
wet	sun	0.1
dry	sun	0.9
wet	rain	0.7
dry	rain	0.3

$D$	$W$	$P$
wet	sun	0.18
dry	sun	0.72
wet	rain	0.14
dry	rain	0.06

以上规则可扩展得到链规则（Chain Rule），任何联合分布都可表示为条件分布的递增乘积

$$P(x_1, x_2, \dots, x_n) = \prod_i P(x_i | x_1, \dots, x_{i-1})$$

## Bayes Rule

由  $P(x, y) = P(x|y)P(y) = P(y|x)P(x)$  可推导得出贝叶斯公式：

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

该式子允许从反向条件概率  $P(y|x)$  推导出正向条件概率  $P(x|y)$

例如，在医学诊断中，已知疾病  $x$  导致症状  $y$  的概率  $P(y|x)$ ，可通过贝叶斯规则计算出现症状  $y$  时患疾病  $x$  的概率  $P(x|y)$

已知：

脑膜炎的先验概率： $P(+m) = 0.0001$

患脑膜炎时脖子僵硬的概率： $P(+s|m) = 0.8$

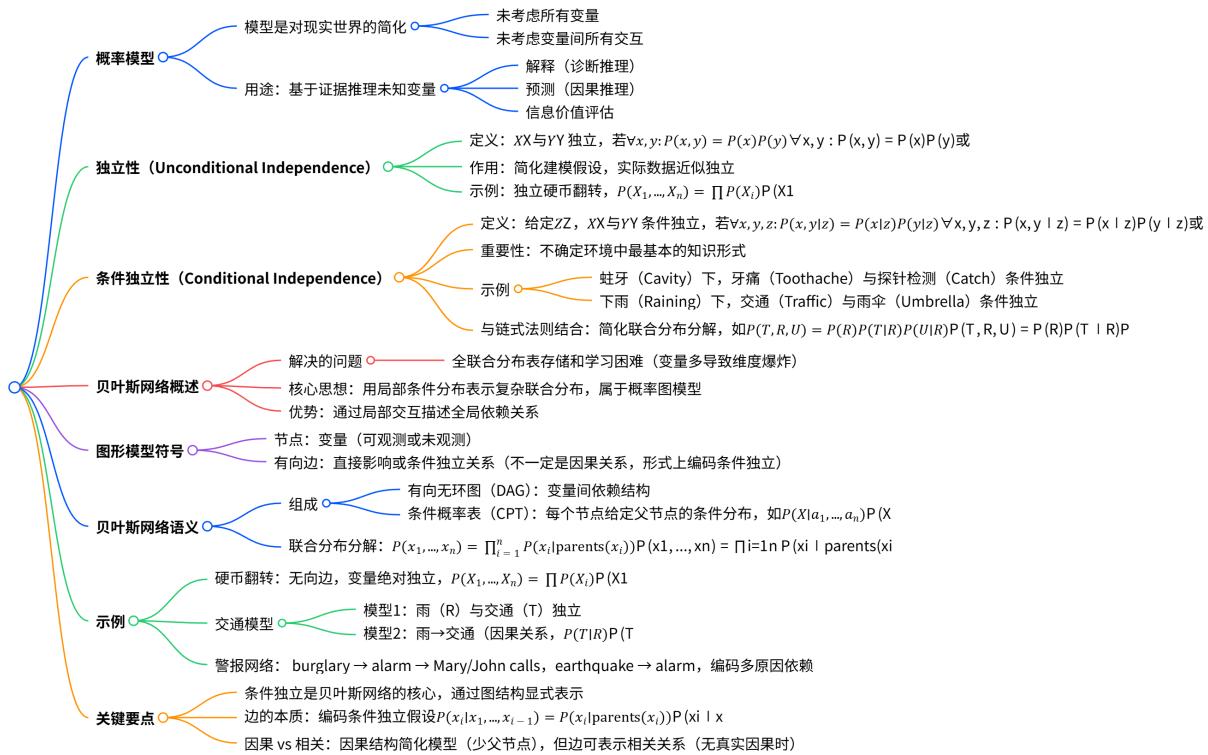
不患脑膜炎时脖子僵硬的概率： $P(+s|-m) = 0.01$

那么在出现脖子僵硬情况下，得脑膜炎的概率：

$$\begin{aligned} P(+m|+s) &= \frac{P(+s|m)P(+m)}{P(+s)} \\ &= \frac{P(+s|m)P(+m)}{P(+s|m)P(+m) + P(+s|-m)P(-m)} \\ &= \frac{0.8 \times 0.0001}{0.8 \times 0.0001 + 0.01 \times 0.9999} \end{aligned}$$

计算表明，即使出现脖子僵硬 ( $+s$ )，患脑膜炎 ( $+m$ ) 的后验概率仍然很小

# 19. Bayesian Networks: Representation



豆包

你的 AI 助手, 助力每日工作学习

## Independence

两个变量独立, 当且仅当对所有取值  $x, y$ , 满足  $P(x, y) = P(x)P(y)$

等价形式: 对所有  $x, y$ ,  $P(x|y) = P(x)$ , 即给定  $y$  时  $x$  的条件概率不受  $y$  影响, 通常记为  $X \perp Y$

T	P	W	P
hot	0.5	sun	0.6
cold	0.5	rain	0.4

T	W	P
hot	sun	0.3
hot	rain	0.2
cold	sun	0.3
cold	rain	0.2

### 条件独立性 (Conditional Independence)

无条件（绝对）独立 (Unconditional independence) 在现实中非常罕见，因为变量间通常存在各种关联，完全不依赖任何条件的独立情况极少

条件独立性是我们理解不确定环境时最基本且可靠的知识形式

定义：

- 给定  $Z$  时， $X$  与  $Y$  条件独立 (记作  $X \perp Y | Z$ )，当且仅当对所有  $(x, y, z)$ ，满足  $P(x, y | z) = P(x | z)P(y | z)$
- 另一种等价表述为：对所有  $(x, y, z)$ ， $P(x | y, z) = P(x | z)$ ，即给定  $Y$  和  $Z$  时  $X$  的条件概率，与仅给定  $Z$  时  $X$  的条件概率相等

例如：

- 若存在蛀牙 (+cavity)，探针检测到 (+catch) 的概率与是否牙痛 (+toothache) 无关，即  $P(+catch | +toothache, +cavity) = P(+catch | +cavity)$

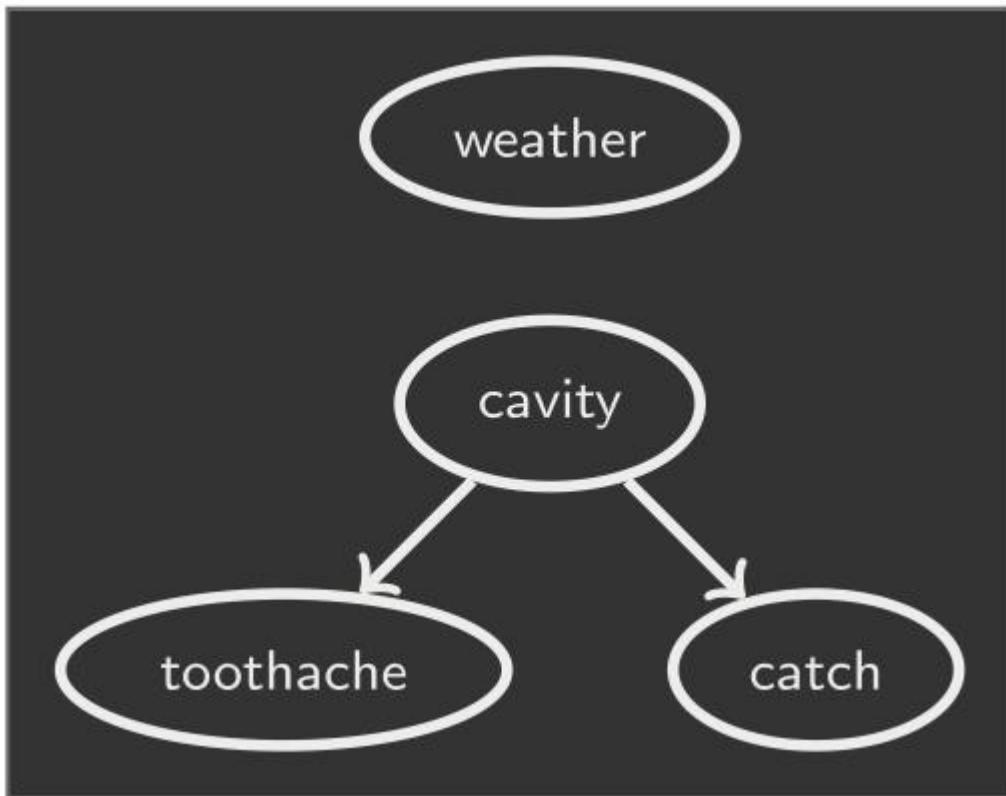
应用到链式法则Chain Rule中， $U \perp T | R$ ，那么  $P(T, R, U) = P(R)P(T | R)P(U | R, T)$

