1.马尔科夫过程(MP)

(一).马尔科夫过程(Markov Process)是一**种随机过程**，其**未来的状态仅依赖于当前状态**，而与**过去的状态无关**，这个性质称为马尔科夫性质，MP 常用于描述系统的转移和决策过程.

(二).随机过程:它是概率论和统计学中的重要概念，用于描述和分析**随时间变化**的**随机现象.**

2.马尔科夫决策过程(MDP)

(一).通过当前**状态、动作、转移概率**与**奖励**帮助决策者在**不确定的环境**中做出**最优选择**;

(二).马尔科夫决策过程的核心是**马尔科夫性质**，即**当前状态**包含了预测未来状态所需要的所有信息，**过去的状态**和**动作**对**未来没有影响**;

(三).MDP 由五元素组成:
1>.状态空间 S: 所有可能得状态集合;
2>.动作空间 A: **在给定状态下**可采取的所有动作集合;
3>.状态转移概率 P: **给定当状态和所选动作 a**，转移到具体下一个转态的概率;
4>.奖励函数 R: **在给定状态和所选动作**后获得的**即时奖励**;
5>.折扣因子 gamma: 用于权衡未来奖励的重要性，通常 0~1 之间.

(四).MDP 的优化目标: 找到一个**最优的行动策略 pi***,该策略能够**最大化从当前状态开始的期望累积奖励;**

(五).策略迭代法优化 MDP 求解最优策略 pi*包含以下过程:

<5.1>.形式化状态空间 S;

<5.2>.初始化状态价值函数;

在构建和解决马尔科夫决策过程（MDP）时，初始化状态价值函数（或状态-动作价值函数）是可选但常见的步骤。这里解释为什么和如何进行初始化：

是否需要初始化状态价值函数

不必须但推荐

- 在理论上，MDP的算法（如策略迭代、值迭代）可以从任意初始值开始，并通过迭代收敛到最优解。
- 但是，合理的初始化可以加速收敛，减少算法的计算复杂度和迭代次数。

随机初始化

一种简单的方法是随机初始化每个状态的状态价值函数。这种方法不需要任何先验知识，但可能需要更多的迭代步骤来收敛到最优解。

```python
value_function = {state: np.random.rand() for state in states}
```

零初始化

另一种常见的方法是将所有状态的初始状态价值函数设置为0。这是一种保守的初始化方法，假设在没有任何信息的情况下，所有状态的价值都是中性（即0）。

```python
value_function = {state: 0 for state in states}
```

基于经验或先验知识的初始化

如果你有关于问题域的一些先验知识或经验，你可以根据这些信息初始化状态价值函数。例如，在迷宫问题中，你可以根据距离目标的远近来初始化状态价值函数。

```python
def initialize_value_function(states, target_state):
    value_function = {}
    for state in states:
        # 例如，根据距离目标状态的曼哈顿距离初始化
        distance = abs(state[0] - target_state[0]) + abs(state[1] -
target_state[1])
        value_function[state] = -distance
    return value_function

target_state = (2, 2)
value_function = initialize_value_function(states, target_state)
```

<5.3>.形式化动作空间 A;

<5.4>.形式化策略;

3.4.1 策略

智能体的**策略**（Policy）通常用字母π表示。策略$\pi(a|s)=P(A_t=a|S_t=s)$是一个函数，表示在输入状态s情况下采取动作a的概率。当一个策略是**确定性策略**（deterministic policy）时，它在每个状态时只输出一个确定性的动作，即只有该动作的概率为 1，其他动作的概率为 0；当一个策略是**随机性策略**（stochastic policy）时，它在每个状态时输出的是关于动作的概率分布，然后根据该分布进行采样就可以得到一个动作。在 MDP 中，由于马尔可夫性质的存在，策略只需要与当前状态有关，不需要考虑历史状态。回顾一下在 MRP 中的价值函数，在 MDP 中也同样可以定义类似的价值函数。但此时的价值函数与策略有关，这意为着对于两个不同的策略来说，它们在同一个状态下的价值也很可能是不同的。这很好理解，因为不同的策略会采取不同的动作，从而之后会遇到不同的状态，以及获得不同的奖励，所以它们的累积奖励的期望也就不同，即状态价值不同。

