分类号:_____

西北大学

NORTHWEST UNIVERSITY

本科生毕业论文(设计)

题目: 基于蚁群算法的影响力传播问题研究

作　者　单　位 西北大学

作　者　姓　名 顾文浩

专　业　班　级 中外一班

作　者　学　号 2020117410

指导教师（职称） 杨文静

2024 年 5 月

诚信声明

本人郑重声明：所呈交的学位论文，是本人在指导教师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或没有公开发表的作品内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：　　日期：　　年　　月　　日

摘 要

近年来，随着在线社交网络的广泛普及，它们已经成为社交网络分析的重要研究平台，尤其是在计算社会科学领域。在线社交网络的特点是用户间复杂的互动关系，这些互动关系在网络中形成了各种大小的群体，其中尤为突出的是“最大团”结构。在社交网络分析中，识别这些最大团是至关重要的，因为它们可以帮助研究人员理解网络中的主要社交动态和群体行为。

然而，寻找最大团的任务在计算上是非常挑战性的，尤其是在大规模网络中。为了解决这一问题，本文对一种创新的混合算法进行研究，该算法结合了蚁群优化和粒子群优化算法的特点，通过两者的互补优势，增强了搜索最大团的效率和准确性。具体来说，该算法通过一个改进的信息素更新机制，利用粒子群算法在全局搜索中的高效表现，与蚁群算法在局部搜索和解空间探索中的优势相结合。

我们在几个流行的标准社交网络数据集上进行了算法性能的测试。测试结果显示，与传统的蚁群优化算法相比，本文提出的混合算法在找到最大团的大小和搜索速度上都有显著提升。这一成果不仅展示了混合算法在处理复杂社交网络数据时的有效性，也为未来社交网络分析提供了新的工具，有望在群体行为分析、社交网络营销等领域发挥重要作用。

关键词：社交网络, 最大影响力问题, 蚁群算法, 粒子群算法

Abstract

In recent years, the widespread popularity of online social networks has made them a vital platform for social network analysis, particularly in the field of computational social science. These networks are characterized by complex interactions among users, which form various group sizes, notably the ”largest clique” structures. Identifying these largest cliques is crucial in social network analysis as they help researchers understand the main social dynamics and group behaviors within the network.

However, finding the largest cliques is computationally challenging, especially in large-scale networks. To address this issue, this paper introduces an innovative hybrid algorithm that combines the features of Ant Colony Optimization and Particle Swarm Optimization. By leveraging the complementary strengths of both, the algorithm enhances the efficiency and accuracy of searching for the largest cliques. Specifically, it incorporates an improved pheromone update mechanism that utilizes the efficiency of Particle Swarm Optimization in global searches, combined with the strengths of Ant Colony Optimization in local search and exploration of the solution space.

We tested the performance of our algorithm on several popular standard social network datasets. The results demonstrate that, compared to traditional Ant Colony Optimization algorithms, our proposed hybrid algorithm significantly improves in terms of the size of the largest cliques found and the speed of the search. This achievement not only demonstrates the effectiveness of the hybrid algorithm in handling complex social network data but also provides new tools for future social network analysis, potentially playing a significant role in areas such as group behavior analysis and social network marketing.

Key words: Social network ,Influence maximization problem ,Ant colony optimization(ACO) ,Particle swarm optimization(PSO)

目 录

1 绪论

1.1 研究背景

在近几十年中，社会性的概念在各个科学领域中受到了广泛关注。社会性是生物体参与集体活动或社会互动的一种属性，对于社会的发展至关重要。这种属性表明个体不能脱离社会独立生存。包括人类在内的社会性动物表现出个体间以及群体与个体间复杂的互动关系。在自然界中，这种互动表现为群体行为，可以提高捕食效率或降低被捕食风险。在人类社会中，先进的沟通手段使社会影响力渗透到生活的各个层面，从日常活动如选择音乐或餐馆到更重大的决策。人们的选择常常受到家庭、同事、朋友乃至公众意见的影响。具有重大影响力的个体对其群体可以产生深远的影响。例如，董宇辉通过社交媒体平台推广偏远地区的农产品，为这些地区提供了新的经济机会。他利用个人影响力显著增加了这些产品的可见度和销量，为当地经济作出了巨大的贡献。这个例子证明了社交网络在产品线上推广和消费者选择中的强大作用。深入理解影响力的产生和传播方式对于预测个人和群体行为、指导政府和企业决策也极具价值。例如，对天水麻辣烫、淄博烧烤的官方推广极大地促进了当地经济的发展；同时，“猫一杯”上秦朗法国丢失作业的视频虽然引发了广泛的社会关注，但最终被证实为谣言，这显示了具有大影响力的个体或误信息如何对社会产生巨大的影响，浪费了公众的精力和网络资源。这些例子突出了具有影响力的个体对群体（最大团体）的重大正面或负面影响。选择合适的影响力个体或群体及传播渠道可以帮助有效传播政策和控制舆论与谣言。

社会影响力的研究已经有很长的历史，并为人类行为的新见解做出了贡献。例如，Christakis 和 Fowler 基于健康数据的研究[1][2]，证明了肥胖和吸烟行为都可以在网络上传播。基于研究数据表明，影响力在用户选择和选举投票方面有人也占据主导地位。例如，小红书用复杂的推荐算法来影响用户的媒体选择。通过统计、分析用户过去的浏览消费等行为，通过算法加以预测，精准将用户想看的，或者需要的内容投放给用户，这种方法不仅提高了用户满意度，也增加了广告投放的精准度，为产品带来了经济效益。随着互联网、社交媒体和大数据技术的发展，我们现在能够在更大的规模上的平台上更加深入得了解影响力传播的机制。

在这一丰富的研究基础上，本文聚焦于社交影响力分析的计算挑战，即在社交网络中识别最具影响力的群体，或称为“最大团”。我们的研究通过提出一种新颖的混

合优化算法，提高了识别这些团体的计算效率和准确性，从而为社交影响力的研究领域做出了贡献。这一进展不仅加深了我们对大型网络中社会影响力的理解，还为相关利益相关者提供了有效利用影响力群体的实用工具。

1.2 国内外研究现状

1.2.1 社交网络影响力最大化模型

随着网络的发达，信息和影响力传播问题收到了越来越广泛的关注，相关的传播模型被不断提出。2003 年，Kempe、Kleinberg 和 Tardos[3][4]提出了独立级联模型和线性阈值模型。这些模型已经成为影响力传播的经典模型，在影响力最大化、学习和传播领域内被广泛应用和研究。影响力最大化问题是对病毒式营销的数学模型化，上述模型证明了影响力最大化模型是一个 NP-hard 问题。而 NP-hard 问题优化的重要方向时有效的近似算法。再找不到影响力最大种子集时可以寻找较优解，其中两者的比值便是近似算法的近似比。近似算法的设计核心则来自影响力函数的子模性质和贪心算法的技术[5]。在 Kempe 等人的论文中[3]，提出蒙特卡洛算法来模拟影响力传播，从而估算子集合 S 的延展度 $\sigma(S)$ 的近似解。在这种近似解情况下，贪心算法的解能达到的 $\left(1-\frac{1}{e}-\epsilon\right)$ 近似解，可以获得影响力最大化问题 63% 的近似解。

