基于 FORTUNATO 性能度量的社区检测

Srushti Thakar
School of Engineering and Applied Science,
Ahmedabad University
srushti.t@ahduni.edu.in

Amit A. Nanavati
School of Engineering and Applied Science,
Ahmedabad University
amit.nanavati@ahduni.edu.in

ABSTRACT

在他的关于社区检测的论文 [1] 中，Fortunato 引入了一个称为性能的质量函数来评估图划分的好坏。这个度量计算正确“解释”的顶点对的数量，即属于同一个社区并通过边连接的两个顶点，或者属于不同社区且不通过边连接的两个顶点。在这篇论文中，我们探索了 Fortunato 的性能度量（浮点数度量）在检测未加权、无向网络中的社区的应用。首先，我们提供了一个贪婪算法泛化贪婪算法，该算法尝试通过迭代地在两层进行工作来优化小数点度量，即顶点层面和社区层面。在顶点层面，一个顶点仅在小数点后第一位值提高时加入一个社区。一旦完成此操作，就得到了一组初始的社区。在下一阶段，两个社区仅在浮点数度量提高时才会合并。一旦没有进一步改进的可能性，算法会切换回顶点层面，依此类推。峰贪婪在没有任何社区发生变化时终止。然后我们提出了一种更快的启发式算法快速 *Fp*，更适合在大型数据集上运行。我们在几个知名的数据集上展示了这些社区的质量以及计算它们所需的时间。对于一些大型数据集，如 *YouTube* 和活跃日志，我们发现算法快速浮点运算在时间和所获得解决方案质量方面都表现出色。

***K*eywords** 社会网络分析 · 社区检测 · 模块度

1 介绍

网络中的社区检测一直是一个重要问题，其应用范围从社交和通信网络到生物系统。在他们最近的论文“网络社区检测 20 年”中，作者指出分析网络数据时最基本的技术挑战是自动发现社区 [2]。一个普遍接受的原则是，社区内部的链接应该相对频繁，而社区之间的链接则应相对稀少 [3]。

为了区分网络划分为社区的“好”与“坏”划分，拥有一个量化标准来评估图划分的好坏将会很有用。质量函数是将一个数分配给图 [1] 每个划分的函数。模块度 [4] 就是这样一个质量函数。它基于随机图不期望具有聚类结构这一想法，因此需要一个合适的零模型来进行计算。除了作为一个质量函数外，模块度优化本身就是一种流行的社区检测方法，并且已被广泛研究 [5, 6, 7, 8] 。

同一论文 [1] 介绍了另一个质量函数性能（此后称为浮点数，即“Fortunato 的性能”），该函数计算正确“解释”的顶点对的数量，即属于同一个社区并通过边连接的两个顶点，或者属于不同社区且不通过边连接的两个顶点。对于划分 $\mathcal{P}$，浮点数度量计算如下：

$$fp(\mathcal{P}) = \frac{|\{(i,j) \in E, C_i = C_j\}| + |\{(i,j) \notin E, C_i \neq C_j\}|}{n(n-1)/2} \tag{1}$$