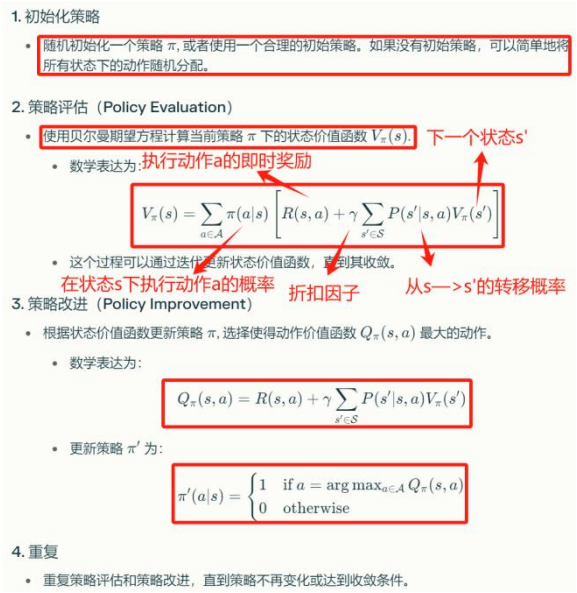
<5.5>.定义折扣因子 gamma;

<5.6>.根据先验知识，构建奖励函数，该函数输入目前的状态 s 与执行的动作 a，输出及时奖励值;

<5.7>.根据先验知识，已形式化的状态空间 S，已形式化的动作空间 A，构建状态转移函数，该函数基于输入的当前状态 s 与执行的动作 a 输出下一个状态 s'与状态转移概率 p(s'|s,a)，p(s'|s,a)表示在状态 s 执行动作 a 后转移至状态 s'的概率;

<5.8>.构建策略迭代:
5.8.1>.策略评估:使用贝尔曼期望方程构建一个基于当前策略更新状态价值函数并使其收敛;
5.8.2>.策略改进:为每个一个状态的所有可能执行动作基于已更新的状态价值函数计算动作价值函数值，并基于最大的动作价值为每一个状态更新最优动作,更新状态执行动作

的过程就是策略改进过程;
5.8.3>.基于策略评估与策略改进构建策略迭代过程来获得最优策略.
5.8.4>.策略迭代过程如下:



$$\pi^0 \xrightarrow{策略评估} V^{\pi^0} \xrightarrow{策略提升} \pi^1 \xrightarrow{策略评估} V^{\pi^1} \xrightarrow{策略提升} \pi^2 \xrightarrow{策略评估} \cdots \xrightarrow{策略提升} \pi^*$$

(六).值迭代法优化 MDP 求解最优策略 pi*包含以下过程:

<6.1>.形式化状态空间 S;

<6.2>.初始化状态价值函数;

<6.3>.形式化动作空间 A;

<6.4>.定义折扣因子 gamma;

<6.5>.定义最大迭代;

<6.6>.定义状态价值函数 V 的收敛阈值 threshold，收敛阈值是为了检查本轮迭代后的每个状态的价值函数值是否都达到了预期的收敛程度，如果有一个状态的价值函数值没有达到预期，则继续迭代优化直至满足收敛预期或者满足最大迭代次数后才能结束优化;

<6.7>.根据先验知识，构建奖励函数;

<6.8>.根据先验知识，构建状态转移函数;

<6.9>.构建值迭代:
6.9.1>.使用贝尔曼最优方程为更新每一个状态价值函数值，其值为此状态可执行动作的

最大动作价值;

贝尔曼最优方程——> $$V^*(s) = \max_{a \in \mathcal{A}} \{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \}$$

6.9.2>.计算**每一个更新后的状态价值函数值**与对应更新前的状态价值函数值的**差值**,并提取其中的最大值 max_delta，使用 max_delta 与 threshold 进行比较判断此时更新后的状态价值函数是否收敛，若收敛则退出值迭代否则继续值迭代直至满足收敛条件或满足最大迭代次数.