可以简化成  $P(T, R, U) = P(R)P(T | R)P(U | R)$

## Bayes Net: Big Picture

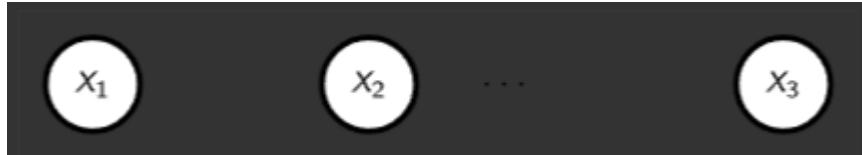
贝叶斯网络是一种通过简单的局部分布（条件概率）来描述复杂联合分布表（模型）的技术，更准确的名称是 概率图模型

其核心在于描述变量间的 局部交互，这些局部交互相互连接，最终形成全局的、间接的交互关系，从而简化对复杂联合分布的建模与分析



- **节点 (Nodes)**: 代表变量 (具有定义域)，变量可以是已赋值 (观察到的) 或未赋值 (未观察到的)
- **弧 (Arcs)**: 表示变量间的交互，类似于约束满足问题 (CSP) 中的约束，指示变量间的“直接影响”

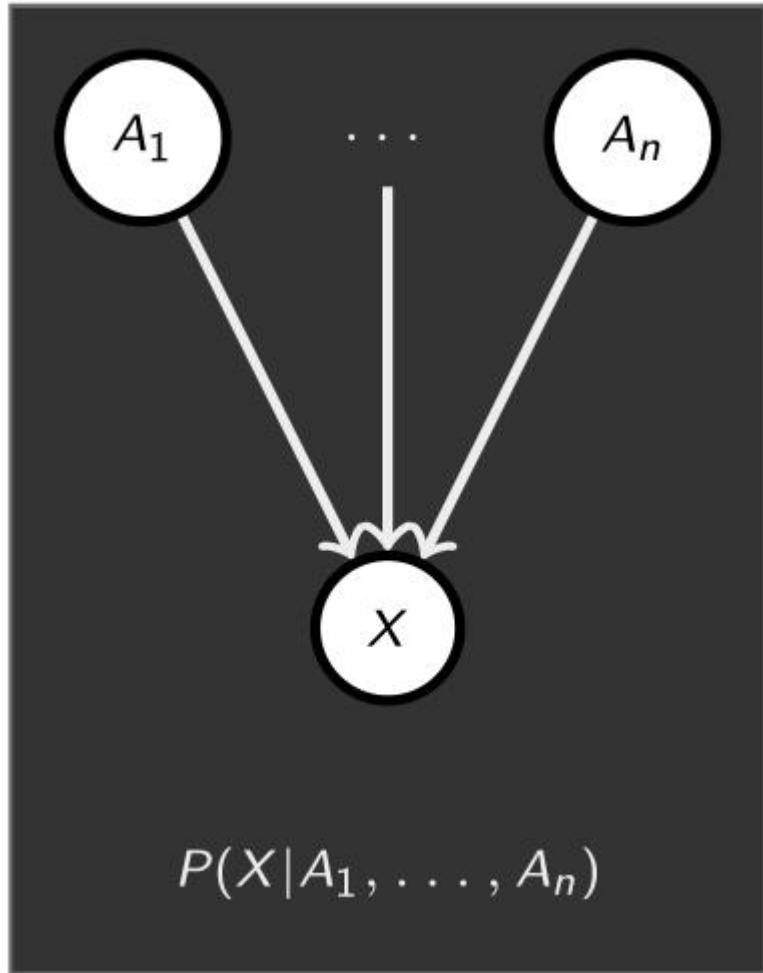
如图中cavity对toothache和catch有影响



而该图中各变量则互相绝对独立absolute independence

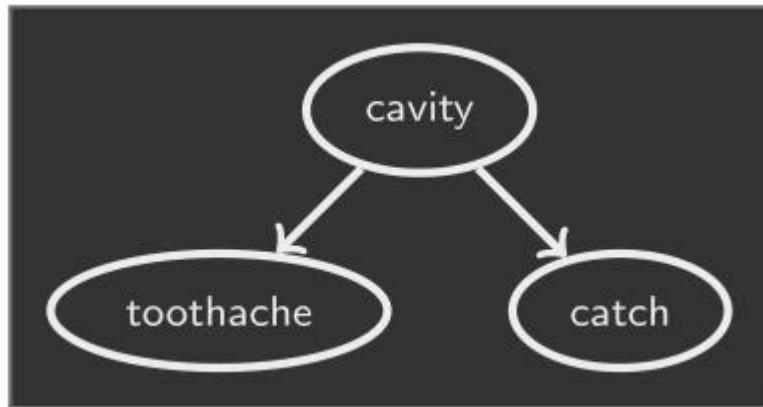
## Bayes Net Semantics

贝叶斯网络的构成：由两部分组成，一是拓扑结构（即有向无环图），二是局部条件概率（每个节点的条件分布）



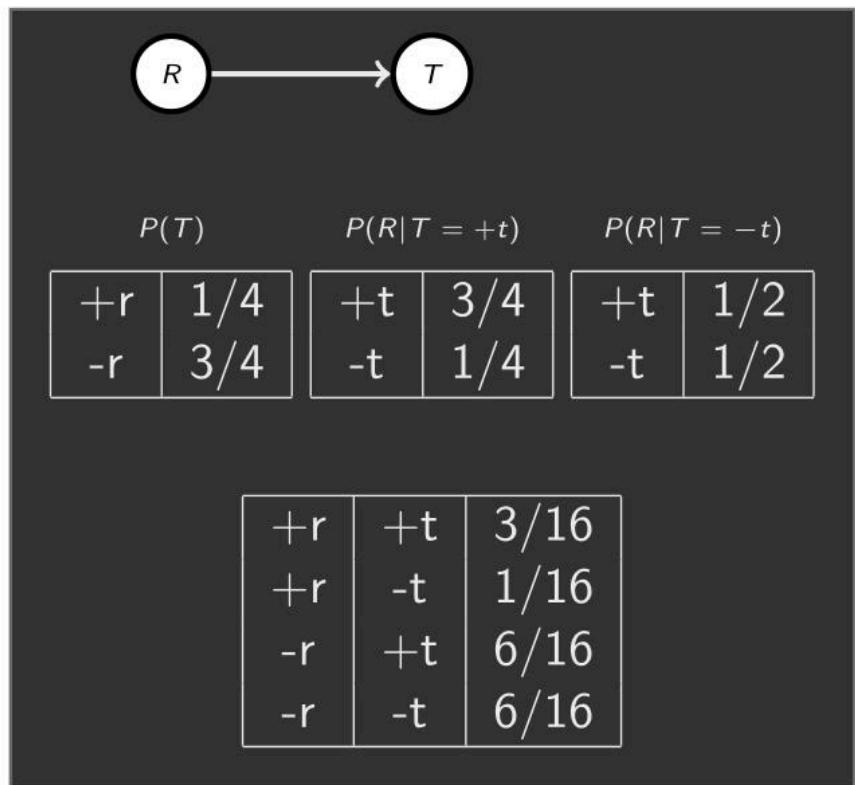
贝叶斯网络通过局部条件分布的乘积隐式编码联合概率分布

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{parents}(x_i))$$



如图，有

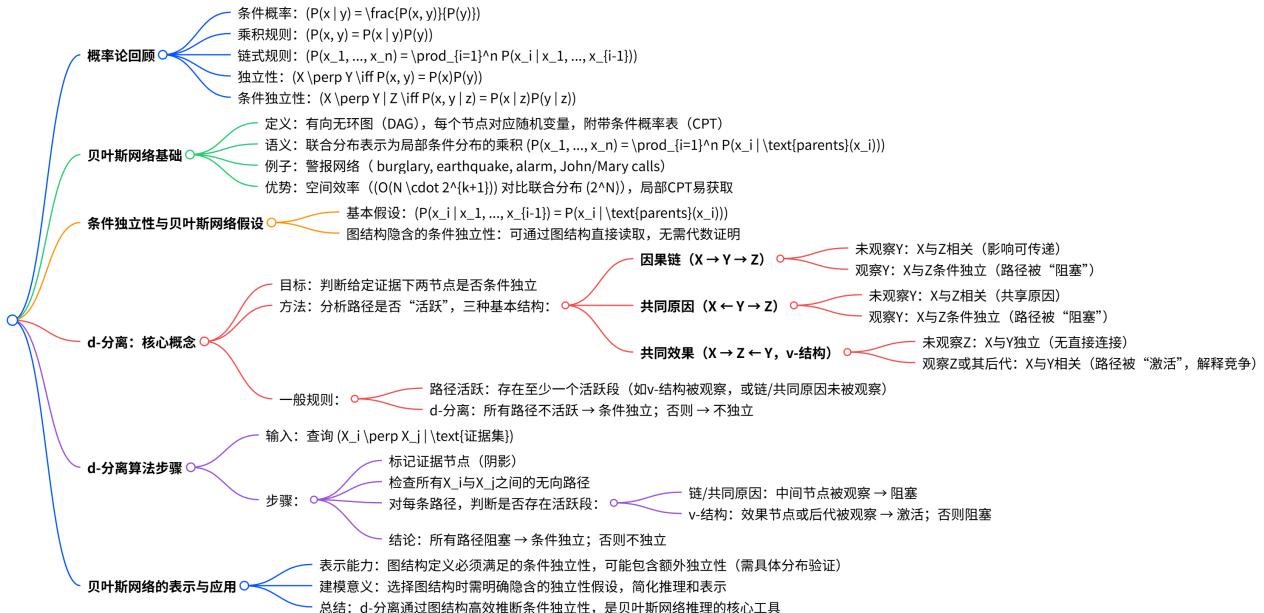
$$P(+cavity, +catch, -toothache) = P(+cavity) \times P(-toothache | +cavity) \times P(+catch | +cavity)$$



$$P(R, T) = P(R) \times P(T|R)$$

其实，贝叶斯网络拓扑结构的**本质是编码条件独立性 encodes conditional independence**，而非必然反映因果关系 **causal structure**