1.2.2 蚁群算法

蚁群算法是一种启发式优化算法，其灵感来源于蚂蚁之间的协同工作方式。它通过模拟蚂蚁在寻找食物过程中释放信息素和跟随信息素的行为，来解决复杂的组合优化问题。

蚁群算法的特点在于它采用了正反馈机制、分布式计算策略以及贪心启发式搜索。算法的核心思想是利用路径上信息素浓度的变化来引导蚂蚁选择更优的路径。随着时间的推移，越来越多的蚂蚁会选择信息素浓度较高的路径，即更优的路径，最终整个蚁群会找到通往食物源的最短路径。自该算法问世以来，在网络表征学习领域展现出了显著的效果，同时在移动社交和网络聚类等应用场景中也取得了不俗的成效[6][7]。

蚁群算法是由 Marco Dorigo 在九十年代初提出的，已被应用于各种组合优化问题，如旅行推销员问题、车辆路径问题、约束满足问题和二次分配问题[8]。

2003 年，Serge Fenet 和 Christine Solnon[9]首次将蚁群算法应用于寻找极大团，并

提出了 Ant-Clique 算法。所提出的算法利用全局信息素更新策略，能够获得 DIMACS 数据集的部分最优解。2006 年，他们[10]进一步研究了通过分别在节点和边上释放信息素的性能。实验结果表明，这两种策略都适合于寻找极大团。同时，也指出通过在边上释放信息素寻找极大团更有效。

2007 年，Xinshun Xu 等人[7]提出了动态调整 Ant-Clique 算法参数的方法，并将模拟退火与蚁群算法结合。参数的动态调整提高了算法的收敛速度和搜索最优解的能力。模拟退火算法加快了构建团的速度。与 Ant-Clique 相比，解的准确性大大提高，但收敛速度仍需进一步改进。

2010 年，王会英和耿佳丽[11]改进了蚂蚁选择节点的策略，他们将轮盘法和最大概率原则方法相结合来选择候选节点。算法随机选择节点，从而扩大搜索空间，使解多样化，并避免局部最优。但解的多样性可能减缓算法的收敛速度。

2011 年，陈荣[12]改进了信息素更新策略，结合了局部和全局方式。信息素的局部更新在迭代过程中增加解的多样性，从而间接指导后续蚂蚁选择未被选中的节点，并搜索尚未探索的解空间区域。与 Ant-Clique 相比，改进的算法可以获得更高的平均准确性和更快的收敛速度。然而，除了信息素更新策略外，这些方法仍有很大的改进空间。

2012 年，Mohammad soleimani 等人[13]提出了 PSO-ACO 算法，将蚁群算法与粒子群优化算法相结合。粒子群优化算法改进了信息素更新过程，并提高了最优解的准确性。实验在社交网络数据集上进行。然而，这个方法仍然存在陷入局部最优的问题。

2014 年，Mohammad soleimani 等人[14]验证了 2012 年中的方法也可以在 DIMACS 基准数据集上获得较高的准确性。目前，上述算法在不同方面改进了在社交网络中寻找最大团的解决方案，但无法同时获得令人满意的准确性和收敛速度。在整体准确性和收敛速度方面仍有很大的改进空间。

2019 年 Chiman Salavati 等人[15]基于蚁群算法提出了名为 PMACO 和 IMOACO 的新方法。通过去除不重要的节点，消除对其邻居影响程度小于特定阈值的节点，有效地选择有影响力的用户的子集。PMACO 注重利润最大化，而 IMOACO 也考虑减少选定用户之间的相似性以增强覆盖范围。在性能和计算效率方面对现有方法进行了改进。

总之，寻找最大团的蚁群算法主要受制于局部最优和收敛速度。多样性的解有助于防止解陷入局部最优，但同时减慢了收敛速度。

1.3 研究内容

随着社交网络的分析的关注度逐渐增高，越来越多的学者开始对这个领域展开研究。社交网络中的影响力最大化问题主要涉及在网络中识别出具有最大潜在影响力的 k 个用户。这些用户的选择基于特定的传播模型，目的是通过他们启动的信息传播，使得影响力在整个社交网络中尽可能广泛地扩散，从而覆盖最大的用户群体。目前，这一问题的研究已经取得了许多有价值的成果，研究焦点主要集中在传播模型的改进和算法性能的提升两个方面。本文的研究将主要探讨社交网络中的蚁群算法传播模型及积极影响力最大化算法：我们将主要研究蚁群算法和它的性能改进，本文还将介绍另一种粒子群算法来优化蚁群算法。

1.4 文章组织架构

本文的主要内容是基于蚁群算法的影响力传播问题研究，主要将对收敛速度和避免陷入全局最优方面做出改进。正文共分为 5 个章节，具体内容如下。

第一章绪论在绪论部分，我们将首先阐述影响力最大化问题的背景和现实意义。随着社交网络的快速发展，影响力传播已成为一个备受关注的研究领域。接着，我们将对基本传播模型和蚁群算法的国内外研究现状进行综述，以明确本文在现有研究中的位置和创新点。然后，我们将简要介绍本文的研究内容，即如何改进蚁群算法以提高其在影响力最大化问题中的性能。最后，我们将概述文章的组织架构，为读者提供一个清晰的阅读指南。

第二章相关知识理论在第二章中，我们将首先介绍社交网络和影响力问题的定义。通过明确这些基本概念，我们将为后续的研究奠定坚实的理论基础。接着，我们将阐述影响力最大化模型和影响力最大化算法的基本原理。这将有助于读者理解影响力最大化问题的本质和求解方法。最后，我们将对蚁群算法进行基础介绍，包括其原理、特点和应用领域。这将为后续章节中蚁群算法的改进和应用提供必要的背景知识。

第三章基于粒子群算法的蚁群算法相结合的 ACO-PSO-IM 算法在第三章中，我们将详细介绍本文的核心研究内容——基于粒子群算法的蚁群算法相结合的 ACO-PSO-IM 算法。首先，我们将介绍粒子群优化算法的基础知识，包括其原理、算法流程和特点。然后，我们将阐述粒子群算法如何对蚁群算法进行优化，以提高其收敛速度和避免陷入全局最优解的能力。最后，我们将给出具体的算法设计，包括算法流程、