<6.10>.最优策略提取:**基于最优状态价值函数**，计算每个状态可能执行所有动作的动作价值，选取动作价值最大的具体动作为此状态的动作，并以此逻辑逐步为所有状态找到最优动作，最终，由状态与其对应的最优动作构成最优策略

(七).定义值函数

7.1>.**状态价值函数 v_pi(s):**表示在状态 s 下，按照策略 pi 采取动作后，预期能获得的总回报。

$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$ 2 3 5.

这里，$\pi(a|s)$ 表示在状态 s 下选择动作 a 的概率。

7.2>.**动作价值函数 q_pi(s,a):**表示在状态 s 下采取动作 a 后，按照策略 pi 采取后续动作，预期能获得的总回报。

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

7.3>.两个值函数的进一步解释

7.3.1>.状态价值函数是**在状态 s 下**，**所有可能动作的动作价值函数的加权和**，**权重**为在**当前状态下**策略选择其动作的**概率**

$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$

这意味着状态价值函数是所有可能动作的动作价值函数的加权和，权重为每个动作在当前状态下的选择概率 2 3.

7.3.2>.动作价值函数包括了**即时奖励**和**未来状态的预期价值**，未来预期价值是根据当前策略计算的

$q_\pi(s,a) =$ 即时奖励 $\mathcal{R}_s^a$ $+ \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$ 未来状态的预期价值

(八).相关知识点补充

<8.1>.状态转移概率函数

在构建马尔科夫决策过程（MDP）时，状态转移概率通常是基于先验知识或通过数据估计得到的。

先验知识

- 在许多情况下，状态转移概率可以通过对问题域的理解和先验知识来确定。例如，在一个迷宫问题中，你可能知道从一个位置移动到另一个位置的概率是如何分布的，这些信息可以直接用于构建状态转移概率矩阵 [1] [2]。

数据估计

- 如果没有足够的先验知识，状态转移概率可以通过观察和数据分析来估计。例如，通过收集历史数据，可以估计出从一个状态转移到另一个状态的概率分布 [2] [5]。

定义和计算

- 状态转移概率在MDP中被定义为从一个状态 s 转移到另一个状态 s' 的概率，通常记为 $P(s'|s,a)$，表示在状态 s 下采取动作 a 后转移到状态 s' 的概率。

<8.2>.奖励函数构建

在马尔科夫决策过程（MDP）中，奖励函数是描述在不同状态下采取不同动作后获得的即时奖励的关键组成部分。以下是如何构建奖励函数的详细解释：

奖励函数的定义

奖励函数 $\mathcal{R}$ 定义了在状态 s 下采取动作 a 后获得的即时奖励，它可以用以下形式表示：

$$\mathcal{R}(s,a) = r$$

这里，r 是在状态 s 下采取动作 a 后获得的奖励。

构建奖励函数的步骤

<8.3>.折扣因子 gamma 的物理含义:远期利益具有一定的不确定性，有时我们希望最优策略关注近期的奖励，所以引入折扣因子对远期的奖励大一些折扣，接近 1 的 gamma 更关注长期的累计奖励，接近 0 的 gamma 更关注短期奖励。

8.4>.贝尔曼最优方程与贝尔曼期望方程

贝尔曼最优方程

贝尔曼最优方程是用于求解最优状态价值函数 $V^*(s)$ 和最优动作价值函数 $Q^*(s,a)$ 的递归方程。它可以写为：

最优状态价值函数

$$V^*(s) = \max_{a\in\mathcal{A}} Q^*(s,a)$$

最优动作价值函数

$$Q^*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a V^*(s')$$

这里，$\mathcal{R}_s^a$ 是在状态 s 下采取动作 a 后获得的即时奖励，γ 是折扣因子，$\mathcal{P}_{ss'}^a$ 是从状态 s 转移到状态 s' 的概率 [1] [2] [4]。

贝尔曼期望方程

贝尔曼期望方程则用于计算给定策略 π 下的状态价值函数 $V_\pi(s)$。它可以写为：

状态价值函数

$$V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s]$$

$$V_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \left[\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a V_\pi(s')\right]$$

动作价值函数

$$Q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a'\in\mathcal{A}} \pi(a'|s') Q_\pi(s',a')$$