# 20. Bayesian Networks: Independence



豆包  
你的 AI 助手，助力每日工作学习

**联合分布的规模：**对于  $N$  个布尔变量的联合分布，其规模为  $2^N$

**贝叶斯网络的规模：**若一个  $N$  节点的贝叶斯网络中，每个节点最多有  $k$  个父节点，其规模为  $O(N * 2^{k+1})$

相比联合分布  $2^N$  的指数级增长，贝叶斯网络能大幅节省空间

## Independence Conditions

回顾：

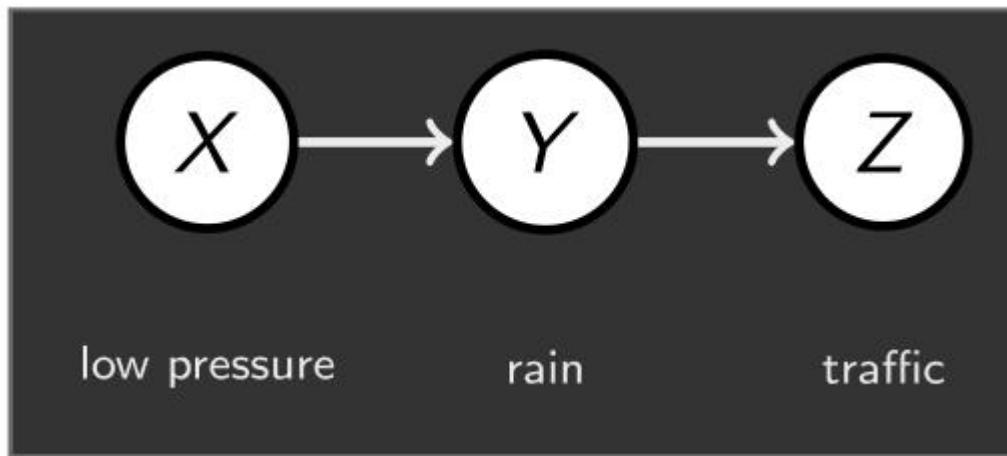
- 独立：若对于所有  $x, y$ ，有  $P(x, y) = P(x)P(y)$ ，则称  $X$  和  $Y$  独立，记作  $X \perp Y$
- 条件独立：若对于所有  $x, y, z$ ，有  $P(x, y|z) = P(x|z)P(y|z)$ ，则称  $X$  和  $Y$  在给定  $Z$  时条件独立，记作  $X \perp Y|Z$

(条件) 独立性是概率分布的一种属性

贝叶斯网络通常包含额外的条件独立性，这些独立性可直接从图结构中读取，无需复杂计算

## D-Separation

### Causal Chains

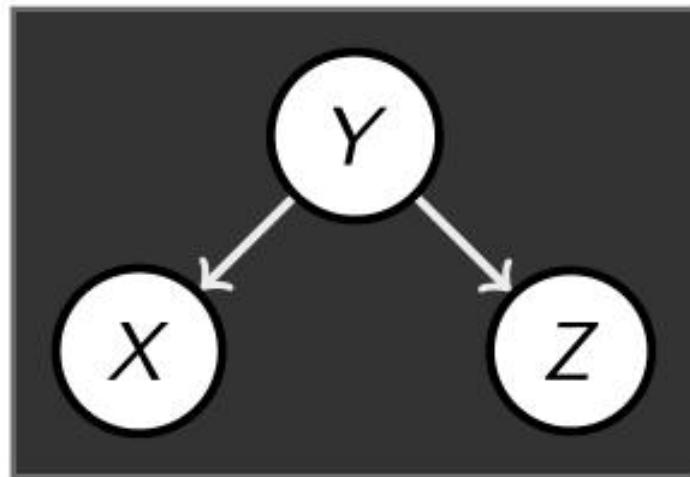


以上结构中，X和Z不一定独立，只能说给定Y的前提下X和Z两者条件独立

$$P(z|x,y) = \frac{P(x,y,z)}{P(x,y)} = \frac{P(x)P(y|x)P(z|y)}{P(x)P(y|x)} = P(z|y)$$

链上的证据（如Y）“阻断 (blocks)”了X对Z的影响

### Common Cause



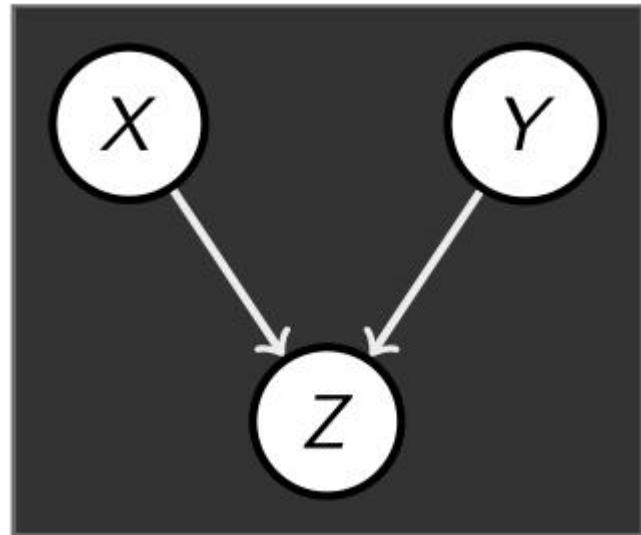
共同原因，如上图，项目截止导致论坛繁忙和实验室满员

$$P(z|x,y) = \frac{P(x,y,z)}{P(x,y)} = \frac{P(y)P(x|y)P(z|y)}{P(y)P(x|y)} = P(z|y)$$