伪代码和参数设置等。这将为读者提供一个全面的了解本文所提算法的机会。

第四章实验在第四章中，我们将通过实验来验证本文所提算法的有效性。首先，我们将介绍实验前的具体准备工作，包括。这些准备工作将确保实验的可靠性和有效性。然后，我们实验所用到的环境、数据集以及来源、实验参数设计等将给出具体的实验数据和相关的总结。通过对比不同算法在相同数据集上的性能表现，我们将证明本文所提算法在收敛速度和避免陷入全局最优解方面的优势。

总结与展望在总结与展望部分，我们将对本文的研究内容进行简要概述，并总结本文的主要贡献和创新点。然后，我们将探讨进一步的研究思路和方向，以推动影响力最大化问题的深入研究和发展。通过展望未来的研究方向，我们将为读者提供一个新的视角和思考空间。

2 相关知识理论

本章将深入探讨社交网络中的影响力最大化问题及其相关理论。首先，我们将对社交网络的结构和特性进行详细描述。接下来，将定义影响力最大化问题，并阐述其在社交网络分析中的重要性。最后，本章将介绍一些广泛使用的影响传播模型以及几种经典的影响力最大化算法，这些算法旨在优化信息传播的范围和效果。

2.1 社交网络概述

社交网络最早由 Barnes[16]在 1954 年提出，社交网络是由个体通过网络联系在一起的网状社会系统，个体之间存在各种社会关系，如友谊、合作、交流等[17]。社交网络通常用有向图表示 $G = (V, E)$，其中 V 是节点的集合，$E \subseteq V \times V$ 是有向边的集合。每一个节点 $v \in V$ 代表一个社交网络中的个体，每一条边 $(u, v) \in E$ 代表节点 u 到节点 v 的影响力关系，但节点 v 对节点 u 可能没有影响力。这是因为有向的边表明了影响力的方向性。节点 u 对节点 v 有影响力在数学建模中通常会加上额外的权重来使模型更接近现实。[18]

定义 2.1.1 (节点的度). 对于一条有向边 $(u, v) \in E$，它叫做节点 u 的出边，节点 v 的入边。在无向图中，一个节点的度（记为 $d(v)$）是简单地计算与该节点相连的边的数量。公式可以表示为：$d(v) =$ 边数连接到节点v 。在有向图中，我们通常区分入度和出度：

- 入度（记为 $d_{in}(v)$ ）是指向节点 v 的边的数量。$d_{in}(v) =$ 边数指向节点v
- 出度（记为 $d_{out}(v)$ ）是从节点 v 出发指向其他节点的边的数量。
  $d_{out}(v) =$ 边数从节点v 出发

节点的总度数即为节点出度和节点入度的加和。节点的度数是网络中节点重要程度的直接指标。节点的度值越高，说明它在网络中的重要性越大。

定义 2.1.2 (邻居). 节点 u 的邻居 $N(u)$ 被定义为一组用户 v，满足 $v \in N(u)$ 当且仅当 $\exists (u, v) \in E$ 且 $v \in V$。$N^-(v)$ 和 $N^+(v)$ 分别表示节点 u 的入邻居和出邻居。

2.2 影响力最大化问题

影响传播建模的一个重要目标是控制和优化信息的传播方式。在这方面，影响力最大化问题是一个核心研究领域，它广泛探讨如何在网络中选择一组节点，以最大程度地扩散信息。

在社交网络中，用户的影响力指的是当某个用户采取某种行为时，其他用户是否也会受到该用户的影响而采取相同的行为。例如，当用户 A 转发一条微博时，如果其好友用户 B 也因为用户 A 的转发而转发了相同的微博，则说明用户 A 具有影响力。在影响力最大化问题中，网络中的每个节点都可以处于激活状态或非激活状态。当节点 A 处于激活状态时，它以概率 p_{AB} 影响其邻居节点 B。如果影响成功，节点 B 将从非激活状态转变为激活状态。节点 A 成功影响的节点数量反映了其影响力。

影响力最大化是指，在给定社交网络图 $G = (V, E)$ 中，在影响力传播模型下，找到一个节点数为 k 的种子节点集 S^*，使得以 S^* 为种子节点启动的影响力扩展范围最大。即 $S^* = \arg\max_{S \subseteq V, |S|=k} \sigma(S)$。

定义 2.2.1 (种子节点). 种子节点 S 是在社交网络中充当信息传播源的一组节点，其中 $|S| = k$，$S \subseteq V$。

定义 2.2.2 (活跃节点). 节点在图中被描述为活跃 *(active)* 和不活跃 *(inactive)* 两种状态，不活跃状态代表节点暂时未收到活跃状态节点激活信息。如果 $u \in S$ 或者 u 收到了之前活跃的节点 $v \in V_A$ 在传播模型 M 中传播的信息，则称节点 $u \in V$ 称为活跃的。一旦节点 u 被激活，则 $V_A \leftarrow V_A \cup \{u\}$。

定义 2.2.3 (影响力扩散). 种子集 S 的影响力扩散 $I_S(S)$ 定义为在某扩散模型下，扩散过程后活跃用户的数量，即 $I_S(S) = |V_A(S)|$。

2.3 影响力传播模型

信息在网络中的传播方式通常是未知的。理解庞大信息背后的传播机制对于多种应用领域非常重要，例如病毒式营销[19-20]、社会行为预测[21-23]、社交推荐[24-26]以及社区发现[27-29]。他们提出了不同种类的信息传播模型来描述和模拟这一过程，其中最著名的就是独立级联（IC）模型和线性阈值（LT）模型。IC 模型侧重于社交网络中朋友之间的个人（独立）互动和影响力。LT 模型关注影响力传播中的阈值行为，我们经常会涉及到这一点——当足够多的朋友购买电子产品、玩新电脑游戏或使用新的

在社交软件时，我们可能会转变为遵循相同的行为行动[30]。假设信息从一个种子节点集开始传播，其他节点只能通过其邻节点获取信息。在独立级联模型的每一次传播循环中，社交网络中的每个节点都被明确地界定为两种状态之一：活跃状态或不活跃状态。尤为关键的是，一旦某个节点被激活至活跃状态，它将保持这种状态，无法再次转变为不活跃状态。这一特性确保了模型在模拟社交网络中的信息传播时，能够精确地反映节点状态的稳定性和持久性，从而为我们提供了对实际传播过程更为贴近的模拟和分析。

2.3.1 独立级连模型

在探讨社交网络中的影响力传播机制时，独立级联模型为我们提供了一种独特的视角，它是对网络中人与人之间独立交互影响行为的精炼抽象。这一模型的核心在于，它利用边上的概率来量化个体间的影响力传递。许多常见的传播现象，由于它们所展现的独立性特质，与独立级联模型的假设相契合。值得一提的是，独立级联模型的有效性已经得到了基于实际数据的影响力学习研究的验证，这使其在众多影响力传播模型中脱颖而出，成为当前领域内最为广泛研究和深入探讨的模型之一。通过独立级联模型，我们能够更深入地理解社交网络中的影响力传播机制，为实际应用提供有价值的理论支撑。下面是独立级连模型的传播过程和伪代码。

Algorithm 1 独立级连模型[31]

Input: 图 $G_{IC} = (V, E, p)$，种子集 $\phi_0 \subseteq V$.
Output: 最终影响集合 Φ^*.