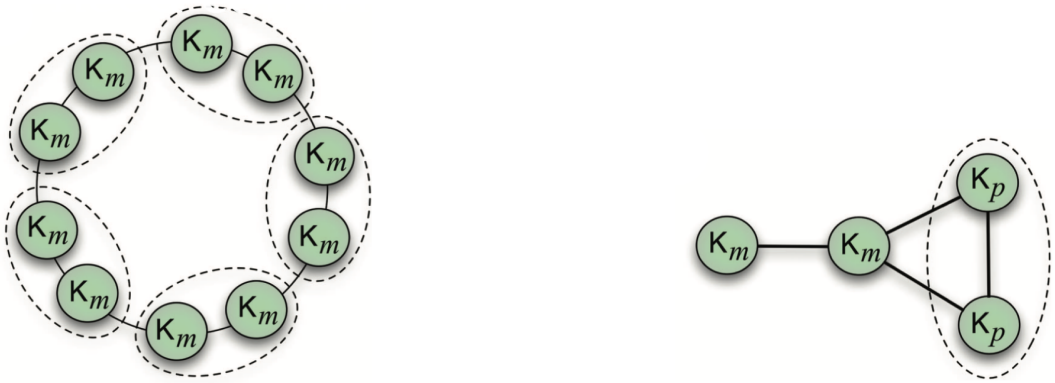
$0 \leq fp(\mathcal{P}) \leq 1$. 这是一个众所周知的质量度量，在诸如 Networkx [9] 等库中用于衡量网络划分的“优劣”。浮点数度量明确捕捉了这样一个直观的概念：社区内部边的数量越多，社区之间边的数量越少，则越好。它不考虑基于随机图的空模型。尽管模块度（也是一种质量度量）在社区检测中被广泛研究，但据我们所知且有些令人惊讶的是，浮点数尚未被探索。在本文中，我们探讨了小数点前数字度量作为网络社区检测优化指标的应用。

由于模块化是一个非常流行的指标，它已经被广泛研究，其一些局限性现在也得到了很好的理解。模块化包含一个内在尺度，该尺度取决于网络中总链接数 [8] 。小于这个尺度（分辨率限制）的模块可能无法被解析，即使在它们是通过单一桥梁连接的完全图这种极端情况下也是如此。浮点数测量方法简单、直观，并且不受分辨率限制问题的影响。

我们的贡献。我们讨论了将浮点数度量用作质量指标的适用性，并强调了在社区检测中使用它的优势。我们介绍了两种算法：算法全局贪婪通过贪婪地合并节点到社区来最大化小数点后第一位度量。算法快速浮点数处理使用了一种更快的启发式方法，适用于在大型网络上运行。我们在各种现实数据集上实验了这两种算法，以找到小数点后浮点数值以及获取它们所需的时间。

在第 2 节中，我们讨论了相关工作。第 3 节详细介绍了算法预测贪婪，该算法贪婪地优化顶点级别和社区级别的浮点数度量。我们展示了在一些知名网络上运行算法全局贪婪算法所获得的结果。第 4 节讨论了算法快速 *Fp*，这是一种更快的启发式方法，特别适合大型网络。第 5 节列出了在各种数据集上运行这两种算法的比较。第 6 节讨论了本文的局限性以及未来可能的研究工作。

2 为什么要优化小数点后第一位？



(a)　　(b)

图 1: 两个来自 [8] 的例子表明模块化优化可能导致反直觉的结果。在两幅图中，模块化优化导致了一个分区，在该分区中团被合并成两个或更多个组（用虚线表示）。每种情况下直观且正确的答案是每个团都是一个单独的社区。优化浮点数度量给出了直观的答案（表 1）。

表 1: 分辨率限制情况（a）和（b），其中 m=20，p=5 [8]。与模块化优化不同，浮点数优化产生了预期的模块（社区）数量。

	模块化优化			**浮点优化**		
	浮点数	**模**	**#模块**	**浮点数**	**模**	**#模块**
(a)	0.9706	0.8871	16	0.9973	0.8758	30
(b)	0.9780	0.5426	3	0.9967	0.5416	4

模块化。大多数社区检测算法都集中在 Newman 和 Girvan 提供的模块化优化上 [4]。模块化是一种流行的品质函数，它将实际观测到的内部边密度与在随机零模型下预期的情况进行比较，并定义为：

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \gamma \frac{k_i k_j}{2m} \right) \delta(c_i, c_j)$$

术语 $\frac{k_i k_j}{2m}$ 是在给定它们度数的情况下，在一个随机网络中节点 i 和 j 之间的预期边数。然而，用于社区检测的模块化优化存在分辨率限制，这解释了它为何无法识别出小规模社区，无论这些社区在网络中的结构重要性如何。这一限制由 Fortunato 和 Barthélemy [8] 分析过，并证明模块化倾向于合并较小但有意义的社区。作者提供了两个示例（图 1），来解释这一限制。表 1 展示了模块化优化和浮点数度量优化的结果。后者能够正确识别社区的数量。

