基于代表节点的重叠社区发现程序设计与改进说明

1 原方法与程序更改构思

1.1 原方法的基本思想

本项目的基础方法是基于代表节点的重叠社区发现方法。该方法的基本思路是：先对网络进行非重叠社区划分，得到一组基础社区；然后在每个基础社区中选择一个具有代表性的节点；最后根据普通节点与各社区代表节点之间的相似度，判断该节点是否可能同时属于多个社区。

设输入网络为加权无向图：

$$G = (V, E, w), \tag{1.1}$$

其中，V 表示节点集合，E 表示边集合，$w(u, v)$ 表示节点 u 与节点 v 之间的连接权重。如果两个节点之间不存在边，则有：

$$w(u, v) = 0. \tag{1.2}$$

经过非重叠社区划分后，网络中的每个节点先被分配到一个基础社区中。设基础社区集合为：

$$C = \{C_1, C_2, \ldots, C_K\}, \tag{1.3}$$

其中，K 表示基础社区数量。对于任意节点 v，其基础社区编号可以记为：

$$v \to z_v. \tag{1.4}$$

在代表节点方法中，每个基础社区 C_c 都会选择一个代表节点 r_c：

$$r_c \in C_c. \tag{1.5}$$

该代表节点用于表示社区 C_c 的整体特征。之后，节点 v 与社区 C_c 的关系不再直接通过 v 与整个社区中所有节点的关系来判断，而是通过节点 v 与代表节点 r_c 的相似度来

近似表示：

$$sim(v, c) = sim(v, r_c). \tag{1.6}$$

在给定阈值 t 的情况下，如果节点 v 与社区 c 的代表节点相似度超过阈值，即：

$$sim(v, c) > t, \tag{1.7}$$

则认为节点 v 可能属于社区 c。因此，在阈值 t 下，节点 v 的候选社区集合可以表示为：

$$C_v(t) = \{c \mid sim(v, c) > t\}. \tag{1.8}$$

如果一个节点同时对应多个候选社区，则说明它可能处在多个社区的交界位置。此时可以将该节点判定为重叠节点：

$$v \in O_t \Longleftrightarrow |C_v(t)| > 1, \tag{1.9}$$

其中，O_t 表示阈值 t 下识别出的重叠节点集合。

1.2 原方法在程序实现中的问题

单个真实节点对整个社区的表达能力有限。当社区内部存在多个局部中心，或者社区边界较模糊时，代表节点的选择会直接影响后续相似度计算结果。

相似度计算方式对网络结构较敏感。对于稀疏图而言，许多节点之间并不存在直接边。如果直接使用节点之间的边权作为相似度，那么大量相似度值可能为 0，导致潜在的社区关系无法被识别。对于密图而言，节点之间连接较充分，直接边权本身又可能具有较强的结构含义。

仅依赖单个阈值结果缺少稳定性判断。对于同一组相似度结果，不同阈值会产生不同的节点-社区归属关系。如果相邻阈值下的结果变化很大，说明当前阈值附近的候选结果不稳定。仅直接采用某一个阈值对应的结果，容易受到参数选择的影响。

1.3 当前程序的总体更改构思

当前程序仍然保留如下核心判定方式：

$$C_v(t) = \{c \mid sim(v, c) > t\}, \tag{1.10}$$

$$v \in O_t \Longleftrightarrow |C_v(t)| > 1. \tag{1.11}$$

节点是否被判定为重叠节点，仍然取决于它在某个阈值下是否同时满足多个社区的相似度条件。

将原本较长的程序流程拆分为三个部分。Part 1 负责数据读取、基础社区选择、代

表节点构造和相似度计算；Part 2 负责阈值扫描、候选重叠社区生成和候选过滤；Part 3 负责候选结果质量评价和最终候选选择。后续只修改代表节点构造方式，主要影响 Part 1；如果只修改候选筛选策略，主要影响 Part 3；中间的阈值扫描和候选生成逻辑可以保持相对稳定。

构思了基于虚拟几何中心的社区代表方式。在虚拟几何中心方法中，社区不再完全依赖某一个真实节点表示，而是通过社区内所有节点的特征平均值来表示社区整体。

程序首先为每个节点构造社区连接向量：

$$x_v = [x_{v,0}, x_{v,1}, \ldots, x_{v,K-1}], \tag{1.12}$$

其中，第 c 维表示节点 v 与社区 c 中所有节点之间的连接强度总和：

$$x_{v,c} = \sum_{u:z_u=c,u\neq v} w(v,u). \tag{1.13}$$

对于某个社区 C，可以将该社区中所有节点的社区连接向量求平均，得到该社区的虚拟几何中心：

$$g_C = \frac{1}{|C|}\sum_{v\in C} x_v. \tag{1.14}$$

该向量 g_C 并不是图中的真实节点，而是用于表示社区整体连接特征的虚拟中心。相比单个真实代表节点，虚拟几何中心能够更直接地综合社区内部所有节点的结构信息。

根据网络稀疏程度选择不同的相似度计算方式。程序使用图密度判断网络类型：

$$\rho = \frac{2|E|}{n(n-1)}, \tag{1.15}$$

其中，$|E|$ 为网络边数，n 为节点数量。当 ρ 较小时，网络较稀疏；当 ρ 较大时，网络较密集。当前程序采用如下判断：

$$\rho < 0.25 \Rightarrow similarityType = 0, \tag{1.16}$$

$$\rho \geq 0.25 \Rightarrow similarityType = 1. \tag{1.17}$$

在稀疏图模式下，程序使用向量形式的相似度。若采用虚拟几何中心，则节点 v 与社区 C 的相似度可表示为：

$$sim(v,C) = \frac{x_v \cdot g_C}{\|x_v\|\|g_C\|}. \tag{1.18}$$

该公式关注的是节点 v 与社区整体中心在社区连接分布上的相似程度，而不是两者之间是否存在一条直接边。

在密图模式下，由于节点之间连接较充分，可以使用节点到社区整体的平均边权来

表示节点与社区的关系：

$$sim(v, C) = \frac{1}{|C| - 1} \sum_{u \in C, u \neq v} w(v, u). \tag{1.19}$$

该值表示节点 v 与社区 C 中其他节点的平均连接强度。相比只看节点与单个代表节点之间的边权，这种方式更能反映节点与整个社区之间的直接联系。

将单一阈值判断改为阈值扫描。程序不再只依赖某一个固定阈值，而是在给定范围内扫描多个阈值：

$$t = bottomThreshold,\ bottomThreshold + 0.01, \ldots, topThreshold. \tag{1.20}$$

每一个阈值都会生成一个候选结果：

$$Candidate(t_i). \tag{1.21}$$

所有候选结果组成候选集合：

$$\mathcal{C} = \{Candidate(t_1), Candidate(t_2), \ldots, Candidate(t_k)\}. \tag{1.22}$$

这样做的目的是避免结果完全依赖单个阈值，而是先生成多个可能结果，再从中选择较稳定、较可靠的候选。

在候选生成之后加入过滤和筛选机制。对于稀疏图，程序使用隶属度过滤弱节点-社区对。设节点 v 在阈值 t 下的候选社区集合为 $C_v(t)$，则节点 v 在这些候选社区中的总连接强度为：

$$W_v = \sum_{c \in C_v(t)} x_{v,c}. \tag{1.23}$$

节点 v 对社区 c 的隶属度定义为：

$$m(v, c) = \frac{x_{v,c}}{W_v}. \tag{1.24}$$

如果某个节点-社区对的隶属度过低，则说明节点与该社区的连接占比不足。程序设置过滤条件：

$$m(v, c) \geq 0.05. \tag{1.25}$$

不满足条件的节点-社区对 (v, c) 会被删除。这里删除的不是节点本身，而是该节点对某个社区的弱归属关系。

对于候选结果的最终选择，程序引入节点-社区归属集合：

$$A_t = \{(v, c) \mid v \in O_t,\ c \in C_v(t)\}. \tag{1.26}$$

该集合记录了当前候选结果中所有重叠节点的具体社区归属关系。为了衡量候选结果在相邻阈值下是否稳定，程序计算归属稳定性：

$$AffStab(i) = \frac{1}{N_i} \sum_j \frac{|A_i \cap A_j|}{|A_i \cup A_j|}. \tag{1.27}$$