Let $\Phi = \phi_0, \bar{\Phi} = V \setminus \phi_0$
Let $k = 0$;
while $\phi_k \neq \emptyset$ **do**
    Let $k = k + 1, \phi_k = \emptyset$;
    for $j \in \bar{\Phi}$ **do**
        for $i \in N_j \cap \phi_{k-1}$ **do**
            if rand$(0) \leq p_{ij}$ **then**
                $\phi_k = \phi_k \cup \{j\}$;
            end if
        end for
    end for
    $\Phi = \Phi \cup \phi_k$
    $\bar{\Phi} = \bar{\Phi} \setminus \phi_k$
end while
Let $\Phi^* = \Phi$.
return Φ^*

初始激活

1. 初始设定：在时间 $t = 0$ 时刻，选择一个预先确定的种子节点集合 S_0，这些节点被标记为激活状态。所有不在 S_0 中的节点处于未激活状态。

动态传播过程

2. 激活尝试：在每一个离散的时间点 $t \geq 1$，每个在上一时间点 $t-1$ 新被激活的节点 u，即 $u \in S_{t-1} \setminus S_{t-2}$（这里 $S_{-1} = \emptyset$ 表示一个空集，用于初始化过程），会尝试激活它的每个尚未激活的出邻居节点 v，即 $v \in N^+(u) \setminus S_{t-1}$。
3. 激活概率：节点 u 尝试激活节点 v 的成功概率为 $p(u, v)$。每次激活尝试都是独立的，不受其他尝试的影响。
4. 激活结果：
   - 成功激活：如果尝试成功，节点 v 在时刻 t 被激活，即 v 加入到集合 S_t 中。
   - 未成功激活：如果尝试失败，且节点 v 的其他入邻居在时刻 t 也未能成功激活 v，则 v 在时刻 t 保持未激活状态。

迭代与终止

5. 迭代传播：重复步骤 2 和 3，直到在某一时间点没有新的节点被激活，即 $S_t = S_{t-1}$，这时传播过程结束。
6. 终止条件：当没有新的节点可以被激活时，即在某一时刻 t 没有任何新节点加入到激活状态集合 S_t，传播过程自然终止。

如图Figure 2.1 我们使用一个简化的社交网络模型来描述信息的传播过程：

- $t = 0$: 种子节点 3，6 处于活跃状态。
- $t = 1$: 种子节点 3，6 尝试激活邻居节点 1, 2, 5, 8。假设节点 1, 2, 8 被激活。
- $t = 2$: 节点 2 试图激活其邻节点 5,7。假设节点 5, 7 被激活，处于活跃态。节点 8 试图激活邻节点 40。节点 4,0 未被激活，处于非活跃态。
- $t = 3$: 节点 8 试图激活邻节点 4 ，0 。节点 4 ，0 未被激活，处于非活跃态。

传播过程结束，因为没有新的节点可以被激活。

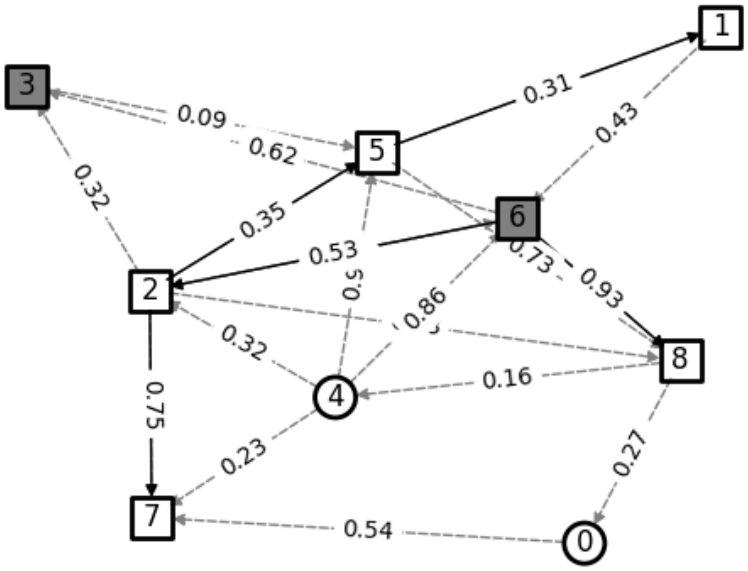

图 2.1: ICM 传播实例

2.3.2 线性阈值模型

在线性阈值模型中，随机性的核心要素在于节点所设定的影响阈值。一旦这些随机阈值被明确设定，后续的信息传播过程将不再具有随机性，而是变得完全可预测和确定性。在此模型中，阈值的选择通常在 0 到 1 的范围内随机进行，这种随机性体现了对实际节点阈值不确定性的一种模拟。然而，在实际情境中，人类的被影响阈值虽然具有一定的随机性，但这种随机性往往局限于一个相对狭窄的范围内。这是因为人类的行为和决策往往受到多种复杂因素的影响，这些因素会在一定程度上限制阈值的波动范围。

若是在线性阈值模型中采用更窄范围的随机阈值（例如接近固定阈值），虽然能更贴近实际情境，但也会带来一系列挑战。这种限制性的阈值选择会显著增加模型的分析和计算复杂度，因为需要更精细地考虑不同阈值下信息传播的可能性和路径。

因此，线性阈值模型在阈值选择上面临着一个两难的境地：既要保持一定的随机性以模拟现实世界的复杂性，又要避免过度随机化导致模型分析和计算的困难。这也是为何在实际应用中，独立级联模型因其更简单的阈值设定和更易处理的分析过程而得到了更广泛的应用。

在线性阈值模型（Linear Threshold Model, LTM）中，社交网络被描述为一个有向图，其中的每一条有向边 $(u, v) \in E$ 都具有一个介于 0 和 1 之间的权重 $w(u, v)$。这个权重反映了节点 u 对节点 v 的影响力在 v 的所有入邻节点中的相对重要性。对于任一

节点 v，其所有入邻节点 u 的权重之和需满足 $\sum_{u \in N^-(v)} w(u,v) \leq 1$。

此外，每个节点 v 都有一个固定的被影响阈值 θ_v，该阈值在 [0,1] 范围内随机选取，并在整个传播过程中保持不变。传播过程开始时，只有初始种子集合 S_0 中的节点被激活。