推导表明，给定Y时，Z的概率仅取决于Y，与X无关，即X和Z在给定Y时条件独立

共同原因Y会阻断X与Z间的影响

### Common Effect



共同效果，如上图球赛和下雨本身无直接关联，却都能导致交通问题

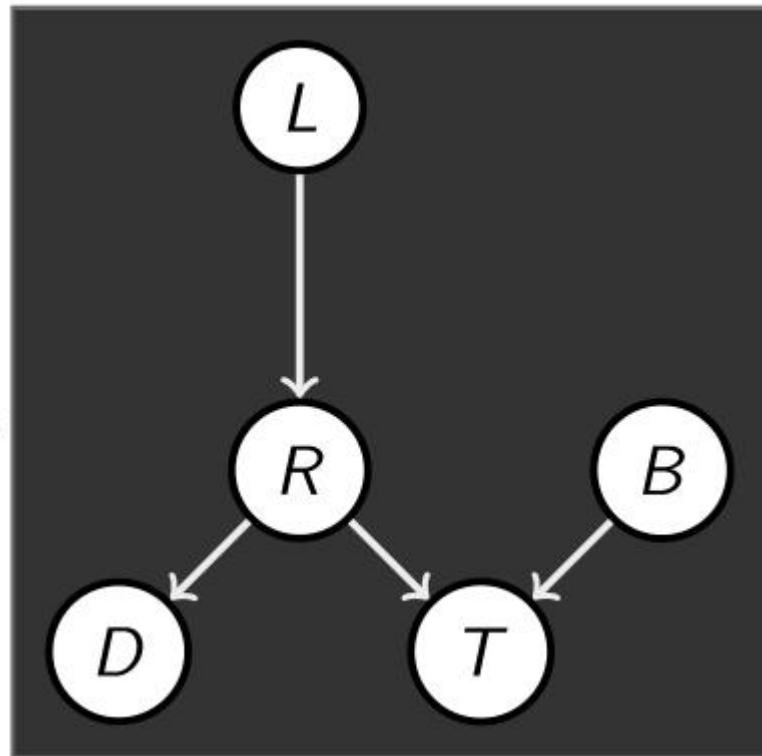
X 和 Y 原本独立，但给定结果 Z 反而时变得不独立

## The General Case

任何复杂的贝叶斯网络实例，均可拆解为三种典型结构（因果链、共同原因、共同效果）的重复组合。利用这三种结构的条件独立性特性可系统分析复杂网络中变量的独立性，将复杂问题简化为对基本结构的逐一判断

在分析时两个节点由一条未阻断的路径连接，则认为可达

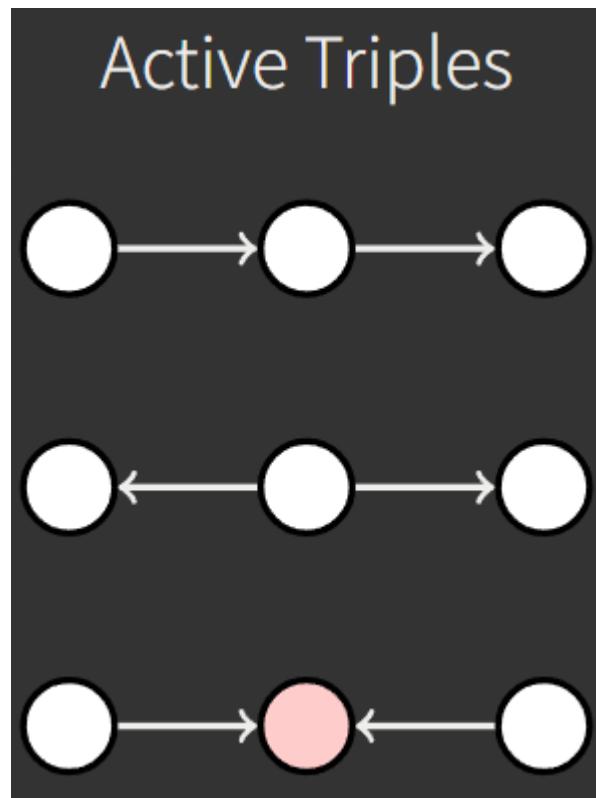
然而在Common Effect结构中，必须被阻断（涂色）才被认为可达，例如T不被阻断则认为L不可达B



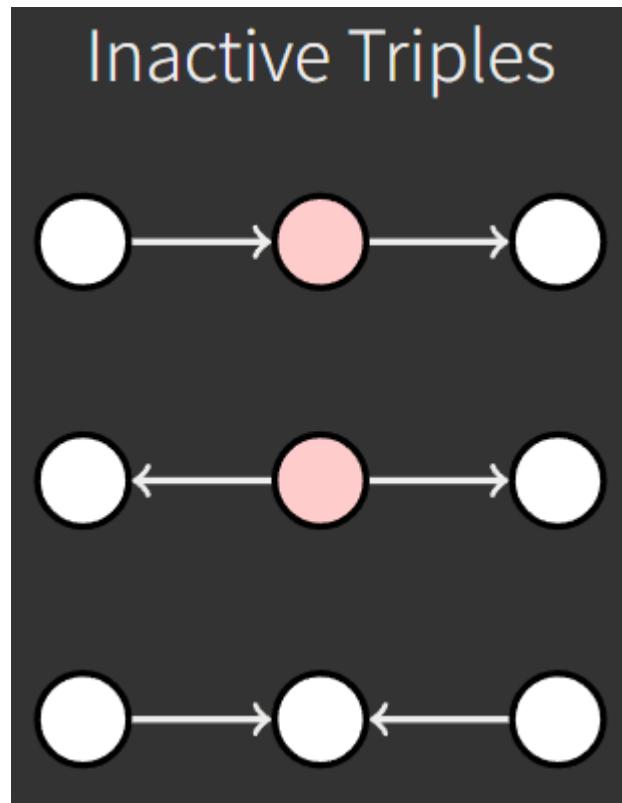
判断在给定证据变量 Z 时，变量 X 和 Y 是否条件独立

方法：分析所有从 X 到 Y 的（无向）路径，若无活跃路径，则 X 和 Y 条件独立，即  $X \perp\!\!\!\perp Y | \{X_{k_1}, \dots, X_{k_n}\}$

活跃路径：



不活跃路径：



#### d - 分离 (D - SEPARATION)

判断  $X_i$  与  $X_j$  在给定证据集  $\{X_{k_1}, \dots, X_{k_n}\}$  时是否条件独立，即  $X_i \perp\!\!\!\perp X_j | \{X_{k_1}, \dots, X_{k_n}\}$

判断方法：检查  $X_i$  与  $X_j$  之间的所有（无向）路径

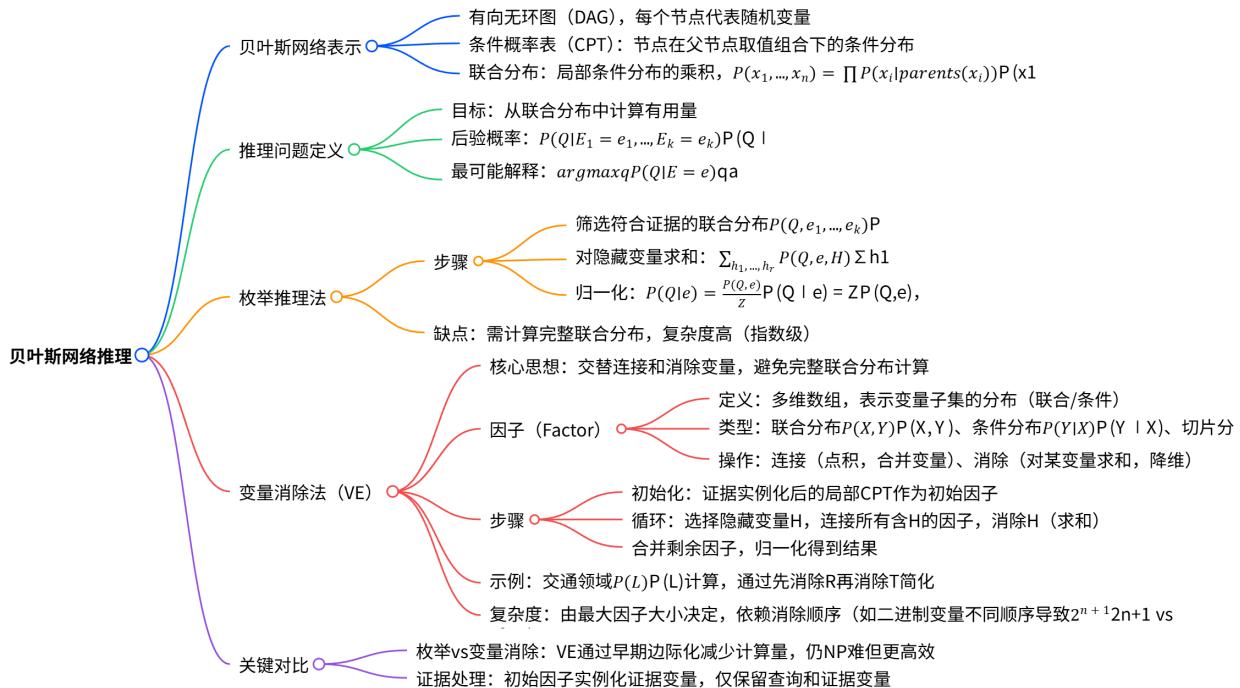
- 若存在**一条或多条活跃路径**，则  $X_i$  与  $X_j$  的独立性无法保证，记作  $X_i \not\perp X_j | \{X_{k_1}, \dots, X_{k_n}\}$
- 若**所有路径均不活跃**，则  $X_i$  与  $X_j$  条件独立，记作  $X_i \perp X_j | \{X_{k_1}, \dots, X_{k_n}\}$