其中，A_i 表示第 i 个候选结果的节点-社区归属集合，A_j 表示其相邻候选结果的归属集合，N_i 表示参与比较的相邻候选数量。该指标越高，说明当前阈值附近的节点-社区归属关系越稳定。

程序还使用隶属度质量指标 $Mem(t)$ 衡量候选结果中重叠节点的多社区归属是否具有足够支撑：

$$Mem(t) = \frac{1}{|O_t|} \sum_{v \in O_t} \min_{c \in C_v(t)} m(v, c). \tag{1.28}$$

该指标关注每个重叠节点最弱的社区归属。如果一个节点虽然被分到多个社区，但其中某个社区的隶属度很低，则该节点对候选整体质量的贡献也会较低。

当前程序从原来的“代表节点 + 单阈值判断”扩展为“代表节点或虚拟中心 + 自适应相似度 + 多阈值候选 + 候选质量筛选”的结构。

2 Part 1：数据读取、基础社区划分与相似度计算

Part 1 将原始输入数据转换为后续阈值扫描所需的中间数据。完成网络读取、真实社区读取、非重叠基础社区划分、代表节点或虚拟中心构造，以及节点与各社区之间相似度表的计算。

2.1 Part 1 的总体输入与输出

Part 1 的输入包括两个文件。第一个文件是网络文件，用于描述节点之间的边和边权；第二个文件是真实社区文件，用于保存基准社区划分。

网络文件可以表示为若干条加权边：

$$(u_i, v_i, w_i), \tag{2.1}$$

其中，u_i 和 v_i 表示边的两个端点，w_i 表示这条边的权重。程序读取这些边后，构造加权无向图：

$$G = (V, E, w), \tag{2.2}$$

其中，V 表示节点集合，E 表示边集合，$w(u, v)$ 表示节点 u 和节点 v 之间的连接权重。如果两个节点之间不存在边，则有：

$$w(u, v) = 0. \tag{2.3}$$

真实社区文件用于提供基准社区信息。程序读取后会构造两类映射。第一类是社区到节点的映射：

$$T_c = \{v \mid v \in c\}, \tag{2.4}$$

它表示真实社区 c 中包含哪些节点。第二类是节点到社区的映射：

$$T_v = \{c \mid v \in c\}, \tag{2.5}$$

它表示节点 v 在真实结果中属于哪些社区。该结构主要用于后续导出真实社区结果，并交给评价程序计算指标。

Part 1 的输出主要包括以下几类中间数据：

$$v \to z_v, \tag{2.6}$$

$$c \to r_c \quad 或 \quad c \to g_c, \tag{2.7}$$

$$v \to x_v, \tag{2.8}$$

$$v \to \{(r_c, sim(v,c))\}_{c=0}^{K-1}. \tag{2.9}$$

其中，z_v 表示节点 v 的基础社区编号，r_c 表示真实代表节点版本中的社区代表节点，g_c 表示虚拟几何中心版本中的社区中心向量，x_v 表示节点 v 的社区连接向量，$sim(v,c)$ 表示节点 v 与社区 c 的相似度。

2.2 基础社区划分的生成与选择

Part 1 首先运行若干非重叠社区发现算法，生成多个基础社区划分候选。每一个非重叠划分可以表示为：

程序会将多个算法得到的划分结果保存到统一的候选集合中。

程序优先选择社区数量与真实社区数量一致的基础划分：

如果不存在完全一致的划分，则选择社区数量最接近真实社区数量的划分：

最终得到的基础社区划分记为：

$$v \to z_v. \tag{2.10}$$

2.3 改进版：基于真实代表节点的 Part 1

在改进版本中，Part 1 继续使用真实节点作为社区代表节点。选择“社区内部连接强、社区外部连接弱”的节点。

程序首先计算每个节点的总加权度：

$$d(v) = \sum_{u \in V, u \neq v} w(v, u). \tag{2.11}$$

该值表示节点 v 与全图所有其他节点之间的连接权重总和。

对于某个基础社区 C 中的节点 v，程序计算其社区内度：

$$d_{in}(v) = \sum_{u \in C, u \neq v} w(v, u). \tag{2.12}$$

该值表示节点 v 与本社区内部其他节点的连接强度总和。

社区外度则由总度减去社区内度得到：

$$d_{out}(v) = d(v) - d_{in}(v). \tag{2.13}$$

在此基础上，程序定义节点 v 在社区 C 中的代表性分数：

$$R(v) = d_{in}(v) - d_{out}(v). \tag{2.14}$$

该分数越大，说明节点 v 与本社区内部连接越强，同时与社区外部连接越弱，因此越适合作为该社区的代表节点。最终，社区 C 的代表节点定义为：

$$r_C = \arg\max_{v \in C} R(v). \tag{2.15}$$

2.4 社区连接向量

为了计算节点与社区之间的关系，程序会为每个节点构造一个社区连接向量。设基础社区数量为 K，则节点 v 的社区连接向量为：

$$x_v = [x_{v,0}, x_{v,1}, \ldots, x_{v,K-1}]. \tag{2.16}$$

其中，第 c 维表示节点 v 与基础社区 c 中所有节点之间的连接权重总和：

$$x_{v,c} = \sum_{u : z_u = c, u \neq v} w(v, u). \tag{2.17}$$

该向量在后续有两个作用。在稀疏图模式下，它用于计算节点与代表节点之间的余弦相似度；在 Part 2 和 Part 3 中，它还会用于隶属度过滤和候选质量指标计算。

2.5 改进版本中的 similarityType 自动选择

改进版本在 Part 1 中加入了相似度类型自动选择机制。程序通过图密度判断当前网络更接近稀疏图还是密图。图密度定义为：

$$\rho = \frac{2|E|}{n(n-1)}, \tag{2.18}$$

其中，$|E|$ 表示边数，n 表示节点数，$\frac{n(n-1)}{2}$ 表示无向图最多可能拥有的边数。

程序采用如下规则：

$$\rho < 0.25 \Rightarrow similarityType = 0, \tag{2.19}$$

$$\rho \geq 0.25 \Rightarrow similarityType = 1. \tag{2.20}$$

当 $similarityType = 0$ 时，程序按照稀疏图处理，使用社区连接向量的余弦相似度。节点 v 与社区 c 的代表节点 r_c 的相似度为：

$$sim(v, c) = \frac{x_v \cdot x_{r_c}}{\|x_v\| \|x_{r_c}\|}. \tag{2.21}$$

其中：

$$x_v \cdot x_{r_c} = \sum_{i=0}^{K-1} x_{v,i} x_{r_c,i}, \tag{2.22}$$

$$\|x_v\| = \sqrt{\sum_{i=0}^{K-1} x_{v,i}^2}. \tag{2.23}$$

该相似度关注的是节点 v 和代表节点 r_c 在各个社区连接分布上的相似程度，而不是二者之间是否存在直接边。对于稀疏图而言，这种方法可以缓解直接边缺失带来的相似度为零问题。