在每个时刻 $t \geq 1$，对于所有未激活的节点 $v \in V \setminus S_{t-1}$，计算其所有已激活的入邻节点对其的加权影响总和。如果这个总和大于或等于节点 v 的阈值 θ_v，即满足 $\sum_{u \in N^-(v) \cap S_{t-1}} w(u,v) \geq \theta_v$，则节点 v 在时刻 t 被激活并加入到集合 S_t 中；否则，节点 v 保持不活跃状态。

传播过程持续进行，直到某一时刻不再有新的节点被激活为止，此时传播过程结束。这个模型通过线性组合的方式模拟社交影响力的累积效应，而节点的激活状态由其入邻节点的影响力和阈值共同决定，从而在网络中形成复杂的影响力扩散动态。

Algorithm 2 线性阈值模型[31]

Input: 社交网络 $G_{LT} = (V, E, \lambda)$，种子集 $\phi_0 \subseteq V$.
Output: 最终影响集 Φ^*.

```
Let Φ = φ_0, Φ̄ = V \ φ_0
Let k = 0;
while φ_k ≠ ∅ do
    Let k = k + 1, φ_k = ∅;
    for j ∈ Φ̄ do
        if |Φ_old ∩ N_j| / |N_j| ≥ λ_j then
            φ_k = φ_k ∪ {j};
        end if
    end for
    Φ = Φ ∪ φ_k
    Φ̄ = Φ̄ \ φ_k
end while
Let Φ* = Φ.
return Φ*
```

2.4 影响力最大算法

2.4.1 影响力最大化算法评价指标

以下是部分评价指标：

1. 传播范围（Spread）：它测量由于激活初始种子节点而受到影响的节点的最终数量。传播范围越大，算法被认为越有效。这被认为是主要评估的指标。

2. 运行时间（Running Time）：运行越高效，处理相同社交网络模型用到的时间也就越短。这增强了代码在现实中的可用性。
3. 鲁棒性（Robustness）：衡量算法对于网络结构变化的敏感性。在现实世界中，社交网络的结构可能会因多种因素变化，保证程序不崩溃尤为重要。
4. 可扩展性（Scalability）：算法是否能有效处理更大规模的数据是一个重要考虑因素。

2.4.2 边际效益和子模函数

1. 边际增益（Marginal Gain）：

   边际增益，从字面意义上理解，即指的是在某一特定情境下，当我们向现有集合中引入一个全新元素（例如，在社交网络分析中，这个元素可能是一个新选定的种子节点）时，这一行为所带来的额外效益或增益。具体来说，在影响力最大化的场景中，种子节点的选择对于信息或影响的传播范围和速度具有决定性影响。因此，当我们在选择这些关键节点时，需要仔细权衡每一个潜在选项所带来的潜在效益。边际增益的概念就是在这一过程中起到了关键作用。它帮助我们量化评估，如果我们将一个新的种子节点添加到当前的种子集合中，这一决策将如何影响整体的影响力传播效果。如果我们考虑将一个节点添加到已选择的种子节点集合中，边际增益就是这个节点能够额外影响的未被当前种子集合影响的用户数量。可以用以下公式表示：

   $$\sigma_G(S) = \sigma(S \cup \{v\}) - \sigma(S) \tag{2.1}$$

   节点 v 的边际增益 $\sigma_v(S)$ 定义为当节点 v 被添加到已选节点集合 S 中时，新增被影响的用户数量。这表示在 v 加入 S 后，由于 v 的加入所导致的总影响范围的增加。

2. 子模性质（Submodularity）：

   子模性是一种数学性质，常见于组合优化中。一个函数 $f : 2^N \to \mathbb{R}$ 被称为子模的，如果对于任意两个集合 $S \subseteq T \subseteq N$ 和任意元素 $v \in N$，都满足以下条件：

   $$f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T), \quad \forall v \in V \tag{2.2}$$

   这个不等式表明，向一个较小的集合 S 添加一个元素所带来的增益大于或等于向一个较大的集合 T 添加相同元素的增益。在影响力最大化问题的研究中，子模性

质的重要性不容忽视，因为它为贪心算法在解决此类问题时提供了一种有力的理论支撑。当影响力传播函数 $f(S)$（其中 S 代表种子节点的集合）满足子模性质时，我们可以确信贪心算法能够逐步选取种子节点，并在每一步中挑选出能带来最大边际增益的节点，从而得到一个近似最优解。子模性质确保了随着种子节点集合的逐渐增大，新增节点所带来的边际增益会逐渐减少。这一性质对于贪心算法至关重要，因为它允许我们在每一步中选取当前最优的选择，同时保证整体解决方案的质量。在影响力最大化问题中，应用贪心算法结合子模性质的方法，我们能够有效地在海量节点中识别出最具影响力的种子节点集合，从而最大化信息的传播范围和速度。这种方法的优势在于其高效性和实用性，使得我们能够在复杂网络环境中快速找到有效的解决方案。因此，子模性质在影响力最大化问题中的应用，不仅提高了贪心算法的效率和可靠性，也为我们在实际应用中解决类似问题提供了有力的工具和方法。由于子模性，我们知道这种方法可以得到一个至少是最优解 $(1-1/e)$ 的近似解，其中 e 是自然对数的基数（约等于 2.71828）。这些概念不仅在理论研究中非常重要，也对实际应用，如市场营销策略、信息传播、病毒营销等，有着直接的实用价值。通过理解边际增益和子模性质，可以更有效地设计和实施策略，以最大化资源利用效率和影响力。

2.4.3 贪心算法

尽管贪心算法在每一步都仅选择当前看似最优的解，从而得到的解被称为局部最优解，且其决策过程并未全局考量所有可能性，但我们仍然可以利用贪心算法的特性来逼近全局最优解。贪心策略的关键在于，尽管它是基于局部最优选择的，但在某些问题中，特别是当满足特定性质（如子模性）时，这些局部最优选择能够累积成高质量的近似全局最优解。因此，尽管贪心算法不直接追求全局最优，但在许多实际应用中，它仍然能够有效地为我们提供全剧最优的近似解。

Algorithm 3 贪心算法 $Greedy(k, f)$

input: 子集 f, 种子集节点数 k
output: 种子集 S

```
    Initialize S = ∅
    for i = 1 to k do
3:      Select u = arg max_{w∈V\S} {f(S ∪ {w}) − f(S)}
        S = S ∪ {u}
    end for
6:  Output S
```

贪心算法时间复杂度：

在解决影响力最大化问题时，当种子节点集合 S 初始只包含一个节点时（即 $|S| = 1$），算法的首要任务是在所有可能的节点中，根据独立级联（IC）模型，识别出能带来最大边际增益的节点。为实现这一目的，我们通常采用蒙特卡洛模拟来评估每个节点的边际增益，即对每个节点进行 m 次模拟。评估过程涉及遍历所有 n 个节点，并对每个节点执行 m 次模拟，因此这一步的时间复杂度是 $O(n \times m)$，其中 n 是节点总数，m 是模拟次数。