## Summary

---

- **紧凑编码联合分布**：贝叶斯网络通过有向无环图和局部条件概率表，将高维联合分布紧凑地编码为局部条件概率的乘积，避免了联合分布的指数级存储开销
- **图结构推导独立性**：从贝叶斯网络的图结构可直接推导出分布中**必然存在的（条件）独立性**，这些独立性是图结构隐含的，无需依赖具体概率数值
- **d - 分离的作用**：d-分离算法仅通过分析图结构，就能精确判定节点间的条件独立性，无需复杂计算，为条件独立性提供了明确的图结构判断依据
- **额外独立性的存在**：贝叶斯网络的联合分布可能存在图结构未体现的**更多（条件）独立性**，这些独立性需通过分析具体的概率分布才能被发现，图结构仅保证了部分明确的独立关系

# 21. Bayesian Networks: Inference



豆包

你的 AI 助手, 助力每日工作学习

## Inference

推理指从联合概率分布中计算某些有用的量

枚举推理 Inference by Enumeration:

- **证据变量 (Evidence variables):**  $E_1, \dots, E_r$ , 取值为  $e_1, \dots, e_r$ 。
- **查询变量 (Query variables):**  $Q$ , 即我们关心其概率的变量。
- **隐藏变量 (Hidden variables):**  $H_1, \dots, H_r$ , 这些变量未被观察, 但影响推理结果。
- **目标:** 计算条件概率  $P(Q|e_1, \dots, e_r)$

步骤

1. 选择与证据一致的条目: 从联合分布中筛选出所有与证据  $e_1, \dots, e_r$  匹配的项。

2. 对隐藏变量求和: 计算查询变量  $Q$  与证据  $e_1, \dots, e_r$  的联合概率, 公式为:

$$P(Q, e_1, \dots, e_r) = \sum_{h_1, \dots, h_r} P(Q, e_1, \dots, e_r, h_1, \dots, h_r) \text{ 即对隐藏变量 } H_1, \dots, H_r \text{ 的所有可能取值求和。}$$

3. 归一化

- 计算归一化常数  $Z = P(e_1, \dots, e_r) = \sum_Q P(Q, e_1, \dots, e_r)$ 。
- 最终条件概率为:  $P(Q|e_1, \dots, e_r) = \frac{1}{Z} P(Q, e_1, \dots, e_r)$

但以上枚举推理速度很慢, 联合分布的计算复杂度呈指数级, 导致效率低下

考虑使用变量消除法。该方法虽仍属于 NP - hard 问题，但通常比枚举推理快得多

## Factors

联合分布 Joint Distribution：对于所有  $x, y$ ，其条目为  $P(x, y)$ ，且总和为 1

$P(T, W)$		
$T$	$W$	$P$
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

切片分布 Selected Distribution：联合分布的一个“切片”，固定  $x$  的值，对所有  $y$  取值的  $P(x, y)$  进行汇总。这些条目之和为  $P(x)$ 。公式中大写字母的数量代表表格的维度，例如下图只有  $W$ ，实际为一维表格

$P(cold, W)$		
$T$	$W$	$P$
cold	sun	0.2
cold	rain	0.3

因子 (Factor)：

- $P(Y_1, \dots, Y_N | X_1, \dots, X_M)$  是一个“因子”，本质为**多维数组**
- 数组中的值是具体的条件概率  $P(y_1, \dots, y_N | x_1, \dots, x_M)$ ，即当  $Y_1, \dots, Y_N$  取  $y_1, \dots, y_N$ ， $X_1, \dots, X_M$  取  $x_1, \dots, x_M$  时的条件概率值
- 若  $X$  或  $Y$  被赋值（用小写字母表示，如  $x_1$  或  $y_1$ ），则表示从该多维数组中**缺失（选定）**了一个维度。例如  $X_1 = x_1$  时， $X_1$  这个维度被固定，不再作为变化的维度存在于数组中

由此，枚举推理的步骤在引入因子概念后成为：

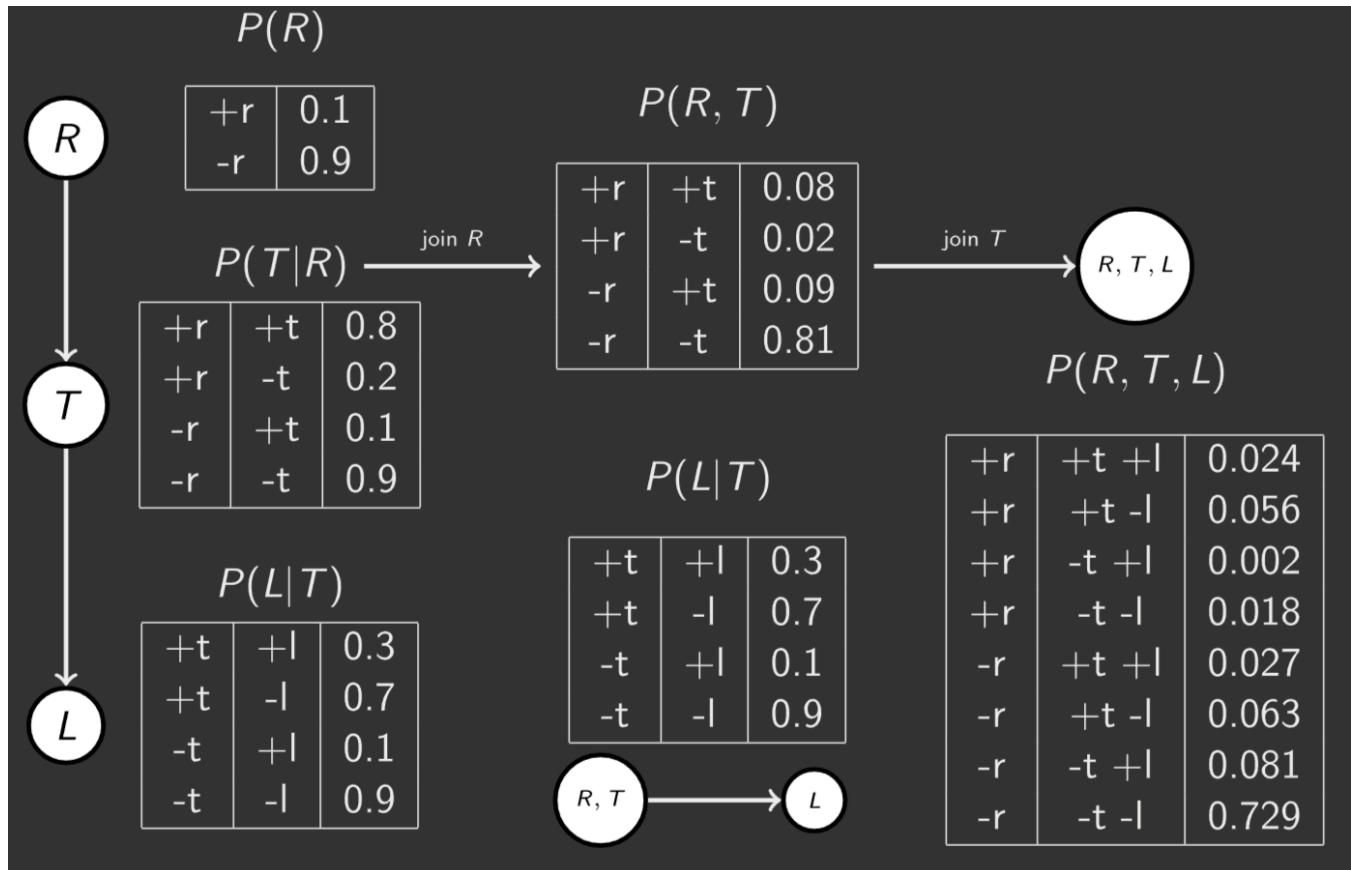
1. **连接所有因子 (Join all factors)**：将所有相关因子整合在一起
2. **消除所有隐藏变量 (Eliminate all hidden variables)**：对不属于查询变量和证据变量的隐藏变量进行求和消除