当 $similarityType = 1$ 时，程序按照密图处理，直接使用节点 v 与社区代表节点 r_c 之间的边权作为相似度：

$$sim(v, c) = w(v, r_c). \tag{2.24}$$

如果节点 v 本身就是代表节点 r_c，则设置：

$$sim(v, c) = 1. \tag{2.25}$$

密图中节点之间连接较充分，直接边权本身可以表达节点与代表节点之间的关系，因此该方式更加直接。

2.6 几何中心版本：基于改进的 Part 1 修改

在改进版本基础上，进一步构造了虚拟几何中心版本。该版本仅修改 Part 1。

几何中心版本的基本思路是：不再完全依赖某个真实节点代表整个社区，而是尝试用社区内所有节点的平均特征来表示该社区。具体来说，仍然先为每个节点构造社区连接向量：

$$x_v = [x_{v,0}, x_{v,1}, \ldots, x_{v,K-1}], \tag{2.26}$$

其中：

$$x_{v,c} = \sum_{u:z_u=c, u\neq v} w(v,u). \tag{2.27}$$

对于某个社区 C，将社区中所有节点的向量求平均，得到该社区的虚拟几何中心：

$$g_C = \frac{1}{|C|}\sum_{v\in C} x_v. \tag{2.28}$$

这里的 g_C 不是网络中的真实节点，而是一个向量形式的虚拟中心。它表示社区 C 内所有节点在社区连接空间中的平均位置。因此，几何中心版本希望用 g_C 表示社区的整体连接特征，而不是只用某一个真实节点作为代表。

在稀疏图模式下，节点 v 与社区 C 的相似度定义为节点向量与虚拟中心向量之间的余弦相似度：

$$sim(v,C) = \frac{x_v \cdot g_C}{\|x_v\|\|g_C\|}. \tag{2.29}$$

在密图模式下，由于虚拟几何中心不是图中的真实节点，不能直接计算 $w(v,g_C)$。因此，使用节点 v 到社区 C 中其他节点的平均边权表示节点与社区整体的连接强度：

$$sim(v,C) = \frac{1}{|C|-1}\sum_{u\in C, u\neq v} w(v,u). \tag{2.30}$$

为了保持 Part 1 和 Part 2 的接口稳定，几何中心版本在输出相似度表时仍然保留原有数据结构形式：

$$v \to \{(r_c, sim(v,c))\}_{c=0}^{K-1}. \tag{2.31}$$

其中，$sim(v,c)$ 已经由虚拟几何中心计算得到；而 r_c 仍然使用一个真实节点编号作为社区标识，以保证 Part 2 可以继续通过该编号找到对应社区。也就是说，在几何中心版本中，真实节点编号主要用于保持接口一致，并不再承担相似度计算中的实际代表作用。

3 Part 2：阈值扫描、候选生成与候选导出

Part 1 已经完成了网络读取、基础社区划分、代表节点或虚拟中心构造，以及节点与各社区之间相似度的计算。

Part 2 在此基础上，对不同阈值进行扫描，并在每个阈值下判断节点可能属于哪些社区，从而得到一组候选重叠社区结果。

当前 Part 2 实际上存在两个版本。第一个是一般版本，用于正常程序运行。它负责生成候选结果，并调用 Part 3 从候选集合中选择最终结果。第二个是用于 Oracle Gap 分析的候选导出版本。该版本不以当前筛选策略为最终结果，而是导出每个阈值下的中间候选社区文件，供后续评价程序逐一计算 NMI，并找出候选集合中的理论最优结果。

3.1 Part 2 在程序中的位置

Part 2 的输入来自 Part 1。Part 1 为每个节点计算了它与各社区之间的相似度，它只需要读取相似度值并执行阈值判断。

Part 2 的核心任务是对一组阈值进行扫描。对于每个阈值 t，程序根据条件：

$$sim(v, c) > t \tag{3.1}$$

判断节点 v 是否可能属于社区 c。所有满足条件的社区组成节点 v 在阈值 t 下的候选社区集合：

$$C_v(t) = \{c \mid sim(v, c) > t\}. \tag{3.2}$$

如果一个节点在当前阈值下对应多个候选社区，则该节点被视为当前阈值下的候选重叠节点。对应的重叠节点集合可以表示为：

$$O_t = \{v \mid |C_v(t)| > 1\}. \tag{3.3}$$

Part 2 的本质是将 Part 1 得到的连续相似度结果转化为不同阈值下的离散候选重叠社区结构。

3.2 一般版本 Part 2 的总体流程

一般版本的 Part 2 主要由两个核心函数构成。第一个是 `getOverlapNodes()`，用于扫描阈值并生成候选结果；第二个是 `showNodeOverlapCom()`，用于根据最终阈值重新生成并导出最终社区结果。

一般版本 Part 2 的整体流程可以概括为：

$$\text{相似度表} \Rightarrow \text{阈值扫描} \Rightarrow \text{候选社区集合} \Rightarrow \text{候选过滤} \Rightarrow \text{候选质量计算} \Rightarrow \text{调用 Part 3.} \tag{3.4}$$

在程序中，每个阈值都会生成一个候选结果，记为：

$$Candidate(t_i). \tag{3.5}$$

所有候选结果组成候选集合：

$$\mathcal{C} = \{Candidate(t_1), Candidate(t_2), \ldots, Candidate(t_k)\}. \tag{3.6}$$

每个候选结果内部保存当前阈值下的重叠节点、节点到社区的映射、节点-社区归属关系集合，以及候选质量指标。其核心数据可以表示为：

$$Candidate(t) = (t,\ O_t,\ C_v(t),\ A_t,\ Mem(t),\ AffStab(t),\ Sep(t)). \tag{3.7}$$

其中，A_t 表示节点-社区归属集合：

$$A_t = \{(v, c) \mid v \in O_t,\ c \in C_v(t)\}. \tag{3.8}$$

该集合比单纯的重叠节点集合 O_t 更细，因为它不仅记录哪些节点是重叠节点，还记录每个重叠节点具体属于哪些社区。

3.3 阈值扫描与初始候选社区生成

Part 2 首先确定阈值扫描范围。设 Part 1 传入的阈值下界和上界分别为 $bottomThreshold$ 和 $topThreshold$，程序按照固定步长 0.01 进行扫描：

$$t = bottomThreshold,\ bottomThreshold + 0.01, \ldots, topThreshold. \tag{3.9}$$

对于每个阈值 t，程序遍历所有节点 v 以及所有社区 c。如果节点与社区的相似度满足：

$$sim(v, c) > t, \tag{3.10}$$

则将社区 c 加入节点 v 的候选社区集合：

$$C_v(t) = C_v(t) \cup \{c\}. \tag{3.11}$$

扫描完成后，可以得到一个节点到候选社区集合的映射：

$$v \to C_v(t). \tag{3.12}$$

该结构在程序中对应 `nodeOverlapCom`。它的含义是：在当前阈值 t 下，每个节点可能属于哪些社区。

例如，如果存在：

$$C_8(t) = \{1, 3\}, \tag{3.13}$$

则表示节点 8 在当前阈值下同时可能属于社区 1 和社区 3，因此节点 8 是当前阈值下的候选重叠节点。

3.4 稀疏图模式下的隶属度过滤

初始阈值判断只考虑 $sim(v, c) > t$，但这可能会产生一些较弱的节点-社区归属关系。尤其在稀疏图中，节点与社区之间的关系更适合用节点到社区整体的连接占比来衡量。使用隶属度过滤。