这里，$\pi(a|s)$ 是在状态 s 下选择动作 a 的概率 [1] [2]。

8.5>.策略迭代与值迭代的收敛性证明

4.7.1 策略迭代

策略迭代的过程如下：

$$\pi^0 \xrightarrow{\text{策略评估}} V^{\pi^0} \xrightarrow{\text{策略提升}} \pi^1 \xrightarrow{\text{策略评估}} V^{\pi^1} \xrightarrow{\text{策略提升}} \pi^2 \xrightarrow{\text{策略评估}} \cdots \xrightarrow{\text{策略提升}} \pi^*$$

根据策略提升定理，我们知道更新后的策略的价值函数满足单调性，即$V^{\pi^{k+1}} \geq V^{\pi^k}$。所以只要所有可能的策略个数是有限的，策略迭代就能收敛到最优策略。假设 MDP 的状态空间大小为$|\mathcal{S}|$，动作空间大小为$|\mathcal{A}|$，此时所有可能的策略个数为$|\mathcal{A}|^{|\mathcal{S}|}$，是有限个，所以策略迭代能够在有限步找到其中的最优策略。

还有另一种类似的证明思路。在有限马尔可夫决策过程中，如果$\gamma < 1$，那么很显然存在一个上界$C = R_{\max}/(1-\gamma)$（这里的$R_{\max}$为最大单步奖励值），使得对于任意策略π和状态s，其价值$V^{\pi}(s) < C$。因此，对于每个状态s，我们可以将策略迭代得到的价值写成数列$\{V^{\pi^k}\}_{k=1\ldots\infty}$。根据实数列的单调有界收敛定理，该数列一定收敛，也即是策略迭代算法一定收敛。

4.7.2 价值迭代

价值迭代的更新公式为：

$$V^{k+1}(s) = \max_{a \in A}\{r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V^k(s')\}$$

我们将其定义为一个**贝尔曼最优算子**$\mathcal{T}$：

$$V^{k+1}(s) = \mathcal{T}V^k(s) = \max_{a \in A}\{r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V^k(s')\}$$

然后我们引入**压缩算子**（contraction operator）：若O是一个算子，如果满足$\|OV - OV'\|_q \leq \|V - V'\|_q$条件，则我们称$O$是一个压缩算子。其中$\|x\|_q$表示$x$的$L_q$范数，包括我们将会用到的无穷范数$\|x\|_\infty = \max_i |x_i|$。

我们接下来证明当$\gamma < 1$时，贝尔曼最优算子$\mathcal{T}$是一个γ-压缩算子。

$$\begin{aligned}
\|\mathcal{T}V - \mathcal{T}V'\|_\infty &= \max_s \left| \max_{a \in A}\{r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s')\} - \max_{a' \in A}\{r(s,a') + \gamma \sum_{s' \in S} P(s'|s,a')V'(s')\} \right| \\
&\leq \max_{s,a} \left| r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s') - r(s,a) - \gamma \sum_{s' \in S} P(s'|s,a)V'(s') \right| \\
&= \gamma \max_{s,a} \left| \sum_{s' \in S} P(s'|s,a)(V(s') - V'(s')) \right| \\
&\leq \gamma \max_{s,a} \sum_{s' \in S} P(s'|s,a) \max_{s'} |V(s') - V'(s')| \\
&= \gamma \|V - V'\|_\infty
\end{aligned}$$

将V'设为最优价值函数V^*，于是有：

$$\|V^{k+1} - V^*\|_\infty = \|\mathcal{T}V^k - \mathcal{T}V^*\|_\infty \leq \gamma \|V^k - V^*\|_\infty \leq \cdots \leq \gamma^{k+1}\|V^0 - V^*\|_\infty$$

这意味着，在$\gamma \leq 1$的情况下，随着迭代次数k越来越大，V^k会越来越接近V^*，即$\lim_{k \to \infty} V^k = V^*$。至此，价值迭代的收敛性得到证明。

3.蒙特卡洛方法

(一).定义:一种基于**随机抽样**的**统计模拟**技术，其基本思想是通过**生成大量的随机样本**，利用这些样本的**统计特性**来估计目标值，特别适用于难以获得**解析解**的问题。

(二).蒙特卡洛方法的基本步骤:
<2.1>.确定随机变量:**识别与问题相关的随机变量**，并分析它们的性质和相关关系;