3. 归一化 (Normalize): 对结果进行归一化处理, 确保概率之和为 1

### 操作: 连接因子 (Join Factors) 两表操作

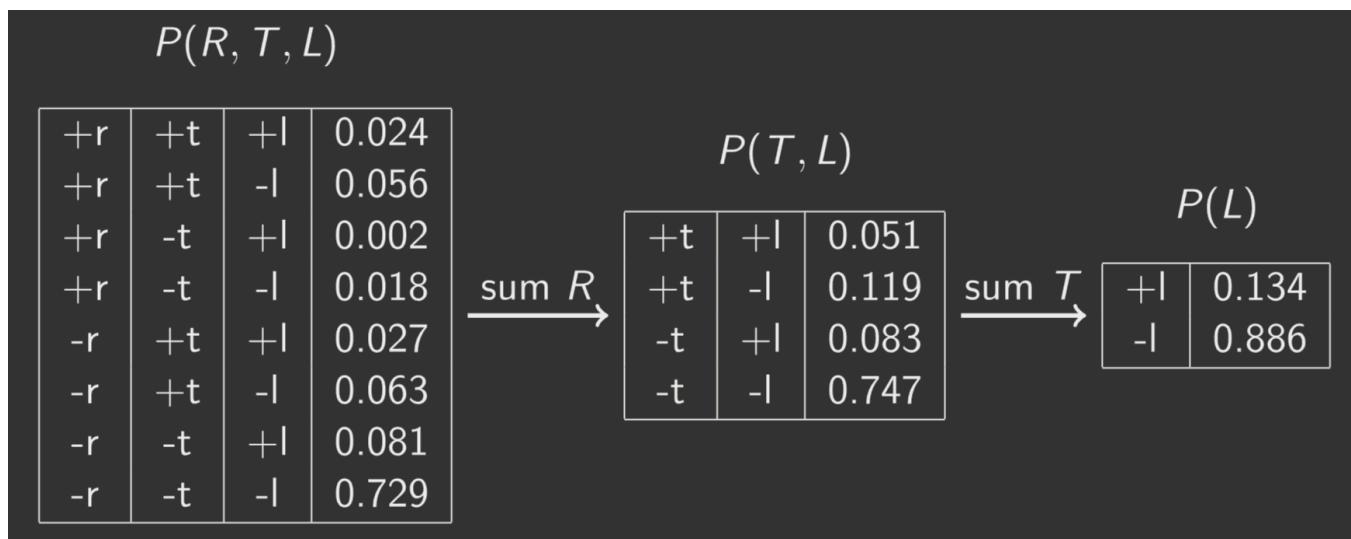
连接因子是基本操作之一, 类似于数据库连接。具体步骤为: 获取所有涉及连接变量的因子, 然后在这些变量的并集上构建新因子

$$\forall r, t : P(r, t) = P(r)P(t | r)$$



### 操作: 消除 (Eliminate) 也称边缘化 (marginalization) 单表内操作

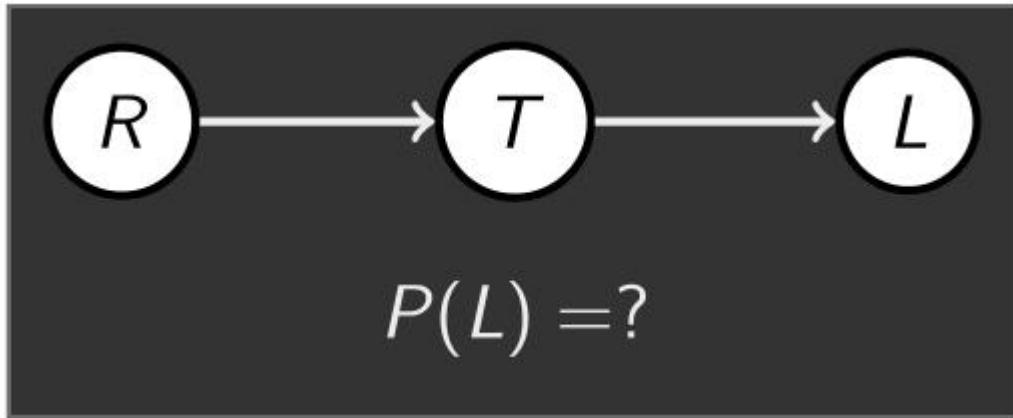
对一个因子中的某一变量进行求和, 从而将该变量从因子中移除, 使因子“缩小”, 是一种投影操作



## Thus Far

枚举推理实际就是先连接后消除

## Variable Elimination (Marginalizing Early)

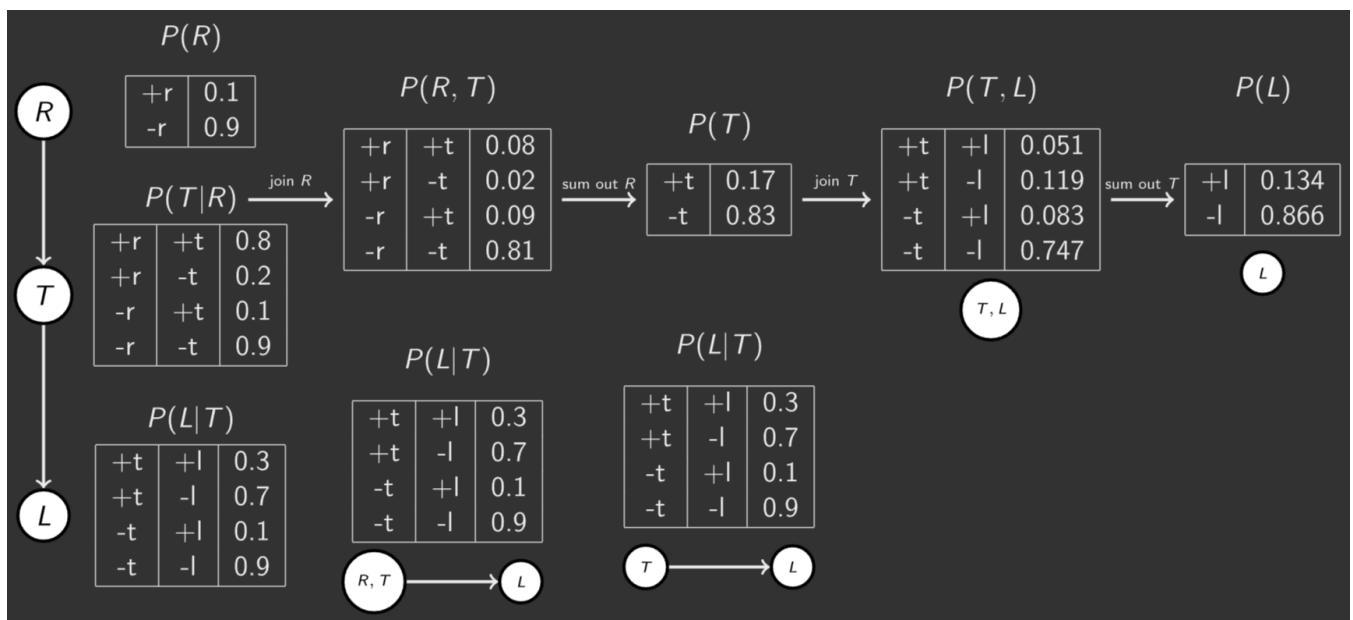


枚举推理：

- 公式:  $\sum_t \sum_r P(L | t)P(r)P(t | r)$
- 步骤
  - 先连接 (join) 变量 r 和 t，整合所有因子。
  - 依次消除 (eliminate) 隐藏变量 r 和 t，计算  $P(L)$ 。

### 变量消除法 (Variable Elimination)

- 公式:  $\sum_t P(L | t) \sum_r P(r)P(t | r)$ 。
- 步骤
  - 先连接 (join) 变量 r。
  - 消除 (eliminate) 变量 r，计算  $\sum_r P(r)P(t | r)$  (即  $P(t)$ )
  - 再连接 (join) 变量 t，最后消除 (eliminate) 变量 t，得到  $P(L)$



当存在证据（即变量具体取值）时，推理从包含该证据的因子开始

例如计算 $P(L|+r)$ 的过程：

The diagram shows the calculation of  $P(L|+r)$  from the joint distribution  $P(R, T, L)$ .