普茨模型变体。为了解决分辨率限制问题，[10] 的作者确定了一个可调参数 γ。通过调整 γ，检测更小社区成为可能。在 [3] 中引入的常数波茨模型（CPM）试图最大化内部边的数量，同时保持相对较小的社区规模。参数 γ 平衡这两个需求。事实上，参数 γ 作为内部和外部边密度阈值。通过设置 γ，可以控制可检测的社区类型。另一方面，浮点数度量不需要任何参数。其公式自然防止了诸如将所有节点合并为单个社区这样的平凡解，这是一个与其他一些质量指标（如覆盖率 [1]）相关的问题。

标签传播。标签传播方法 [11]，计算速度快且高效。社区检测迭代地根据多数局部投票为节点分配标签。尽管它们具有高速度和可扩展性，这些方法往往缺乏稳定性，因为它们可能会基于初始化或破局策略导致不同的分区结果 [12]。

3 优化社区检测中的浮点数度量

我们现在呈现算法泛化贪婪算法，该算法将图作为输入并输出社区。其思路是首先以一种增加浮点数值的方式贪婪地合并节点，当不再可能这样做时，则贪婪地合并社区。然后回到节点。当两个层级都没有变化时，算法终止。

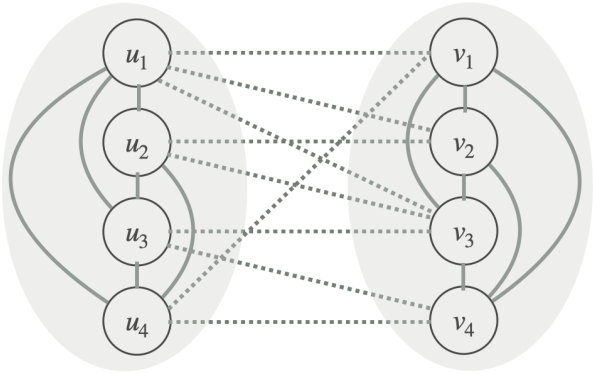


图 2: 在所有节点级别合并完成后尝试社区合并的原因。两个社区的浮点数值为 $\frac{(6+6)+7}{28} = \frac{19}{28}$，而合并版本的值为 $\frac{6+6+9}{28} = \frac{21}{28}$。将任何一个顶点从一个社区移动到另一个社区都会导致浮点数值减少。

它首先将每个顶点分配到自己的社区（第 2 行）。然后，通过检查将 u 分配给其相邻社区后的浮点数指标改进情况来尝试找到每个节点 u 应该移动到的最佳社区，这些相邻社区最初是单个顶点（第 8 行–第 27 行）。如果找到了这样的最佳社区，则将 u 移动到那里（第 23 行–第 26 行）。更改标志被用来跟踪社区结构的任何变化，即社区集合的变化。当没有变化时，更改等于零并且算法退出循环并终止。在完成所有节点级别的合并后，算法试图合并整个社区（第 32 行–第 49 行）。与节点级别的情况一样，选择最佳的要合并的社区（如果存在的话）（第 37 行–第 43 行），然后合并这些社区（第 44 行–第 48 行）。

Algorithm 全部贪婪策略

Input: 图 $G = (V, E)$
Output: 一组社区 $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$