设节点 v 在阈值 t 下的候选社区集合为 $C_v(t)$。节点 v 与社区 c 的连接强度为：

$$x_{v,c} = \sum_{u:z_u=c, u\neq v} w(v, u). \tag{3.14}$$

节点 v 在所有候选社区中的总连接强度为：

$$W_v = \sum_{c\in C_v(t)} x_{v,c}. \tag{3.15}$$

节点 v 对社区 c 的隶属度定义为：

$$m(v, c) = \frac{x_{v,c}}{W_v}. \tag{3.16}$$

该值表示社区 c 在节点 v 的候选归属中占多大比例。如果 $m(v, c)$ 很小，说明节点 v 虽然在相似度阈值上被分到了社区 c，但它与该社区整体的连接占比很低，这种归属更可能是噪声。

因此，程序设置隶属度过滤条件：

$$m(v, c) \geq 0.05. \tag{3.17}$$

不满足该条件的节点 v 对社区 c 的归属关系会被删除：

$$(v, c). \tag{3.18}$$

稀疏图隶属度过滤会修改：

$$C_v(t), \tag{3.19}$$

使其中弱连接的社区编号被删除。如果过滤后节点 v 只剩一个候选社区，或者没有候选社区，则该节点不再作为当前阈值下的重叠节点。

3.5 密图模式下的最大断崖过滤

在密图模式下，节点之间连接较多，许多节点可能与多个社区代表节点都有较高相似度。如果仅依靠阈值 $sim(v,c) > t$，一个节点可能被分配到过多社区。因此，一般版本 Part 2 在密图模式下使用最大断崖过滤。

对于节点 v 的候选社区集合 $C_v(t)$，程序先取出这些候选社区对应的相似度，并从大到小排序：

$$s_1 \geq s_2 \geq \cdots \geq s_k. \tag{3.20}$$

然后计算相邻相似度之间的差值：

$$gap_i = s_i - s_{i+1}. \tag{3.21}$$

最大断崖定义为：

$$bestGap = \max_i gap_i. \tag{3.22}$$

如果最大断崖出现在 s_j 和 s_{j+1} 之间，则程序保留断崖之前的社区，删除断崖之后的社区。断崖之前的社区与节点 v 的关系较强，断崖之后的社区相似度明显下降，更可能是弱相关社区。

为了衡量该节点的相似度分离程度，程序定义节点分离度：

$$Sep_v = \frac{bestGap}{maxScore}, \tag{3.23}$$

其中，$maxScore = s_1$ 表示该节点最高的候选相似度。

对于一个阈值候选结果，所有参与计算的节点分离度取平均，得到候选整体分离度：

$$Sep(t) = \frac{1}{N} \sum_v Sep_v. \tag{3.24}$$

该指标越大，说明候选社区中强相关社区与弱相关社区之间的分界越清楚。该值会被保存到候选结果的 `separationScore` 中，并在 Part 3 的密图最终筛选中使用。

3.6 候选质量 Mem 计算

在完成稀疏图隶属度过滤或密图断崖过滤后，Part 2 会对当前阈值下的候选结果计算基础质量指标。该过程主要由 `fillCandidateQuality()` 完成。

首先，程序根据过滤后的 $C_v(t)$ 判断重叠节点。

然后，将每个重叠节点对应的社区归属展开为节点-社区对：

$$A_t = \{(v, c) \mid v \in O_t,\ c \in C_v(t)\}. \tag{3.25}$$

例如，如果：

$$C_8(t) = \{1, 3\}, \tag{3.26}$$

则会产生两个节点-社区归属关系：

$$(8, 1), \quad (8, 3). \tag{3.27}$$

随后，程序计算候选结果的隶属度质量 $Mem(t)$。对于每个重叠节点 v，先计算它对候选社区的隶属度：

$$m(v, c) = \frac{x_{v,c}}{\sum_{c' \in C_v(t)} x_{v,c'}}. \tag{3.28}$$

然后取该节点所有候选社区中的最小隶属度：

$$minMem(v) = \min_{c \in C_v(t)} m(v, c). \tag{3.29}$$

最后，对所有重叠节点求平均：

$$Mem(t) = \frac{1}{|O_t|} \sum_{v \in O_t} minMem(v). \tag{3.30}$$

展开后可以写为：

$$Mem(t) = \frac{1}{|O_t|} \sum_{v \in O_t} \min_{c \in C_v(t)} \frac{x_{v,c}}{\sum_{c' \in C_v(t)} x_{v,c'}}. \tag{3.31}$$

该指标关注的是重叠节点最弱的社区归属是否具有足够支撑。如果一个节点虽然被分配到多个社区，但其中某个社区的隶属度很低，那么该节点对应的 $minMem(v)$ 也会较低，从而降低整个候选结果的 $Mem(t)$。

3.7 一般版本 Part 2 与 Part 3 的衔接

一般版本 Part 2 会将所有有效候选结果保存到候选集合：

$$\mathcal{C} = \{Candidate(t_1), Candidate(t_2), \ldots, Candidate(t_k)\}. \tag{3.32}$$

随后，Part 2 调用 Part 3 中的候选选择函数，从候选集合中选择最终候选：

$$Candidate(t^*) = \underset{Candidate(t_i)\in\mathcal{C}}{\arg\operatorname{select}} \; Candidate(t_i). \tag{3.33}$$

其中，t^* 表示最终选择的阈值。Part 3 的选择依据包括归属稳定性 $AffStab$、隶属度质量 Mem、密图分离度 Sep、重叠节点数量以及节点-社区归属数量等指标。

在得到最终阈值 t^* 后，一般版本 Part 2 会调用 `showNodeOverlapCom()`，根据最终阈值重新生成最终的节点-社区归属关系。对于非重叠节点，保留其基础社区：

$$myparts[v] = \{z_v\}. \tag{3.34}$$

对于最终识别出的重叠节点，使用其过滤后的候选社区集合作为最终社区归属：

$$myparts[v] = C_v(t^*). \tag{3.35}$$

因此，最终输出可以写为：

$$myparts[v] = \begin{cases} \{z_v\}, & v \notin O_{t^*}, \\ C_v(t^*), & v \in O_{t^*}. \end{cases} \tag{3.36}$$

最后，程序将该结果导出为：

$$myComData.Dat, \tag{3.37}$$

并同时导出真实社区结果：

$$TrueComData.Dat. \tag{3.38}$$

这两个文件用于后续评价程序计算 NMI、Recall、Precision 和 F-score 等指标。

3.8 Oracle Gap 版本 Part 2 的设计目的

除了正常运行的一般版本 Part 2 外，程序还构造了用于 Oracle Gap 分析的 Part 2 版本。该版本的目的不是直接生成最终算法结果，而是导出阈值扫描过程中产生的所有候选结果，并让后续评价程序逐一计算这些候选结果的 NMI。

一般版本 Part 2 的流程是：

$$\mathcal{C} \Rightarrow Part\ 3 \Rightarrow Candidate(t^*) \Rightarrow myComData.Dat. \tag{3.39}$$

Oracle Gap 版本 Part 2 的流程是：

$$\mathcal{C} \Rightarrow \{myComData(t_1), myComData(t_2), \ldots, myComData(t_k)\}. \tag{3.40}$$

也就是说，Oracle Gap 版本不会只导出 Part 3 选择的一个候选，而是将每一个有效阈值候选都导出为独立的社区结果文件。

这样可以回答一个关键问题：当前算法结果不好时，问题究竟出在候选生成阶段，还是出在 Part 3 的最终选择阶段。

如果候选集合中存在某个候选结果的 NMI 很高，而 Part 3 选择的结果 NMI 较低，则说明 Part 2 已经生成了较好的候选，但 Part 3 没有选中它。此时优化重点应放在 Part 3 的候选选择规则上。