- Joint Distribution  $P(+r, T, L)$ :**

$+r$	$+t$	$+l$	0.08
$+r$	$+t$	$-l$	0.02
$+r$	$-t$	$+l$	0.09
$+r$	$-t$	$-l$	0.81
- Marginal  $P(+r)$ :**

$+r$	0.1
------	-----
- Marginal  $P(T|+r)$ :**

$+r$	$+t$	0.8
$+r$	$-t$	0.2
- Marginal  $P(L|T)$ :**

$+t$	$+l$	0.3
$+t$	$-l$	0.7
$-t$	$+l$	0.1
$-t$	$-l$	0.9

消除查询变量和证据变量之外的所有变量即消除T

得到结果后需要进行归一化

The diagram shows the normalization of the joint distribution  $P(+r, L)$  to calculate  $P(L|+r)$ .

- Joint Distribution  $P(+r, L)$ :**

$+r$	$+l$	0.026
$+r$	$-l$	0.074
- Normalized Distribution  $P(L|+r)$ :**

$+l$	0.26
$-l$	0.74

A horizontal arrow labeled "Normalize" points from the joint distribution to the normalized distribution.

### 通用变量消除法 (General Variable Elimination)

计算 $P(Q | E_1 = e_1, \dots, E_k = e_k)$

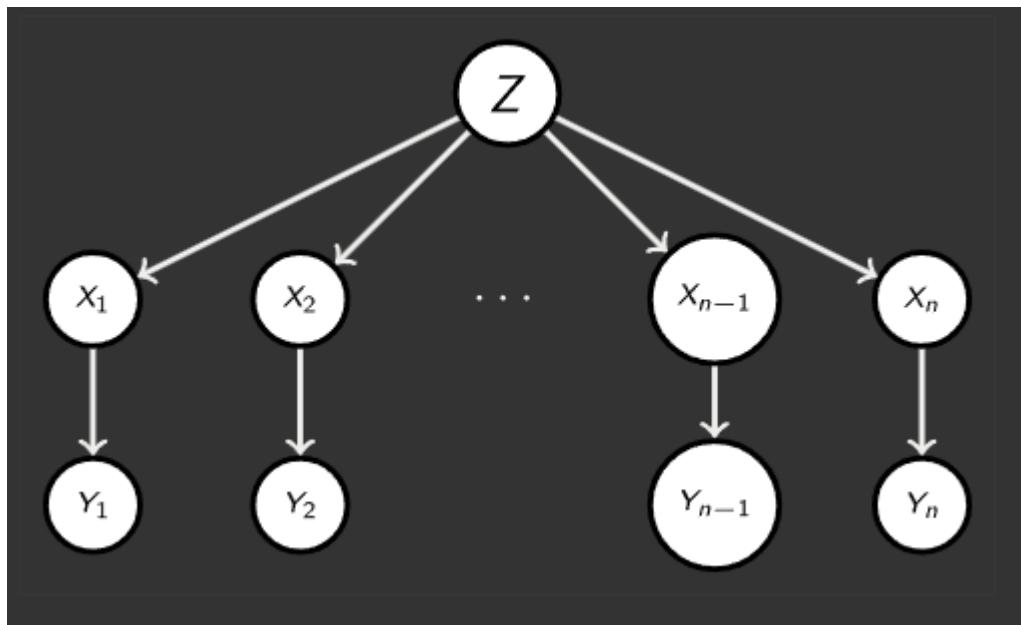
1. **初始化**: 从局部条件概率表 (CPT) 开始，并用证据实例化相关变量（固定证据变量的值）

2. **循环消除隐藏变量**

- 选择一个隐藏变量  $H$  (既非查询变量  $Q$ , 也非证据变量)。
- 连接 (Join) 所有涉及  $H$  的因子 (通过逐点相乘合并因子)。
- 对  $H$  进行消除 (求和), 即对  $H$  的所有取值求和, 从因子中移除  $H$ 。

3. **最终处理**: 循环直至无隐藏变量, 连接剩余所有因子并归一化, 得到  $P(Q | E_1 = e_1, \dots, E_k = e_k)$

- 变量消除法的计算和空间复杂度由过程中产生的 **最大因子** 决定。因子越大, 计算与存储开销越高
- 消除变量的顺序对最大因子的大小有显著影响。



如此网络中, 要求  $P(X_n | y_1, \dots, y_n)$ :

**顺序一**:  $Z, X_1, \dots, X_{n-1}$ 。消除  $Z$  时, 由于  $Z$  与  $X_1, \dots, X_n$  相关, 若变量为二进制, 最大因子大小为  $2^{n+1}$

**顺序二**:  $X_1, \dots, X_{n-1}, Z$ 。先消除  $X_1, \dots, X_{n-1}$ , 最后消除  $Z$ 。此时最大因子仅涉及  $Z$  和  $X_n$ , 二进制下大小为  $2^2$

# 22. Bayesian Networks: Sampling



豆包

你的 AI 助手，助力每日工作学习

# Approximate Inference: Sampling

为何需要采样

学习：从未知分布中获取样本

推理：获取样本比通过变量消去法等精确计算答案更快捷

## Prior Sampling

Prior Sampling:

```
Function prior-sampling(bayes net)
  for  $i = 1, 2, \dots, n$  do
    | Sample  $x_i$  from  $P(X_i | Parents(X_i))$  ;
    | return  $(x_1, x_2, \dots, x_n)$ ;
  end
```

采样  $X_i$  时，依据其条件概率分布  $P(X_i | Parents(X_i))$ 。若  $X_i$  无父节点（根节点），则  $P(X_i | Parents(X_i))$  退化为边缘分布  $P(X_i)$ ；若有父节点，父节点已完成采样，直接利用父节点的当前状态进行采样

例如，对于简单贝叶斯网络  $A \rightarrow B$ ，先按  $P(A)$  采样 A，再根据 A 的采样值，按  $P(B|A)$  采样 B，最终得到样本  $(A, B)$

先验采样生成的样本服从贝叶斯网络的联合概率分布

优点：速度快

缺点：先验采样生成样本基于联合分布  $P(X_1, X_2, \dots, X_n)$ ，无法直接处理已知证据例如要求  $P(C|+w)$ ，难以直接利用  $+w$

## Rejection Sampling

若要估计条件概率  $P(C|+s)$ ，同样统计 C 的结果，但需拒绝  $S \neq +s$  的样本，仅保留  $S = +s$  的样本用于计数。这一过程即为 **拒绝采样**

## Rejection Sampling:

```
Function rejection-sampling(bayes-net,  

evidence-instantiation)  

for i = 1, 2, . . . , n do  

    | Sample xi from  $P(X_i | \text{Parents}(X_i))$  ;  

    | if xi not consistent with evidence then  

    |   | Reject: return – no sample is generated in this cycle  

    | end  

    | return (x1, x2, ..., xn);  

end
```

1. 对贝叶斯网络中的每个变量  $X_i$  进行采样，采样依据是其条件概率分布  $P(X_i | \text{Parents}(X_i))$
2. 检查当前采样的  $x_i$  是否与证据一致。若不一致，直接拒绝该样本