// 初始化每个节点 $u \in V$ 为其自身的社区
$\mathcal{C} \leftarrow \{u \in V : \{u\}\}$
$change \leftarrow 1$
while true **do**
    if $change == 1$ **then**
        $change \leftarrow 0$
        // Node-level merging
        for all node $u \in V$ **do**
            $C_u \leftarrow$ 当前社区为 u
            // store *fp* of current communities
            $best\_gain \leftarrow \mathrm{fp}(\mathcal{C})$
            $best\_comm \leftarrow C_u$
            $NeighComms \leftarrow$ 社区包含
              邻近的 u
            for all community $C_v \in NeighComms$ **do**
                // performance if u is moved from C_u to C_v
                // let the modified set of communities be $\mathcal{C}'$
                $gain \leftarrow \mathrm{fp}(\mathcal{C}')$
                if $gain > best\_gain$ **then**
                    $best\_gain \leftarrow gain$
                    $best\_comm \leftarrow C_v$
                end if
            end for
            if $best\_comm \neq C_u$ **then**
                将 u 移动到 $best\_comm$
                $change \leftarrow 1$
            end if
        end for
    end if
    if $change == 1$ **then**
        $change \leftarrow 0$
        // Community-level merging
        for all community $C_i \in \mathcal{C}$ **do**
            // store *fp* of current communities
            $best\_gain \leftarrow$ 小数点后第一位 $(\mathcal{C})$
            $best\_comm \leftarrow C_i$
            $NeighComms \leftarrow$ 社区邻近于 C_i
            for all community $C_j \in NeighComms$ **do**
                $gain \leftarrow$ 性能如果 C_i 和 C_j 合并
                if $gain > best\_gain$ **then**
                    $best\_gain \leftarrow gain$
                    $best\_comm \leftarrow C_j$
                end if
            end for
            if $best\_comm \neq C_i$ **then**
                合并 C_i 和 $best\_comm$
                更新 $\mathcal{C}$
                $change \leftarrow 1$

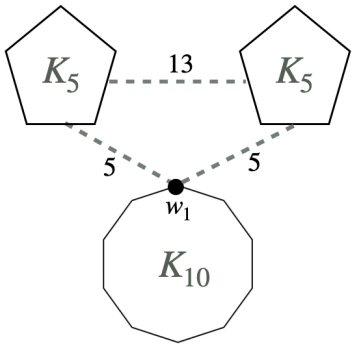


图 3: 在社区合并完成后尝试节点级合并的原因。节点 w_1 与图中的每个顶点相连：它是大小为 10 的团的一部分（K_{10}）每个都有 5 个邻居，分别位于另外两个社区中。这两个 K_5 社区之间有 13 条边（最多可能值为 25）。在两个 K_5 社区合并后，w_1 在新合并的社区中有 10 个邻居（相比它目前社区中的 9 个）。当 w_1 移动到上述合并的社区时，小数点表示法的值从 0.884 增加到 0.889。在两个 K_5 社区合并之前，将 w_1 移动到任意一个 K_5 社区都会减少浮点数值。

图 2 展示了当所有节点级合并完成后，社区合并如何改善小数点值的一个示例。节点 u_1 在社区内有 3 个邻居，在另一个社区也有 3 个邻居，因此它没有动机切换到另一个社区。对于 v_3 也是如此。其余每个顶点在其社区内的邻居都比社区外多。两个社区的小数点后第一位值为 $\frac{(6+6)+7}{28} = \frac{19}{28}$，合并后的版本为 $\frac{6+6+9}{28} = \frac{21}{28}$。

图 3 显示了一个示例，说明了在完成所有社区级别的合并后，节点级别合并如何能够提高小数点的值。节点 w_2 每个其他两个社区有 2 个邻居。在上述两个社区合并之后，w_2 在合并后的社区中有 4 个邻居（它在其当前社区中只有 3 个）。因此，它有动力切换到新合并的社区。

fpGreed 的次优性算法在 u 与 v 合并早于 v 与 w 合并时做出次优选择（如果先合并后者能得到更好的浮点数），因此上述算法的随机化版本可能会给出更好的答案。如果 C_i 与 C_j 合并发生在 C_j 与 C_k 合并之前，可能会出现同样的问题，这可能导致更好的小数点后部分值。另一种次优性的来源如图 4 所示。考虑所有顶点组合进行合并的成本是极其高昂的。