一旦初始种子节点被选定，算法会进入迭代过程，继续选择其他种子节点，直到种子集合的大小达到预定的 k。在每次迭代中，算法都会寻找那个在当前种子集合下能带来最大边际增益的节点，并将其加入到集合 S 中。这一过程中，对于剩余的每个节点，都需要再次进行 m 次模拟来计算其边际增益，并比较所有节点的边际增益以做出选择。因此，每次选择种子的时间复杂度也是 $O(n \times m)$。

考虑到需要选择 k 个种子节点，且每次选择的时间复杂度为 $O(n \times m)$，算法的总时间复杂度在直观上似乎是 $O(k \times n \times m)$。然而，需要注意的是，在每次选择过程中，随着种子集合 S 的扩大，计算剩余节点边际增益的复杂性也会增加，因为需要考虑的节点之间的相互影响会变得更加复杂。但这并不意味着总的时间复杂度会达到 $O(k \times n^2 \times m^2)$，因为模拟的每次迭代仍然是独立的，并且边际增益的计算可以利用之前迭代的结果进行优化。因此，虽然实际的计算复杂度可能会受到节点间相互影响和模拟次数的影响而略高于 $O(k \times n \times m)$，但通常不会达到 $O(k \times n^2 \times m^2)$ 的级别。在实际应用中，我们通常会根据问题的具体特性和计算资源来调整模拟次数 m，以在可接受的计算时间内获得高质量的近似解。

综上所述，原始贪心算法在影响力最大化问题中的时间复杂度是非常高的，特别是在节点数量和所需模拟次数较大的情况下。这也解释了为什么在实际应用中常常寻求更有效的算法，如使用启发式或近似算法来降低时间复杂度。

2.4.4 启发式算法

在影响力最大化问题中，最基础的启发式算法包括由 Kempe[3]等人提出的随机算法（Random）、度启发式算法（Degree）[32]和中心启发式算法（Centrality）[33]。这些算法主要利用网络的结构特征来识别具有较高影响力的节点，它们的运行速度较快但精度较低。

随机启发式算法在选择种子节点时，完全不考虑节点的具体影响力或其他相关因素，而是随机选择 k 个节点。度启发式算法则通过计算网络中每个节点的连接度

数，选择度数最高的前 k 个节点作为种子节点。度中心性启发式算法的核心思想是节点的中心性越高，其影响力越大，即节点与其他所有节点的平均距离越短，其影响力越大。因此，此算法选择平均距离中心性最小的前 k 个节点作为种子节点。

尽管这些启发式算法简单且计算速度快，它们由于没有充分考虑影响力传播的动态过程和节点的具体影响力特征，因此在精度上表现不佳。为了提高效率和准确性，一些学者提出了将贪婪算法与启发式算法结合的混合方法。这种方法的基本策略是先使用贪婪算法找到一部分种子节点，然后利用启发式算法完成剩余种子节点的选择。例如，在启发阶段，可以根据度中心性评估每个未激活节点的潜在影响力，选择具有较大潜在影响力的节点作为一部分种子节点，而剩余的种子节点则由贪婪算法选择。这种结合方法不仅降低了计算的时间复杂度，还提高了算法的精度。

定义 2.4.1 (度中心性[34]). 度中心性指的是与一个节点直接相连的边的数量，可以表示为 $CD(u) = |N(u)|$，即节点 u 的邻接节点数量。在 *ACO-IM*（蚁群优化算法的影响力最大化）问题中，我们特别关注节点的出度，因此将度中心性定义为 $CD(u) = |N_{out}(u)|$，这里 $N_{out}(u)$ 表示从节点 u 出发指向其他节点的边的集合。

2.5 蚁群算法

2.5.1 蚁群算法基础介绍

蚁群算法（Ant Colony Optimization, ACO）是 Marco Dorigo 于 1992 年提出的一种启发式优化算法，它受到自然界中蚂蚁觅食行为的启发[35]。该算法以其卓越的性能，特别擅长于解决一系列路径优化问题，诸如经典的旅行商问题（TSP）和复杂的车辆调度问题等。受自然界中蚂蚁寻找食物源与巢穴之间路径的行为启发，ACO 算法模拟了蚂蚁通过释放和感知“信息素”这一化学物质来进行路径选择的机制。在算法执行过程中，信息素会根据不同路径的优劣进行不断的构建和更新。这种机制使得 ACO 算法能够逐步聚焦于最优解的搜索，从而有效地优化给定的问题。通过模拟蚂蚁寻找食物源的自然行为，ACO 算法展现出了强大的优化能力和广泛的应用前景。

2.5.2 蚁群算法-最大影响力问题

1. 对图中每一个节点的信息素浓度进行初始化。
2. 基于影响力构造可行解集（可以用线性阈值模型，独立级连模型，传染病模型等传播模型估算影响力）。

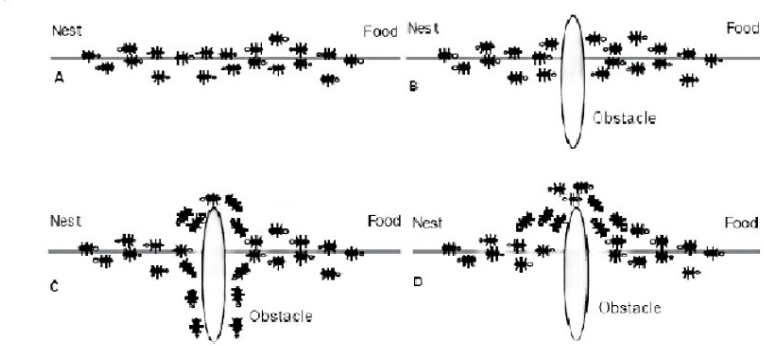

图 2.2: 蚂蚁寻路示意图

3. 可行解进行评估（找到扩散最优的可行解，并作记录）。
4. 节点信息素浓度更新。
5. 迭代步骤 2 到 4，直到设定的迭代次数。

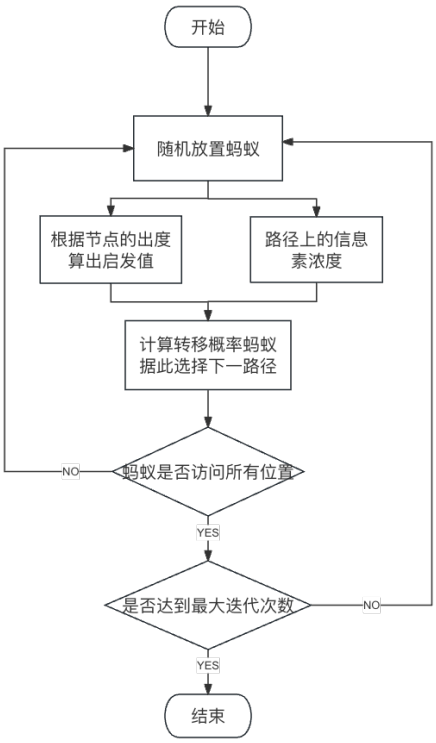

图 2.3: ACO-IM 流程图

信息素值: 信息素值描述了解的质量。通过一定百分比减少所有信息素来模拟随时间自然衰减的轨迹强度（信息素蒸发），使得未来的蚂蚁不太可能遵循旧路径。蒸发后，在成功解决方案的路径上沉积额外的信息素。这样做是为了加强好的解决方案并使其对未来的蚂蚁更具吸引力。最初，节点的信息素值是由 ϵ 初始化的。设 $\tau(u)$ 和 ρ 分别为节点 u 上的信息素值和信息素的蒸发率，如果 I 是迭代次数，那么节点 u 的