<2.2>.为随机变量构建概率分布模型:**为识别的随机变量构建适合的概率分布**，构建的概率分布越接近识别随机变量的真实概率分布，蒙特卡洛方法解决问题精度越高;

<2.3>.基于概率分布模型生成随机样本:使用概率分布模型生成随机样本;

<2.4>.构建解决问题的模型并进行模拟:基于随机样本构建解决问题数学模型，进行问题模拟输出模拟结果;

<2.5>.对模拟结果进行分析，例如计算期望，方差，并绘制相应的概率分布图等

4.GNAS 蒙特卡洛树搜索(MCTS)案例

(一)GNAS 搜索问题定义:
给定一个图神经网络的搜索空间 GNN Search Space，搜索空间包含 7 个结构组件，每个结构组件有其候选项集合如下:

perturbation_intensity_candidate = [0.5, 0.1, 0.05, 0.01];
perturbation_probability_candidate = [0.1, 0.3, 0.7];
distance_aggregator_candidate = ["5", "10", "30"];
feature_aggregator_candidate = ["GCNConv", "SGConv", "None"];
updator_dimension_candidate = [128, 256, 512, 1024];
updator_activation_candidate = ["LeakyRelu", "Relu", "Relu6"];
fusion_candidate = ["cat", "sum", "mean", "max", "weighted_cat", "weighted_sum"];

每个结构组件采样一个候选项可以组成一个 GNN 结构，例如:
GNN_Architecture = [0.5,0.1,"5","SGConv",128,"Relu","cat"]
每个 GNN_Architecture 送入评估函数后将返回一个验证评估反馈，例如:
Validation Score = estimator(GNN_Architecture).
搜索任务:在 GNN Search Space 识别出最优的 Validation Score 对应的 GNN_Architecture

(二).蒙特卡洛树搜索:是一种用于决策过程的**启发式搜索算法**，特别适用于**大型、高复杂度**的搜索空间，如:棋类游戏、规划问题和强化学习等。MCTS 的核心思想是通过**随机模型(蒙特卡洛方法)**来评估搜索树中的节点价值，从而指导搜索过程.

(三).GNAS 场景下的蒙特卡洛树搜索流程

<1>创建节点类:

1>.**属性:**
(1)当前节点的**父节点对象**;
(2)当前节点的**子节点列表**;
(3)当前节点**对应子节点**的组件索引;
(4)当前节点**被访问次数**;
(5)当前节点**累计评估值**;
(6)当前节点在本层树结构**可扩展的子节点候选列表**（**此候选项**对应的**下一层组件**还未探索可选择的候选项）
(7)当前节点**是否扩展完毕标志位**。

2>.**方法:**
(1)**判断当前节点是否为终端节点,返回**是否是终端节点标志位;
(2)**判断当前节点是否已完全扩展完毕**(对于当前候选项是否还有**下一层组件候选项**需要扩展),**返回**是否扩展完毕节点标志位;
(3)**从当前节点可扩展子节点列表中扩展一个子节点，如果本节点无可扩展子节点则将当前节点扩展完毕标志位置 True,返回**扩展子节点对象;
(4)使用最优子节点选择策略(一般是 UCB1 策略)选择当前节点的最优子节点进行探索，

如果有多个最优子节点则随机选择一个，**返回**当前节点的最优子节点对象;

<2>创建树策略:——>[**输入**访问节点——>**输出**访问节点的子节点]
ps:包含选子节点择与扩展操作，**优先扩展(expansion)未探索的子节点**，当所有子节点探索完毕后再选择(selection)最优子节点进行探索
(1)**输入需要进行扩展或选择的节点对象**;
(2)判断节点是否是终止节点，若不是则基于输入节点对象递归访;
(3)判断节点是否有**可扩展的子节点**,如果没有则扩展子节点并**返回**子节点对象**退出**上层循环;
(4)如果当前节点全部子节点已扩展完毕则**使用最优节点选择策略探索子节点**,**返回**最优子节点对象**继续探索过程**;
(5)若使用 UCB1 最优节点选择方法，探索率权重越大，则蒙特卡洛树偏向于**广度优先**方式建树—>更慢到达最终叶子节点，若探索率权重越小，则蒙特卡洛树偏向于**深度优先**方式建树—>更快到达最终稿的叶子节点