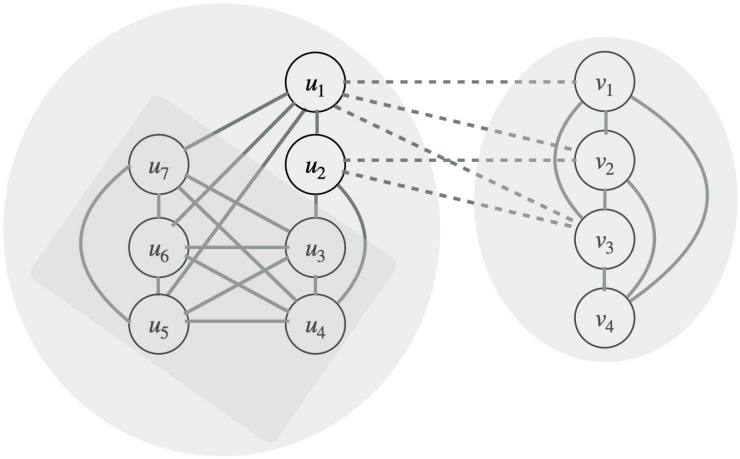


图 4: 此示例展示了算法全局贪婪为何无法保证得到最优解。在上述示例中，通过考虑节点级别的移动和社区合并，无法改进浮点数的值。然而，如果我们把 u_1 和 u_2 一起移入包含 v_1, v_2, v_3 和 v_4 的社区，则可以改进小数点后第一位的值。

表 2 展示了在一些知名数据集上运行自由贪心算法的结果。这种方法也能够识别较小的社区。

表 2: 真实世界数据集中的社区结构

数据集	**性能**	**模块数量**
Karate Club Network	0.9090	19
Dolphin Network	0.9476	28
Florentine Families	0.8952	7
Les Misérables	0.9638	34
Football Network	0.9583	16

4 更快的启发式算法

由于算法峰贪婪不适用于大规模图，我们提出了一种更快的启发式方法，如算法快速浮点运算所示。快速浮点数处理从给定的图构造一个加权图，在该图中，边上的权重表示端点之间的共同邻居数量以及这些共同邻居之间的连接数。

表 3: 各种数据集上全局贪婪和快速 *Fp* 的比较

Dataset	Nodes	Edges	全局贪婪算法浮点数	快速 *Fp* 浮点数	全景贪婪 time	快速 *Fp* time
Karate Club	34	78	0.9090	0.6791	100.2 ms	83.7 ms
Dolphin	62	159	0.9476	0.8895	177 ms	90.6 ms
Florentine Families	15	20	0.8952	0.8762	86.6 ms	85.6 ms
Les Misérables	77	254	0.9648	0.8516	221.7 ms	88.1 ms
Football Network	115	613	0.9583	0.8828	560.6 ms	102.7 ms
Email EU Core	1005	16706	0.9759	0.4125	1145.839 ms	1.723 s
Facebook	4039	88234	0.9933	0.9437	33695.973 ms	30.277 s
Arxiv dataset	5242	14496	0.9995	0.9983	31495.057 ms	1.253 s
Wikivote	7115	100762	–	0.7519	–	26.065 s
CA-condmat	23133	93497	–	0.9979	–	19.480 s
Internet	26475	53381	–	0.9678	–	44.009 s
Epinions	75879	405740	–	0.9804	–	336.667 s
Slashdot0902	82168	546487	–	0.9856	–	416.937 s
DBLP	317080	1049866	–	0.9999	–	3935.676 s
Web-nd.edu	325729	1497134	–	0.9987	–	2568.600 s
YouTube	2987624	1134890	–	0.9937	–	41476.909 s
Live Journal	3997962	34681189	–	0.9997	–	573574.157 s