表 2.1: 概念

符号	描述
$G(V, E)$	具有点集 V 和边集 E 的社交网络
$N(u)$	节点 u 的邻居集
S	种子集
k	种子集中的节点数 $\|S\|$
$\sigma(S)$	网络中种子集的预期影响力
pop	蚁群中的数量
Solution(A_i)	蚂蚁 A_i 的解向量
$I_{\max}$	最大迭代次数
$\tau_I(u)$	迭代 I 时节点 u 的信息素值
ρ	蒸发率参数
$\eta(u)$	节点 u 的质量值
f(Solution(A_i))	由 A_i 生成的解决方案的适应度值

信息素 $\tau(u)$ 可以按以下方式更新：

$$\tau_{I+1}(u) \leftarrow (1-\rho) \cdot \tau_I(u) + \Delta\tau_I, \tag{2.3}$$

其中，ρ 是信息素挥发系数，(1_ρ) 是信息素的残留系数，如果 ρ 值过小，会导致路径上信息素残留过多，无法有效区别最优解和较优解的差距，如果该值过大，虽然可以排除无效路径，但也可能大幅度减少有效路径上的信息素，从而增加了算法所需的时间。$\Delta\tau_I$ 是信息素的增量通常计算为：

$$\Delta\tau_i = \frac{Q}{L} \tag{2.4}$$

Q 是蚁群算法的信息素更新强度系数。L 是解决方案的质量，通常通过目标函数值（如当前解集 S 的影响力扩散）来衡量。

启发式信息: 启发式值是基于适应度函数估算的，即 ACO-IM 中的局部影响。表示节点是否容易被激活，$\eta(u)$ 越大节点 u 被选择的概率也会更高。设 $d_{out}(u)$ 为节点的出度，则节点 u 的质量值 $\eta(u)$ 评估如下：

$$\eta(u) \leftarrow d_{out}(u). \tag{2.5}$$

解集: 每只蚂蚁 A_i 维护一个解集 Solution$(A_i) = (s_1, s_2, \ldots, s_k)$，其中 $|\text{Solution}(A_i)| = k$。解向量在每次迭代中使用 (2.3) 更新，参数 α 是信息素启发式因子，它表示信息素浓度对蚂蚁选择节点的转移概率的影响程度。当 α 的值增大时，蚂蚁更倾向于选择

信息素浓度较高的路径，因此搜索过程中的随机性减弱，但也可能导致降低解的多样性，从而减少发现全局最优解的概率。相反，当 α 较小时，蚂蚁的搜索范围减小，容易陷入局部最优解。另一方面，参数 β 是期望启发因子，它影响着算法成为正反馈搜索的程度。较大的 β 值会使蚂蚁更倾向于选择局部最优路径，因此可能加快收敛速度，但可能无法找到全局最优路径。位于节点 x 的蚂蚁 A_i 决定按以下规则移动到节点 y

$$y \leftarrow \begin{cases} \arg\max_{v \in V \setminus \{x\}} \left\{ [\tau(v)]^{\alpha} [\eta(v)]^{\beta} \right\} & r < r_0, \\ N & r \geq r_0, \end{cases} \tag{2.6}$$

其中 $N \in V$ 是基于以下概率分布选择的随机节点：

$$p(u) \leftarrow \frac{[\tau(u)]^{\alpha} [\eta(u)]^{\beta}}{\sum_{w \in V} [\tau(w)]^{\alpha} [\eta(w)]^{\beta}}. \tag{2.7}$$

Algorithm 4 蚁群算法影响力最大化问题 (ACO-IM)

input:ϵ：初始信息素水平，q0：开发和探索之间的平衡，k：选择的影响者数量，ants‘：蚂蚁代理的数量，α：信息素影响的权重，β：启发式影响的权重，ρ：信息素蒸发率，steps：迭代次数，
output: 解集 S

初始化所有边的初始信息素为 ϵ，和节点的影响力为 0
for i in iteration **do**
    for 每一只 ant **do**
        初始化解集为空 S
        while $|S| < k$ **do**
            根据 (2.6) 选择节点 v
            将 v 添加到 S
        end while
        评估 S 的影响力扩散（使用 IC 模型）
    end for
    更新信息素:
    for each node i **do**
        $\tau_{i+1} \leftarrow (1 - \rho) \cdot \tau_i$ ▷ 信息素蒸发
        for 每一只在迭代次数为 i 的在它当前解中 **do**
            $\tau_{i+1} \leftarrow \tau_i + \Delta\tau_i$ ▷ 信息更新
        end for
    end for
end for
返回最佳解集 S

ACOIM 算法的时间复杂度

在 ACOIM 算法中，时间复杂度的描述如下：

1. 初始阶段：算法首先对网络图中的信息素浓度进行初始化。这一步的时间复杂度是 $O(|V||E|\log m)$，其中 $|V|$ 是网络中节点的数量，$|E|$ 是边的数量，m 是某个与信息素相关的参数。
2. 迭代阶段：每次迭代包含以下几个步骤：
   - 每次迭代中，算法需要构造 m 个可行解，每个解由 k 个节点组成，每次节点的选择涉及对所有 $|E|$ 条边的考量。因此，这一步的时间复杂度为 $O(mk|V||E|)$。
   - 每个解的评估需要考察每条边，所以这一步的时间复杂度为 $O(m|E|)$。
   - 更新信息素同样需要遍历所有边，并且考虑到每个解，这一步的时间复杂度为 $O(m|E|k)$。
3. 迭代总时长：如果算法进行 N_c 次迭代，总的迭代时间为 $t = N_c(mk|V| + m|E| + m|E|k)$。