- **UCB1 算法：** 在节点选择时，使用 UCB1 公式平衡探索和利用。
  - **公式：** `score = (total_value / visits) + exploration_weight * sqrt(log(parent.visits) / visits)`
  - **解释：**
    - **平均价值（Exploit）：** `total_value / visits`，表示节点的平均评估值。
    - **探索项（Explore）：** `exploration_weight * sqrt(log(parent.visits) / visits)`，鼓励选择访问次数少的节点。

(6)**返回扩展子节点对象或终端节点对象.**

<3>创建模拟(simulation)函数:——>[**输入**被访问节点的**扩展**或**选择**子节点对象——>**输出**基于此子节点进行模拟的奖励分数]
(1)**输入**节点对象;
(2)通过改节点回溯其父节点构建已探索的 GNN 结构组件;
(3)使用模拟策略获得剩余 GNN 结构组件候选项构建完整的 GNN 结构(一般可使用随机模拟策略（蒙特卡洛方法）);
(4)将完整的 GNN 结构送入 GNN 结构评估函数获得评估反馈构成此次树探索的奖励,**返回**奖励分数;

<4>回溯(backpropagation)本次树探索结果:——>[**输入扩展**或**选择**子节点对象，奖励分数，折扣因子——>**输出:无,根据在树中访问的路径跟新从此子节点到 root 节点对象的相关属性值**]
(1)**输入**节点，奖励分数，折扣因子;
(2)判断节点对象是不是根节点的父节点 None,不是给此节点对象被访问次数属性增加 1，累计评估值属性增加奖励分数*折扣因子，回溯其父节点;
(3)若回溯至 None 节点则退出回溯;

<5>构建蒙特卡洛搜索逻辑输出最优 GNN 结构
(1)输入根节点对象，迭代次数，折扣因子;
(2)构建循环进行树搜索:
<1>.执行树搜索策略**输入根节点**(每一次迭代都从 root 节点出发)**返回扩展或选择**的

子节点;

<2>.对**扩展或选择**的子节点进行模拟,**输入**子节点**返回**奖励分数;

<3>.对**扩展或选择**的子节点进行回溯,**输入**子节点、奖励分数，折扣因子;

(3)对蒙特卡洛搜索过程生成的整个树结构从**根节点**开始进行**子节点扩展**与**最优子节点选择**构建最优 GNN 结构，当迭代结束后可以基于每个子节点的被访问次数、平均评估值（积累估值/被访问次数）等其他有意义的指标判断最优子节点。

(4)最优子节点构建最优 GNN 架构，计算最优子节点序列最后一个终端节点的平均奖励分数（积累估值/被访问次数）作为此最优 GNN 架构的最优验证评估性能指标

(5)**返回**最优 GNN 架构，最优 GNN 架构验证性能

(四).输出最优 GNN 结构过程中最优子节点选择知识点补充:

在蒙特卡洛树搜索（Monte Carlo Tree Search, MCTS）过程中，每个节点的访问次数和累计评估值都扮演着重要的角色。它们用于指导搜索过程，帮助算法在庞大的搜索空间中平衡探索新节点和利用已有信息。

1. 节点被访问次数的意义：

- **定义：** 节点的访问次数（`visits`）表示在整个搜索过程中，该节点被访问（经过或选择）的总次数。
- **意义：**
  - **探索程度的指标：** 访问次数较少的节点表示探索程度较低，可能包含尚未充分挖掘的潜在优质解。
  - **指导选择策略：** 在选择节点时，访问次数用于计算探索项，帮助算法平衡探索和利用。

2. 节点的累计评估值的意义：

- **定义：** 节点的累计评估值（`total_value`）是从该节点出发的所有模拟结果（评估值）的总和。
- **意义：**
  - **性能的累积指标：** 累计评估值越高，表示从该节点出发的模拟在总体上取得了较好的评估结果。
  - **用于计算平均评估值：** 通过累计评估值和访问次数，可以计算节点的平均评估值（`total_value / visits`），反映从该节点出发的平均性能。