第一步是基于所有节点对的邻域创建加权图 G_2，这些节点可能在输入图 G 中通过边相连也可能不相连（第 2—16 行）。这一步也为一个节点对 u, v 提供了机会，即使它们之间没有边连接，只要它们有多个共同邻居，就可以属于同一个社区。权重取决于 u 和 v 的共同邻居之间的边（第 6—10 行）。下一步是在加权图中创建两个顶点 u 和 v，如果权重超过阈值 t，它们之间可能由一条加权边连接。我们将 $t = 3$（可能的最小值）设置为本文所有实验的值。选择一个更大的值应该会减少运行时间，但以浮点数值为代价。带权重的边可能会在 u 和 v 之间创建，即使它们在 G 中没有通过边相连。步骤 3（第 26 – 33 行）通过考虑按边权重非递减顺序的边来创建初始社区集合。这是为了确保最强的社区首先形成。节点 w 可能是两个（或更多）节点对的共同邻居。这一步确保它加入最强的一对，然后从其他社区的考虑中移除（第 32 行）。下一步（第 35 – 53 行）是在浮点数值改善时合并社区（详情请参见引理 1）。

Algorithm 快速 *Fp*

Input: 图 $G = (V, E)$
Output: 一组社区 $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$

// 步骤 1：计算所有节点对之间的权重
// 基于它们的共同邻居之间的边来计算
for all $(u, v) \in V \times V, u \neq v$ **do**
    $CN \leftarrow$ 共同邻居 u 和 v
    $k \leftarrow |CN|$
    $e \leftarrow 0$
    for all $(w_1, w_2) \in CN \times CN, w_1 \neq w_2$ **do**
        if $(w_1, w_2) \in E$ **then**
            $e \leftarrow e + 1$
        end if
    end for
    if $(u, v) \in E$ **then**
        $w(u, v) \leftarrow 2k + e + 1$
    else
        $w(u, v) \leftarrow 2k + e$
    end if
end for
// 步骤 2: 构建加权图 $G_2 = (V_2, E_2, W_2)$
// 基于权重阈值 t 的
$V_2 \leftarrow V$, $E_2 \leftarrow \emptyset, W_2 \leftarrow \emptyset$
for all $(u, v) \in V \times V, u \neq v$ **do**
    if $w(u, v) \geq t$ **then**
        添加具有权重 $w(u, v)$ 的边 (u, v) 到 E_2
    end if
end for
$G_2 \leftarrow (V_2, E_2, W_2)$
// Step 3: Initial Community Detection
$\mathcal{C} \leftarrow []$
while $E_2 \neq \emptyset$ **do**
    查找具有最高 $w(u, v)$ 的边 (u, v)
    $CN \leftarrow$ 公共邻居 u 和 v
    $Community \leftarrow \{u, v\} \cup CN$
    将 $Community$ 添加到 $\mathcal{C}$
    从 G_2 中移除所有在 $Community$ 中的节点
end while
// Step 4: Merge Communities
$Merged \leftarrow$ **真实**
while $Merged$ **do**
    $Merged \leftarrow$ **假**
    for all pairs $(C_1, C_2) \in \mathcal{C}$ **do**
        $crossEdgeCount \leftarrow 0$
        for all $u \in C_1, v \in C_2$ **do**
            if $(u, v) \in E$ **then**
                $crossEdgeCount \leftarrow crossEdgeCount + 1$
            end if
            // a cross-community edge exists only if they are
            // connected in the original graph
        end for

引理 1. *考虑两个社区 C_1 和 C_2，分别具有 m 和 n 个顶点。令 e_m 和 e_n 分别表示 C_1 和 C_2 内部的边数。如果两个社区之间的交叉边（跨社区边）e_{cc} 数量大于 $\frac{m\times n}{2}$，则合并后的浮点数值更高。合并前，*

$$\begin{aligned} \text{fp(separate)} &= \frac{|e_m + e_n| + |(m \times n) - e_{cc}|}{m \times n} && (2)\\ \text{fp(merged)} &= \frac{|e_m + e_n + e_{cc}|}{m \times n} && (3)\\ \text{if } e_{cc} &> \frac{m \times n}{2}, && (4)\\ \text{fp(merged)} &> \text{fp(separate)}. && (5) \end{aligned}$$