问题：

1. **证据稀有性**：若证据出现概率低，会拒绝大量样本，效率低下。
2. **证据未利用**：采样过程未利用证据信息，仅在最后筛选，未主动结合证据。

## Likelihood Weighting

固定证据变量（如 Color = blue），对其余变量（如 Shape）采样

## Likelihood Weighting:

**Function** *likelihood-weighting(bayes-net, evidence-instantiation)*

```

 $w = 1.0 ;$ 
for  $i = 1, 2, \dots, n$  do
    Sample  $x_i$  from  $P(X_i | Parents(X_i))$  ;
    if  $x_i$  is an evidence variable then
         $X_i = observation x_i$  for  $X_i$  ;
        Set  $w = w * P(x_i | Parents(X_i))$  ;
    end
    else
        Sample  $x_i$  from  $P(X_i | Parents(X_i))$  ;
    end
    return  $(x_1, x_2, \dots, x_n), w$ ;
end

```

对贝叶斯网络中的每个变量  $X_i$  执行以下操作：

- 若  $X_i$  是证据变量
  - 固定  $X_i$  的值为观察值  $x_i$  (即证据给定的值)。
  - 更新权重：设置  $w = w * P(x_i | Parents(X_i))$ ，将证据  $x_i$  在其父节点状态下的条件概率纳入权重，体现证据的影响。
- 若  $X_i$  不是证据变量
  - 按条件概率分布  $P(X_i | Parents(X_i))$  正常采样  $x_i$ 。

返回完整样本  $(x_1, x_2, \dots, x_n)$  及其权重  $w$ 。权重  $w$  累积了证据变量的概率信息，使得样本分布能更高效地逼近目标条件分布，避免了拒绝采样中大量丢弃样本的问题

似然加权通过结合采样分布与证据权重，使最终分布等同于真实联合分布

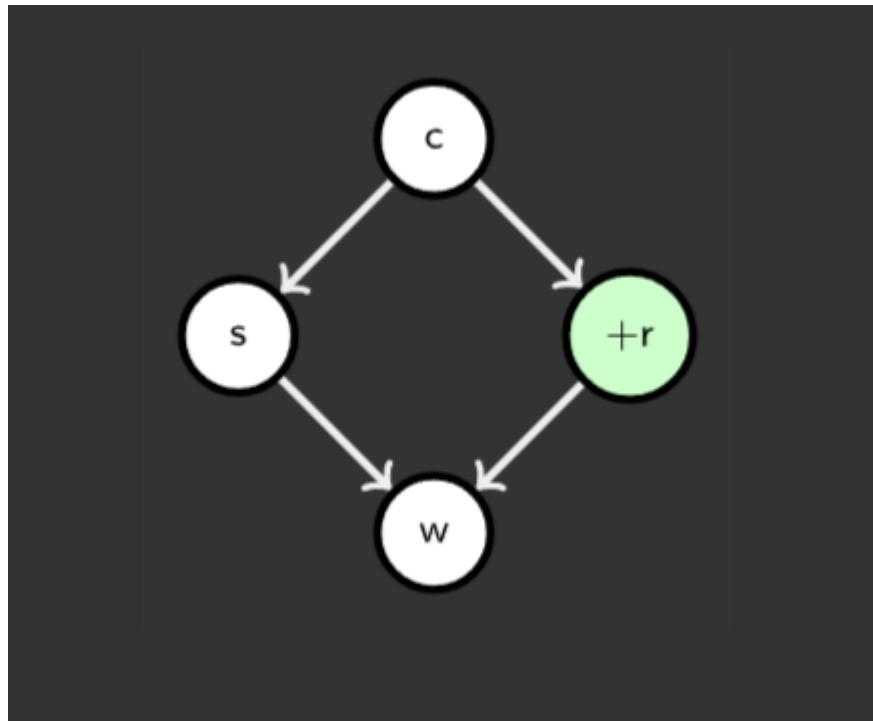
似然加权在生成样本时考虑了证据，但证据仅影响下游变量的选择，对上游变量无效

## Gibbs Sampling

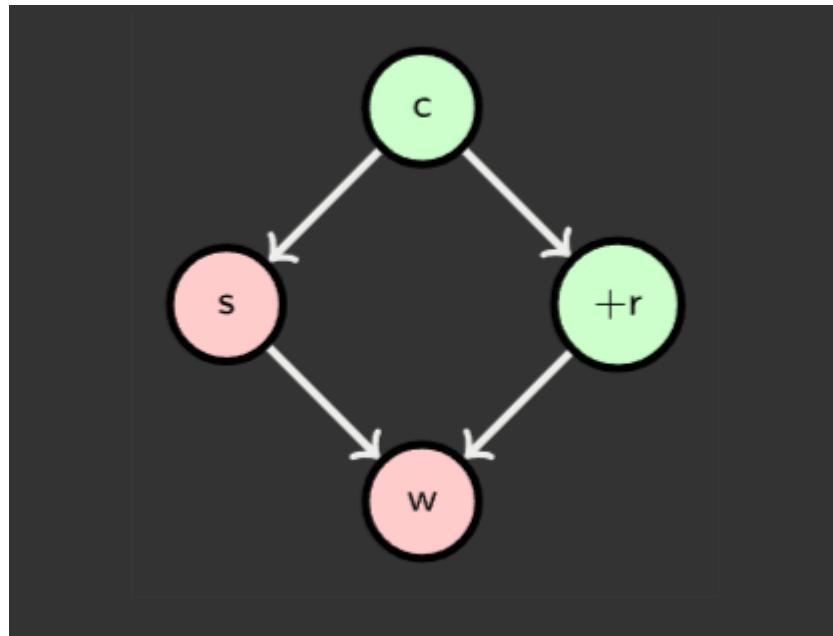
每次基于其他所有变量的当前状态采样一个变量（证据变量固定不变），并长时间重复此过程

上游和下游变量均基于证据进行条件设定。与似然加权（仅基于上游证据，可能导致权重很小，有效样本少）不同，吉布斯采样在采样每个变量时都充分考虑证据，无论其是上游还是下游

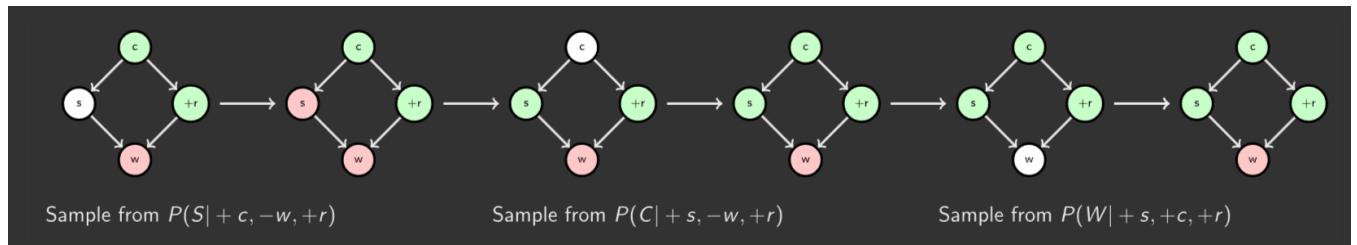
**固定证据 (Step 1)：**明确证据  $R = +r$



**初始化其他变量 (Step 2)：**对非证据变量 C、S、W 进行随机初始化



**循环采样 (Step 3)：**选择不同的固定证据，重复过程



重采样任意变量时，只需考虑该变量及其相关的条件概率表，无需涉及其他无关变量

## Summary

---

- **先验采样 (Prior Sampling)**: 用于估计  $P(Q)$ , 即无证据时目标变量 Q 的边缘概率。它直接从贝叶斯网络的先验分布中生成样本, 无需处理证据。
- **拒绝采样 (Rejection Sampling)**: 用于估计  $P(Q|e)$ , 即给定证据 e 时 Q 的条件概率。通过先验采样生成样本, 拒绝与证据 e 不一致的样本, 仅对符合证据的样本统计 Q 的分布。
- **似然加权 (Likelihood Weighting)**: 同样用于  $P(Q|e)$ 。它固定证据变量, 对非证据变量采样, 并根据证据变量在父节点状态下的概率对样本加权, 避免直接拒绝样本, 提升了效率。
- **吉布斯采样 (Gibbs Sampling)**: 也用于  $P(Q|e)$ 。它维护一个完整的变量实例, 每次基于其他所有变量 (固定证据) 的当前状态更新一个非证据变量, 通过长时间迭代, 样本收敛到基于证据 e 的真实条件分布。

简言之, 先验采样适用于无证据的边缘概率估计, 后三种方法用于有证据的条件概率  $P(Q|e)$ , 但实现机制各有不同: 拒绝采样通过筛选样本, 似然加权通过加权样本, 吉布斯采样通过迭代更新变量来达成目标