如果候选集合中所有候选结果的 NMI 都较低，则说明 Part 2 生成的候选集合本身质量不足。此时即使 Part 3 选择最优候选，也难以得到较好结果，优化重点应回到 Part 1 或 Part 2，例如代表节点构造、相似度计算、阈值范围设置或候选过滤方法。

3.9 Oracle Gap 版本 Part 2 的导出内容

Oracle Gap 版本 Part 2 与一般版本 Part 2 在候选生成逻辑上保持一致。

区别在于，Oracle Gap 版本在每个候选结果生成后，会将该候选结果导出。对于第 i 个候选结果，可以记为：

$$Candidate(t_i) \Rightarrow myComData(t_i). \tag{3.41}$$

同时，程序还会导出候选索引信息，用于记录每个候选对应的阈值、重叠节点数量、节点-社区归属数量以及候选质量指标。可以表示为：

$$Index(t_i) = (t_i,\ |O_{t_i}|,\ |A_{t_i}|,\ Mem(t_i),\ AffStab(t_i),\ Sep(t_i))\,. \tag{3.42}$$

后续评价程序读取这些候选社区文件，并分别计算：

$$NMI(Candidate(t_i)). \tag{3.43}$$

最终取候选集合中的最大值作为 Oracle NMI：

$$NMI_{oracle} = \max_i NMI(Candidate(t_i)). \tag{3.44}$$

一般版本最终选择结果对应的 NMI 记为：

$$NMI_{selected}. \tag{3.45}$$

两者差值定义为：

$$NMI_{gap} = NMI_{oracle} - NMI_{selected}. \tag{3.46}$$

进一步计算相对差距：

$$GapRatio = \frac{NMI_{oracle} - NMI_{selected}}{NMI_{oracle}}. \tag{3.47}$$

该指标越大，说明当前 Part 3 选择结果与候选集合中的理论最优结果之间差距越大。

4 Part 3：候选质量评价与最终结果选择

Part 2 负责扫描阈值并生成多个候选重叠社区结果，而 Part 3 的任务是在这些候选结果中选出一个最终结果。

设 Part 2 生成的候选集合为：

$$\mathcal{C} = \{Candidate(t_1), Candidate(t_2), \ldots, Candidate(t_k)\}. \tag{4.1}$$

其中，每一个候选 $Candidate(t_i)$ 对应一个阈值 t_i。该候选结果中保存了当前阈值下的重叠节点集合、节点-社区归属关系、隶属度质量以及密图分离度等信息。Part 3 的目标可以表示为：

$$Candidate(t^*) = \underset{Candidate(t_i)\in\mathcal{C}}{\arg\operatorname{select}} \; Candidate(t_i), \tag{4.2}$$

其中，t^* 表示最终被选中的阈值。

4.1 Part 3 的输入数据

Part 3 的输入是 Part 2 构造出的候选集合。对于任意一个候选 $Candidate(t)$，其主要内容可以表示为：

$$Candidate(t) = (t,\ O_t,\ C_v(t),\ A_t,\ Mem(t),\ Sep(t)). \tag{4.3}$$

其中，t 是当前候选对应的阈值；O_t 是当前阈值下识别出的重叠节点集合：

$$O_t = \{v \mid |C_v(t)| > 1\}; \tag{4.4}$$

$C_v(t)$ 是节点 v 在当前阈值下的候选社区集合：

$$C_v(t) = \{c \mid sim(v, c) > t\}; \tag{4.5}$$

A_t 是节点-社区归属关系集合：

$$A_t = \{(v, c) \mid v \in O_t,\ c \in C_v(t)\}. \tag{4.6}$$

4.2 归属稳定性 AffStab

Part 3 首先计算每个候选结果的归属稳定性。该指标用于衡量一个候选结果在相邻阈值下是否稳定。这里的稳定性不是只看重叠节点数量是否变化，而是看更细的节点-社区归属关系是否变化。

对于第 i 个候选结果，其节点-社区归属集合记为：

$$A_i = \{(v, c) \mid v \in O_{t_i},\ c \in C_v(t_i)\}. \tag{4.7}$$

程序会将候选 i 与其附近的若干候选进行比较。当前版本中，邻近窗口取 2，即候选 i 会与 $i-2$、$i-1$、$i+1$、$i+2$ 这些候选比较。记候选 i 的邻居集合为：

$$\mathcal{N}(i) = \{j \mid 0 < |i - j| \le 2,\ 1 \le j \le k\}. \tag{4.8}$$

对于候选 i 和候选 j，程序使用 Jaccard 相似度比较它们的节点-社区归属集合：

$$J(A_i, A_j) = \frac{|A_i \cap A_j|}{|A_i \cup A_j|}. \tag{4.9}$$

其中，$A_i \cap A_j$ 表示两个候选结果中共同存在的节点-社区归属关系，$A_i \cup A_j$ 表示两个候选结果中出现过的所有节点-社区归属关系。如果两个候选结果非常接近，则交集接近并集，$J(A_i, A_j)$ 接近 1；如果两个候选差异较大，则交集较小，$J(A_i, A_j)$ 接近 0。

候选 i 的归属稳定性定义为它与邻近候选 Jaccard 相似度的平均值：

$$AffStab(i) = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} J(A_i, A_j). \tag{4.10}$$

代入 Jaccard 相似度后，可写为：

$$AffStab(i) = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \frac{|A_i \cap A_j|}{|A_i \cup A_j|}. \tag{4.11}$$

该指标越大，说明阈值在 t_i 附近变化时，节点-社区归属关系越稳定。反之，如果 $AffStab(i)$ 较低，则说明该候选对阈值变化较敏感，可靠性相对较弱。

4.3 隶属度质量 Mem

Mem 用于衡量候选结果中重叠节点的多社区归属是否具有足够支撑。

对于节点 v，其在阈值 t 下的候选社区集合为 $C_v(t)$。节点 v 与社区 c 的连接强度记为：

$$x_{v,c}. \tag{4.12}$$

节点 v 在候选社区集合中的总连接强度为：

$$W_v = \sum_{c \in C_v(t)} x_{v,c}. \tag{4.13}$$

节点 v 对社区 c 的隶属度定义为：

$$m(v,c) = \frac{x_{v,c}}{W_v}. \tag{4.14}$$

程序对每个重叠节点取其所有候选社区中的最小隶属度：

$$Mem(t) = \frac{1}{|O_t|} \sum_{v \in O_t} \min_{c \in C_v(t)} \frac{x_{v,c}}{\sum_{c' \in C_v(t)} x_{v,c'}}. \tag{4.15}$$

该指标越大，说明候选结果中的重叠节点对多个社区的归属更加均衡，也更有连接支撑。如果 $Mem(t)$ 较低，则说明候选结果中可能存在较多弱归属关系。

4.4 密图分离度 Sep

Sep 主要用于密图模式。密图中节点之间连接较多，一个节点可能与多个社区代表节点都有较高相似度。如果只依靠阈值判断，容易产生过多候选社区。Part 2 计算分离度 Sep。