3. 为什么使用节点的访问次数来选择最优节点：

在构建最佳 GNN 架构时，我们从根节点开始，在每一层选择访问次数最多的子节点，原因如下：

- **代表性强：** 访问次数最多的节点意味着在搜索过程中，该节点被认为是具有潜力的选择，被算法多次选中。
- **统计稳定性：** 访问次数多的节点，其平均评估值（`total_value / visits`）更具有统计意义，受单次模拟结果的波动影响较小。
- **探索与利用的结果：** MCTS 的核心思想是通过大量的模拟来估计节点的价值。访问次数多的节点，通常是经过算法充分探索和评估的，体现了算法对该节点的信心。

4. 为什么不直接使用节点的累计评估值来选择最优节点：

- **规模差异：** 累计评估值没有考虑访问次数的影响。一个访问次数少但累计评估值高的节点，其平均评估值可能并不高，可能是偶然取得的高评估值。
- **避免偏差：** 直接使用累计评估值可能会偏向于访问次数多、但平均评估值低的节点，因为它们的累计评估值可能较高，但性能并不好。
- **平均评估更合理：** 平均评估值（`total_value / visits`）能够更客观地反映节点的真实性能，因为它平衡了评估值和访问次数。

5. 综合考虑：

在选择最优节点时，使用节点的访问次数来构建最佳架构是一种常见的策略，但也可以考虑使用平均评估值或其他指标。具体来说：

- **访问次数最多的节点：**
  - **优势：** 代表算法认为最有希望的路径，经过了充分的探索。
  - **劣势：** 可能包含平均评估值不高的节点。
- **平均评估值最高的节点：**
  - **优势：** 直接反映节点的平均性能。
  - **劣势：** 如果访问次数较少，平均评估值可能不稳定，受随机性的影响较大。

7. 是否可以使用平均评估值来选择节点：

确实，可以考虑使用平均评估值（`total_value / visits`）来选择最佳节点。这可能在某些情况下更能反映节点的性能。但是需要注意：

- **数据可靠性：** 如果节点的访问次数较少，其平均评估值可能不可靠，受单次模拟结果影响较大。
- **过拟合风险：** 过于依赖少数高评估值的节点，可能导致选择的路径在总体上并不理想。

8. 结合访问次数和平均评估值：

一种折衷的做法是同时考虑节点的访问次数和平均评估值。例如，可以在访问次数超过一定阈值的节点中，选择平均评估值最高的节点。

示例修改：

python Copy code

```python
# 从根节点开始，构建最佳架构
best_architecture = []
node = root
while not node.is_terminal():
    # 在访问次数超过阈值的子节点中，选择平均评估值最高的
    threshold = 10  # 访问次数阈值，可根据需求调整
    qualified_children = [child for child in node.children if child.visits >= threshold]
    if qualified_children:
        node = max(qualified_children, key=lambda c: c.total_value / c.visits)
    else:
        # 如果没有满足条件的节点，仍选择访问次数最多的
        node = max(node.children, key=lambda c: c.visits)
    best_architecture.append(node.component_value)
```

(五).蒙特卡洛搜索树搜索过程中的约束方法:

2. 调整 UCB 算法的探索参数

- **方法：** 调整 UCB1 中的探索参数 `c`，使算法更倾向于利用已知的高价值节点，减少对新节点的探索。
- **实现：** 在 `best_child` 方法中，减小 `exploration_weight` 的值，例如从 `√2` 调整为 `0.5`。

python Copy code

```python
def best_child(self, exploration_weight=0.5):
    # 其余代码保持不变
```

注意： 减小探索参数可能导致算法陷入局部最优，需要根据实际情况权衡。

5. 剪枝和约束

a. 剪枝无效或不良节点

- **方法：** 对于已知不可能产生高评估值的节点，直接剪枝，避免浪费计算资源。
- **实现：** 在节点扩展时，判断当前组件组合是否满足一定条件，不满足则不扩展该节点。

b. 设定搜索约束

- **方法：** 根据需求，设置架构的参数量、深度、计算复杂度等约束，缩小搜索空间。
- **实现：** 在 `expand` 方法中，添加约束条件的检查。