5 实验

表 3 展示了通过运行算法全部贪婪算法和快速 *Fp* 在不同数据集上获得的浮点数值及其所需的时间（秒）。预期地，快速浮点运算比全局贪婪运行得更快，但可能会产生较差的浮点数值。实验是在配备有 12 核 CPU、38 核 GPU、16 核神经引擎、96GB 统一内存和 4TB SSD 存储的 Mac Studio Apple M2 Max 上进行的。算法在 Python [13] 中使用 Networkx [9] 实现。时间计算使用命令行基准测试工具 hyperfine [14]。对于 Algorithm 快速浮点数处理，我们设置阈值参数 $t = 3.$。我们只能在较小的数据集上多次运行这些算法。

对于表中的较大数据集，快速 *Fp* 似乎能产生良好的结果，无论是输出质量还是运行时间方面。

6 讨论与未来工作

优化浮点数度量似乎是基于本文获得的初步结果来寻找社区检测的一个有前景的方向。以下重要问题需要得到解答：

1. 限制：空模型。模块化比较给定模块内部的链接数量与相同大小和相同度序列 [8] 的随机图的预期值。但它受到分辨率限制问题的影响。另一方面，最优的浮点数值将基于网络中社区概念的一致直觉给出最佳可能的社区，即，社区内的边数应尽可能多，而跨社区的边数应尽可能少。并且它是无参数的。但是浮点数度量公式没有考虑到空模型。有没有办法将空模型考虑从我们用于社区检测优化的参数中分离出来？例如，通过优化小数点后第一位将网络划分为社区有意义吗，然后使用这种社区结构来衡量模块化？
2. 限制：未加权图。当前的公式仅适用于无权图。能否将小数点后部分度量推广到有权图？一种可能性是考虑非边的权重为 1，从而得到以下结果：
$$fp_{wt}(\mathcal{P}) = \frac{\sum_{(i,j)\in E, C_i = C_j} w(i,j) + |\{(i,j) \notin E, C_i \neq C_j\}|}{\sum_{i,j\in E} w(i,j) + |\{(i,j) \notin E, C_i \neq C_j|} \quad (6)$$
然而，如果图中的边具有较高的权重，这可能会导致偏向于将所有边添加到单个社区。
3. 未来工作：速度。
(a) 如何进一步加快全局贪婪和快速 *Fp* 的速度是另一个值得关注的问题。例如，可以从引理 1 中获取想法，在全部贪婪算法中减少一些不必要的计算。
(b) 阈值 t 在快速浮点运算中的作用需要进一步探索。目前，它被设置为可能的最小值。提高它的值应该能够提升速度，但可能会降低解决方案的质量。
4. 未来工作：探讨优化小数点后第一位的影响。通过这些方法获得的社区规模分布和社区数量需要进一步探讨。

致谢

本工作部分得到了美国圣何塞思科研究资助。

参考文献

[1] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[2] Santo Fortunato and Mark EJ Newman. 20 years of network community detection. *Nature Physics*, 18(8):848–850, 2022.

[3] Vincent A Traag, Paul Van Dooren, and Yurii Nesterov. Narrow scope for resolution-limit-free community detection. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, 84(1):016114, 2011.

[4] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[5] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, 70(6):066111, 2004.

[6] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[7] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. From louvain to leiden: guaranteeing well-connected communities. *Scientific reports*, 9(1):1–12, 2019.

[8] Santo Fortunato and Marc Barthelemy. Resolution limit in community detection. *Proceedings of the national academy of sciences*, 104(1):36–41, 2007.

[9] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[10] Jussi M Kumpula, Jari Saramäki, Kimmo Kaski, and János Kertész. Limited resolution in complex network community detection with potts model approach. *The European Physical Journal B*, 56(1):41–45, 2007.

[11] Usha Nandini Raghavan, Réka Albert, and Soundar S Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.

[12] Jia Hou Chin and Kuru Ratnavelu. A semi-synchronous label propagation algorithm with constraints for community detection in complex networks. *Scientific Reports*, 7(1):45836, 2017.

[13] Python Software Foundation. *Python Language Reference, version 3.11*, 2023.

[14] David Peter. hyperfine, March 2023.