对于节点 v，设其候选社区对应的相似度从大到小排序为：

$$s_1 \geq s_2 \geq \cdots \geq s_k. \tag{4.16}$$

相邻相似度差值为：

$$gap_i = s_i - s_{i+1}. \tag{4.17}$$

最大断崖定义为：

$$bestGap = \max_i gap_i. \tag{4.18}$$

节点 v 的分离度定义为：

$$Sep_v = \frac{bestGap}{maxScore}, \tag{4.19}$$

其中：

$$maxScore = s_1. \tag{4.20}$$

对一个候选结果而言，所有参与断崖过滤节点的分离度取平均，得到候选整体分离度：

$$Sep(t) = \frac{1}{N} \sum_v Sep_v. \tag{4.21}$$

该指标越高，说明候选社区中强相关社区和弱相关社区之间分界越清楚。在密图模式下，Sep 可以帮助程序避免将节点过度扩展到过多社区。

4.5 分位数门槛的计算

在得到所有候选的 $AffStab$、Mem 和 Sep 后，Part 3 会计算若干分位数门槛，用于筛选候选结果。

对于一组候选指标值：

$$X = \{x_1, x_2, \ldots, x_k\}, \tag{4.22}$$

将其从小到大排序后，分位数可以表示为：

$$Q_p(X) = X_{\lfloor (k-1)p \rfloor}. \tag{4.23}$$

是归属稳定性的中位数：

$$medianAffStab = Q_{0.5}(AffStab). \tag{4.24}$$

隶属度质量的下四分位数：

$$memFloor = Q_{0.25}(Mem). \tag{4.25}$$

密图分离度的中位数：

$$sepFloor = Q_{0.5}(Sep). \tag{4.26}$$

$medianAffStab$ 用于要求候选结果的稳定性至少达到候选集合中的中等水平；$memFloor$

用于排除隶属度质量明显偏低的候选；$sepFloor$ 用于密图模式下判断候选结果的分离度是否偏弱。

4.6 重叠节点数量上限 overlapSizeUpper

在稀疏图模式下，Part 3 还会计算重叠节点数量上限 $overlapSizeUpper$。该指标用于限制最终候选的重叠节点规模，防止低阈值产生过多候选重叠节点。

程序首先在满足基本质量条件的候选中寻找较保守的重叠节点规模。基本条件为：

$$AffStab(i) \geq medianAffStab, \tag{4.27}$$

$$Mem(i) \geq memFloor. \tag{4.28}$$

在满足这两个条件的候选中，找到最小的重叠节点数量：

$$S_{min} = \min_i |O_i|. \tag{4.29}$$

然后设置重叠节点数量上限为：

$$overlapSizeUpper = 1.20 \times S_{min}. \tag{4.30}$$

这样做的含义是：先找到一个稳定性和隶属度质量都不差的保守候选规模，然后允许最终结果在该规模基础上适当放宽，但不允许候选规模无限扩张。

4.7 稀疏图模式下的候选选择规则

当 $g\_overlapDenseMode = false$ 时，程序按照稀疏图规则选择最终候选。稀疏图模式下的选择分为两步：先筛选合格候选，再在合格候选中选择节点-社区归属关系更充分的结果。

首先，候选 i 必须满足以下条件：

$$AffStab(i) \geq medianAffStab, \tag{4.31}$$

$$Mem(i) \geq memFloor, \tag{4.32}$$

$$|O_i| \leq overlapSizeUpper. \tag{4.33}$$

满足上述条件的候选组成合格候选集合：

$$\mathcal{Q} = \{Candidate(i) \mid AffStab(i) \geq medianAffStab, \ \cdots\}. \tag{4.34}$$

如果合格候选集合非空，则程序在其中优先选择节点-社区归属关系数量最多的候选：

$$Candidate(t^*) = \underset{Candidate(i)\in\mathcal{Q}}{\arg\max} |A_i|. \tag{4.35}$$

其中，$|A_i|$ 表示候选 i 中节点-社区归属关系的数量。在已经满足稳定性、隶属度和规模约束的前提下，$|A_i|$ 越大，说明该候选保留的重叠归属信息越充分。

当多个候选具有相同的 $|A_i|$ 时，程序依次比较：

$$AffStab(i), \quad Mem(i), \quad t_i. \tag{4.36}$$

在归属关系数量相同的情况下，优先选择稳定性更高、隶属度质量更高、阈值更高的候选。

如果没有任何候选满足上述筛选条件，即：

$$\mathcal{Q} = \varnothing, \tag{4.37}$$

程序会进入兜底选择。兜底质量分定义为：

$$quality(i) = AffStab(i) \times Mem(i). \tag{4.38}$$

此时选择：

$$Candidate(t^*) = \underset{Candidate(i)\in\mathcal{C}}{\arg\max} quality(i). \tag{4.39}$$

4.8 密图模式下的候选选择规则

当 $g\_overlapDenseMode = true$ 时，程序按照密图规则选择最终候选。密图中节点连接较多，候选归属关系容易膨胀。如果仍然像稀疏图一样优先选择 $|A_i|$ 最大的候选，可能会导致节点被过度分配到多个社区。因此，密图模式下不再以归属关系数量作为主要目标，而是使用稳定性、隶属度质量和分离度的乘积作为候选质量分。

对于候选 i，其质量分定义为：

$$quality(i) = AffStab(i) \times Mem(i) \times Sep(i). \tag{4.40}$$

该公式的含义是：一个密图候选结果需要同时满足三个条件。首先，$AffStab(i)$ 较高，说明相邻阈值下结果稳定；其次，$Mem(i)$ 较高，说明多社区归属有连接支撑；最后，$Sep(i)$ 较高，说明强相关社区和弱相关社区之间分界清楚。

如果候选 i 的分离度低于中位数：

$$Sep(i) < sepFloor, \tag{4.41}$$

则程序对其质量分进行惩罚：

$$quality(i) = 0.5 \times quality(i). \tag{4.42}$$

最终，密图模式选择质量分最高的候选：

$$Candidate(t^*) = \underset{Candidate(i)\in\mathcal{C}}{\arg\max}\ quality(i). \tag{4.43}$$

如果多个候选质量分相同，程序优先选择节点-社区归属关系数量较少的候选：

$$\min |A_i|. \tag{4.44}$$

如果仍然相同，则选择阈值更高的候选：

$$\max t_i. \tag{4.45}$$

这样做是为了在密图中保持结果相对保守，减少由于边较密而造成的过度归属。

4.9 Part 3 输出结果

Part 3 最终输出的是候选集合中的最佳候选下标和最终阈值。设最终选择的候选为：

$$Candidate(t^*). \tag{4.46}$$

对应的最终阈值为：

$$overlapThreshold = t^*. \tag{4.47}$$

Part 2 会根据该阈值重新生成最终社区归属，并输出最终结果。对于非重叠节点，保留基础社区：

$$myparts[v] = \{z_v\}. \tag{4.48}$$

对于最终识别出的重叠节点，使用候选社区集合作为最终归属：

$$myparts[v] = C_v(t^*). \tag{4.49}$$

因此最终输出结构为：

$$myparts[v] = \begin{cases} \{z_v\}, & v \notin O_{t^*}, \\ C_v(t^*), & v \in O_{t^*}. \end{cases} \tag{4.50}$$

随后，程序将该结果导出为：

$$myComData.Dat. \tag{4.51}$$

用于后续计算 NMI、Recall、Precision 和 F-score 等评价指标。

5 当前程序存在的不足与后续改进方向

虽然当前程序相比原始版本在识别效果上有一定提升，程序的逻辑相对完整，但也带来了更高的运行开销和一定程度的冗余设计。

程序整体复杂度较高：当前程序相比原始版本增加了较多计算环节。基础划分选择、图密度判断、不同相似度模式、稀疏图隶属度过滤、密图最大断崖过滤、候选稳定性计算以及最终候选选择等步骤。

时间开销有所增加：当前程序的运行时间明显高于原始版本。根据现有实验结果，在共同 case 的公平对比中，原始版本对于 $N = 1000$ 的网络的平均运行时间约为：

$$T_{original} = 1.410, \tag{5.1}$$

而改进版本的平均运行时间约为：

$$T_{improve} = 3.779. \tag{5.2}$$

基于改进修改的虚拟几何中心版本平均运行时间进一步增加到：

$$T_{center} = 4.419. \tag{5.3}$$

部分设计存在冗余:当前程序为了保证结果稳定，引入了多个质量指标和过滤机制。这些机制从功能上看都有一定作用，但也存在部分重叠。

例如，在稀疏图模式下，程序先使用隶属度过滤弱节点-社区对：

$$m(v, c) \geq 0.05. \tag{5.4}$$

之后，Part 3 又使用候选整体的隶属度质量：

$$Mem(t) = \frac{1}{|O_t|} \sum_{v \in O_t} \min_{c \in C_v(t)} m(v, c). \tag{5.5}$$

类似地，Part 2 中的候选过滤已经会影响重叠节点规模，而 Part 3 中又进一步使用：

$$|O_i| \leq overlapSizeUpper \tag{5.6}$$

限制候选规模。该设计能够防止低阈值下重叠节点数量膨胀，但也可能使筛选逻辑变得较复杂。

阈值扫描仍然带有经验性:当前程序通过阈值扫描生成候选集合：

$$t = bottomThreshold,\ bottomThreshold + 0.01, \ldots, topThreshold. \tag{5.7}$$

这种方法相比单一阈值更加稳健，但仍然依赖扫描范围和步长设置。步长过大时，可能跳过较优阈值；步长过小时，候选数量增加，运行时间也会增加。

候选数量增加后，Part 2 需要生成更多候选结果，Part 3 也需要比较更多候选之间的稳定性。

后续可以考虑引入自适应阈值扫描。例如，先使用较大步长进行粗扫描，再在候选质量较高的区间内进行细扫描。这样可以在保证候选覆盖能力的同时减少不必要的计算。

几何中心版本未取得明显效果:基于改进版本修改的虚拟几何中心版本希望通过社区内节点向量的平均值表示社区整体特征：

$$g_C = \frac{1}{|C|} \sum_{v \in C} x_v. \tag{5.8}$$

从理论上看，虚拟几何中心能够综合社区内所有节点的信息，避免单个真实代表节点对社区表达不充分的问题。但从现有实验结果看，该版本并没有明显超过改进版本真实代表节点版本。

在共同 case 的公平对比中，改进版本的平均 NMI 为：

$$NMI_{improve} = 0.865669, \tag{5.9}$$

而虚拟几何中心版本的平均 NMI 为：

$$NMI_{center} = 0.863890. \tag{5.10}$$

在 F-score 上，改进版本为：

$$F_{improve} = 0.915664, \tag{5.11}$$

虚拟几何中心版本为：

$$F_{center} = 0.903561. \tag{5.12}$$

当前几何中心版本虽然改变了社区表示方式，但并没有带来稳定提升。可能原因是简单平均会平滑掉社区内部的局部结构差异。对于存在多个局部中心的社区，平均向量 g_C 可能落在一个并不具有实际代表意义的位置。此时它虽然表示了整体均值，但未必

能更好地区分边界节点和真正的重叠节点。

密图模式下使用节点到社区内其他节点的平均边权：

$$sim(v, C) = \frac{1}{|C| - 1} \sum_{u \in C, u \neq v} w(v, u) \tag{5.13}$$

也可能受到社区规模影响。较大社区和较小社区之间的平均值分布可能不同，如果不进一步归一化，容易导致不同社区之间的相似度不完全可比。

基础社区选择依赖真实社区数量: 当前程序在选择基础社区划分时，会参考真实社区数量 K^*。程序优先选择社区数量与真实社区数量一致的划分：

$$K_i = K^*. \tag{5.14}$$

如果不存在完全一致的划分，则选择社区数量最接近真实社区数量的划分：

$$i^* = \arg\min_i |K_i - K^*|. \tag{5.15}$$

后续可以使用模块度、社区密度、平均内部连接强度或不同算法结果之间的一致性来选择基础划分。

6 Python 实验工具设计与实现

在 C++ 主程序之外，本项目还编写了三个 Python 实验工具，`eval_current.py` 用于批量运行 C++ 程序并统计评价指标；`run_oracle_gap.py` 用于比较当前 Part 3 选择结果与候选集合中最优结果之间的差距；`nmi_networks.py` 用于对单个网络的真实社区和预测社区进行可视化对比。三者共同构成了当前项目的实验分析部分。

6.1 `eval_current.py`：批量运行与指标统计

`eval_current.py` 自动寻找数据集中的网络文件和真实社区文件，逐个调用 C++ 可执行程序运行社区发现算法，然后读取输出结果并计算 NMI、Recall、Precision 和 F-score 等评价指标。

6.1.1 重叠节点指标计算

`eval_current.py` 中的 `eval_overlap()` 函数用于计算 Recall、Precision 和 F-score。它首先通过 `read_com_file()` 读取社区文件，并构造节点到社区集合的映射：

$$v \to T_v. \tag{6.1}$$

对于预测结果，可以得到：

$$v \to \hat{T}_v. \tag{6.2}$$

如果一个节点属于两个或两个以上社区，则被认为是重叠节点。预测重叠节点集合为：

$$O_{pred} = \{v \mid |\hat{T}_v| \geq 2\}. \tag{6.3}$$

真实重叠节点集合为：

$$O_{true} = \{v \mid |T_v| \geq 2\}. \tag{6.4}$$

正确识别出的重叠节点集合为：

$$O_{correct} = O_{pred} \cap O_{true}. \tag{6.5}$$

因此，Recall 定义为：

$$Recall = \frac{|O_{correct}|}{|O_{true}|}. \tag{6.6}$$

Precision 定义为：

$$Precision = \frac{|O_{correct}|}{|O_{pred}|}. \tag{6.7}$$

F-score 定义为：

$$F\text{-}score = \frac{2 \times Precision \times Recall}{Precision + Recall}. \tag{6.8}$$

如果 $Precision + Recall = 0$，脚本将 F-score 设置为 0，避免除零错误。

6.1.2 NMI_max 计算

`eval_current.py` 中还实现了面向重叠社区的 NMI_{max} 计算。它通过 `read_cover_file()` 读取社区文件，每一行表示一个社区，得到预测社区集合：

$$X = \{X_1, X_2, \ldots, X_p\}, \tag{6.9}$$

以及真实社区集合：

$$Y = \{Y_1, Y_2, \ldots, Y_q\}. \tag{6.10}$$

对于任意一个社区 A，脚本将其看作一个二值变量。设节点总数为 n，则社区 A 的二值熵为：

$$H(A) = -|A| \log_2 \frac{|A|}{n} - (n - |A|) \log_2 \frac{n - |A|}{n}. \tag{6.11}$$

程序中用函数 `h_term()` 表示单项：

$$h(x) = \begin{cases} -x \log_2 \frac{x}{n}, & x > 0, \\ 0, & x = 0. \end{cases} \tag{6.12}$$

因此：

$$H(A) = h(|A|) + h(n - |A|). \tag{6.13}$$

预测社区集合的总熵为：

$$H(X) = \sum_{i=1}^{p} H(X_i), \tag{6.14}$$

真实社区集合的总熵为：

$$H(Y) = \sum_{j=1}^{q} H(Y_j). \tag{6.15}$$

为了计算两个社区 X_i 和 Y_j 之间的条件熵，脚本统计四类节点数量：

$$n_{11} = |X_i \cap Y_j|, \tag{6.16}$$

$$n_{10} = |X_i - Y_j|, \tag{6.17}$$

$$n_{01} = |Y_j - X_i|, \tag{6.18}$$

$$n_{00} = n - n_{11} - n_{10} - n_{01}. \tag{6.19}$$

联合熵为：

$$H(X_i, Y_j) = h(n_{11}) + h(n_{10}) + h(n_{01}) + h(n_{00}). \tag{6.20}$$

一般情况下，条件熵为：

$$H(X_i|Y_j) = H(X_i, Y_j) - H(Y_j). \tag{6.21}$$

但程序中还加入了一个判断条件。如果：

$$h(n_{00}) + h(n_{11}) \leq h(n_{01}) + h(n_{10}), \tag{6.22}$$

则认为 Y_j 对 X_i 的解释效果不好，此时直接令：

$$H(X_i|Y_j) = H(X_i). \tag{6.23}$$

否则使用：

$$H(X_i|Y_j) = H(X_i, Y_j) - H(Y_j). \tag{6.24}$$

对于预测社区集合 X 中的每个社区 X_i，脚本会在真实社区集合 Y 中寻找条件熵最小的社区：

$$\min_j H(X_i|Y_j). \tag{6.25}$$

然后求和得到：

$$H(X|Y) = \sum_{i=1}^{p} \min_j H(X_i|Y_j). \tag{6.26}$$

同理，反向计算：

$$H(Y|X) = \sum_{j=1}^{q} \min_i H(Y_j|X_i). \tag{6.27}$$

互信息定义为：

$$I(X,Y) = \frac{1}{2}\left[H(X) + H(Y) - H(X|Y) - H(Y|X)\right]. \tag{6.28}$$

最终，脚本使用较大的熵进行归一化：

$$NMI_{max}(X,Y) = \frac{I(X,Y)}{\max(H(X), H(Y))}. \tag{6.29}$$

NMI_{max} 关注预测社区结构和真实社区结构的整体一致性。

6.2 `run_oracle_gap.py`：候选最优差距分析

`run_oracle_gap.py` 是用于分析 Part 3 选择策略的脚本。它的重点不是重新评价一个最终结果，而是比较“当前程序选中的候选”和“候选集合中 NMI 最高的候选”之间的差距。

正常程序中，Part 2 会生成一组候选：

$$\mathcal{C} = \{Candidate(t_1), Candidate(t_2), \ldots, Candidate(t_k)\}. \tag{6.30}$$

Part 3 从中选择一个最终候选：

$$Candidate(t^*) = Part3(\mathcal{C}). \tag{6.31}$$

该候选对应的 NMI 记为：

$$NMI_{selected}. \tag{6.32}$$

但是，Part 2 生成的候选集合中可能存在另一个候选，其 NMI 比 Part 3 选中的结果更高。为了判断这种情况是否存在，需要让程序导出所有候选结果，并逐个计算它们的 NMI。

候选集合中的最优 NMI 定义为：

$$NMI_{oracle} = \max_i NMI(Candidate(t_i)). \tag{6.33}$$

在得到 $NMI_{selected}$ 和 NMI_{oracle} 后，脚本计算二者的差值：

$$NMI_{gap} = NMI_{oracle} - NMI_{selected}. \tag{6.34}$$

如果 NMI_{gap} 很大，说明 Part 2 的候选集合中已经存在较好的结果，但 Part 3 没有选中它。此时问题主要出在 Part 3 的候选选择策略上。

如果 NMI_{gap} 很小，说明 Part 3 选中的结果已经接近候选集合中的最优结果。此时如果整体 NMI 仍然不高，问题就更可能出在 Part 1 或 Part 2，例如基础社区划分、代表节点构造、相似度计算或候选生成阶段。

因此，`run_oracle_gap.py` 的核心作用是定位问题：

$$NMI_{gap}{}' \Rightarrow \text{¸}HPart\ 3, \tag{6.35}$$

$$NMI_{gap}\ \text{ffl}FtSNMIN \Rightarrow \text{¸}HPart\ 1/Part\ 2. \tag{6.36}$$

6.3 `nmi_networks.py`：单网络可视化分析

`nmi_networks.py` 是用于观察单个网络社区发现结果的可视化脚本。

6.3.1 单网络 NMI 计算

`nmi_networks.py` 也实现了与 `eval_current.py` 相同的 NMI_{max} 计算逻辑。它读取预测社区和真实社区后，调用：

$$compute\_nmi\_max() \tag{6.37}$$

计算当前单个网络的 NMI：

$$NMI_{max}(\hat{C}, C). \tag{6.38}$$

这一步的作用是先给出整体数值评价，再进入可视化分析。

6.3.2 预测社区与真实社区匹配

为了让预测图和真实图的位置尽量对应，脚本会先将每个预测社区匹配到一个最相似的真实社区。匹配依据是 Jaccard 相似度。

对于预测社区 $\hat{C}_i$ 和真实社区 C_j，Jaccard 相似度定义为：

$$J(\hat{C}_i, C_j) = \frac{|\hat{C}_i \cap C_j|}{|\hat{C}_i \cup C_j|}. \tag{6.39}$$

对于每个预测社区 $\hat{C}_i$，脚本选择 Jaccard 相似度最高的真实社区作为它的对应社区：

$$match(i) = \arg\max_j J(\hat{C}_i, C_j). \tag{6.40}$$

该过程由 `match_pred_to_true()` 完成。它的作用不是评价最终指标，而是为了绘图时把预测社区放到与其最接近的真实社区附近，使两张图更容易对比。

6.3.3 重叠社区的图表示方法

普通网络图中，一个节点通常只画一次。但在重叠社区中，一个节点可能属于多个社区。如果仍然只画一个节点，就很难直观看出它的多社区归属。

因此，`nmi_networks.py` 使用“节点副本”的方法表示重叠社区。对于节点 v，如果它属于社区 c，则在图中建立一个节点副本：

$$(v, c). \tag{6.41}$$

如果节点 v 同时属于多个社区，例如：

$$T_v = \{1, 3\}, \tag{6.42}$$

则图中会出现两个副本：

$$(v, 1), \quad (v, 3). \tag{6.43}$$

脚本通过 `build_cover_graph()` 构造这种节点副本图。对于同一个真实节点的多个副本，脚本会用重叠边连接它们。该重叠边表示：这些副本对应同一个原始节点，只是属于不同社区。

这种表示方式适合观察重叠社区，因为它把“节点属于多个社区”直接转化成了“同一个节点的多个副本之间存在连接”。

6.3.4 布局方式

脚本使用两级圆形布局。

第一层是社区级布局。所有社区中心被放置在一个大圆上。对于第 c 个社区，其中心位置可以理解为：

$$(cx_c, cy_c). \tag{6.44}$$

第二层是社区内部节点布局。每个社区内部的节点副本围绕该社区中心排列在一个小圆上。这样可以使不同社区之间在图上分开，同时又能看出每个社区内部有哪些节点。

真实社区图由 `build_true_layout()` 生成。预测社区图则由 `build_pred_layout()` 生成。预测社区会先通过 Jaccard 匹配到真实社区，然后放置在对应真实社区中心附近。这样预测图和真实图在空间位置上具有可比性。

6.3.5 错误标记

判断某个预测节点副本是否正确，可以用节点-社区归属关系表示。预测归属集合为：

$$\hat{M} = \{(v, c) \mid v \in \hat{C}_c\}. \tag{6.45}$$

真实归属集合为：

$$M = \{(v, c) \mid v \in C_c\}. \tag{6.46}$$

如果：

$$(v, c) \in \hat{M} \cap M, \tag{6.47}$$

则该预测归属是正确的。

如果：

$$(v, c) \in \hat{M} - M, \tag{6.48}$$

则该预测归属是错误的，被标记为红色。

如果：

$$(v, c) \in M - \hat{M}, \tag{6.49}$$

则说明该真实归属被漏检，被打上×。

图片可以显示错误集中在哪些社区、哪些节点被误分配、哪些真实重叠关系没有识别出来。