综上所述，整个 ACOIM 算法的总时间复杂度为 $O(|V||E|\log m + t)$，这里的 t 表示迭代过程中各步骤的总时间复杂度。

3 粒子群优化的蚁群算法解决影响力最大化问题

3.1 粒子群算法

粒子群优化算法（Particle Swarm Optimization, PSO）的构思深受自然界中鸟群觅食行为的启发。其核心理念是通过模拟鸟群成员间的协作和信息交流机制，来探索并寻求复杂优化问题的最优解决方案。这种方法由 Eberhart 和 Kennedy 于 1995 年首次提出[36]。在 PSO 中，每个潜在的解都被比拟为鸟群中的一个个体，这些个体被称为“粒子”。每个粒子都拥有一个由适应度函数评估得出的适应度值，这个值反映了粒子在搜索空间中的优劣程度。通过粒子间的相互协作和信息共享，PSO 算法能够逐步引导粒子群体向问题的最优解逼近。

粒子群优化（PSO）算法的核心在于模拟鸟群的社会行为，其中每个粒子代表解空间中的一个潜在解。这些粒子不仅具有当前的位置，还具备一个速度向量，用以决定它们在解空间中的移动方向。

公式如下：

1. 速度更新公式：

$$v_i^{(t+1)} = w \cdot v_i^{(t)} + c_1 \cdot r_1 \cdot (pBest_i - x_i^{(t)}) + c_2 \cdot r_2 \cdot (gBest - x_i^{(t)}) \tag{3.1}$$

- $v_i^{(t)}$ 是粒子 i 在时间 t 的速度。
- w 是一个非负数。1998 年，Shi 和 Eberhart 引入了惯性权重 w[37]，动态调整这个值可以调整全局或者局部的搜索能力。
- c_1, c_2 是学习因子，通常 c_1 是自我认知部分，c_2 是社会认知部分。通常假设这两个值相等。
- r_1, r_2 是 [0,1] 区间内的随机数，增加搜索的随机性
- $x_i^{(t)}$ 是粒子 i 在时间 t 的位置。
- $pBest_i$ 是粒子 i 的历史最佳位置。
- $gBest$ 是全局最佳位置。

在粒子群优化算法中，公式的分解揭示了其工作的核心机制。首先，我们称之为“记忆项”的部分，它体现了粒子在上一次迭代中速度大小和方向对其当前运动状态的直接影响。紧接着，我们引入“个体认知项”的概念，这一项反映了粒子根

据自身历史搜索记录中发现的最佳解来更新其搜索方向的倾向。最后，我们称之为“群体认知项”的部分，它描述了粒子如何受到其邻域内其他粒子所发现的最优解的影响，展现了粒子间通过协同合作和知识共享来指导搜索方向的能力。简而言之，粒子正是基于自身的经验和群体中的最佳经验，共同决定其下一步的运动方向。

2. 位置更新公式：

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)} \tag{3.2}$$

在这些更新公式中，惯性权重 w 用于平衡全局搜索和局部搜索的能力：较大的 w 值有助于全局搜索，而较小的值有利于局部搜索。学习因子 c_1 和 c_2 决定了粒子是更倾向于自身的历史经验还是群体中其他粒子的经验。在粒子群优化算法的更新公式中，惯性权重 w 扮演了一个至关重要的角色，它用于调控算法在全局搜索和局部搜索之间的平衡。具体而言，较大的 w 值能够赋予粒子更大的动量，使其更易于进行全局性的探索，而较小的 w 值则有助于粒子在局部范围内进行精细的搜索。

此外，学习因子 c_1 和 c_2 也是算法中不可或缺的参数，它们决定了粒子在决策时是更依赖于自己的历史经验还是更受群体中其他粒子经验的影响。c_1 的值越大，粒子越倾向于依赖自己的历史最优解来更新其位置和速度；而 c_2 的值越大，则粒子更可能受到群体中最优粒子的影响，通过社会学习来优化其搜索方向。因此，合理设置这两个学习因子的值，对于实现算法的有效性和高效性至关重要。

3.2 ACO-PSO 算法[13][14]

高复杂度是前一个启发式方法解决团问题的主要缺点，它显著增加了计算量。尽管如此，所有迄今为止提供的方法都采用了几乎相同的方法来计算 $\Delta\tau$。然而，新提出的算法利用了粒子群优化（PSO）算法的优势，旨在改善结果并减少复杂性。

本算法融合了蚁群优化（ACO）和粒子群优化（PSO）的思想，旨在图中寻找最大的团。算法开始时，首先在图上放置一定数量的“蚂蚁”，这些蚂蚁会按照特定的路径搜索以识别最大的团。随着搜索过程的进行，信息素会随时间蒸发，而成功的路径（即找到更大团的路径）则通过其边上的信息素量得到增强。

为了进一步提升搜索效率，算法引入了粒子群优化的策略。具体地，在每次迭代中，使用 PSO 算法来更新信息素的分布，确保信息素的总量与当前步骤中发现的最佳团紧密相关。这一过程中，不仅考虑了当前步骤中路径上的信息素量，还结合了 PSO 算法中粒子间的信息共享和协同合作，以指导蚂蚁在后续的搜索中更准确地定

位最大团的位置。

整个算法通过反复迭代，不断优化信息素的分布和蚂蚁的搜索策略，直到在图上找到最优的团为止。这种结合 ACO 和 PSO 的混合算法策略，有效地提高了在图中寻找最大团的效率和准确性。

现在，所需团的加强信息素 $\Delta\tau$ 将使用 PSO 算法根据以下公式计算：

$$\Delta\tau^{t+1} = \Delta\tau^{t} + \nu^{t} \tag{3.3}$$

其中，$\Delta\tau_t^i$ 是当前信息素的加强量，$\Delta\tau_{t+1}^i$ 是新信息素的加强量，而 ν^t 表示这个方程可以实现的变化量：

$$\nu^{t+1} = c_1 r_1(p\tau\ -\Delta\tau) + c_2 r_2(g\tau\ -\Delta\tau) + c_3\nu^{t} \tag{3.4}$$

其中，v^{t+1} 是 V^t 的新值。r_1 和 r_2 是范围在 (0,1) 内的两个随机值，而 c_1, c_2 和 c_3 是学习参数。$p\tau$ 和 $g\tau$ 分别被视为与当前最佳团和迄今为止发现的最佳团对应的信息素。在这种情况下，信息素的变化量的发现将会更加智能地实施。在探索信息素变化量时，这种方法实现得更加智能。综合考虑上述事项，该算法的伪代码实现如下：

Algorithm 5 粒子群优化的蚁群算法影响最大化 (ACO-PSO-IM)

input:ϵ，q0，k，ants，α，β，ρ，steps，c_1, c_2 和 c_3：学习参数
output: 解集 S

对每只蚂蚁随机选择一个起始顶点 $v_f \in V$
$C \leftarrow \{v_f\}$
candidates $\leftarrow \{v_f|(v_i, v_f) \in E\}$
while candidates $\neq \emptyset$ **do**
    从候选集中根据 (2.3) 的概率选择一个顶点 v_i 并加入到 C 中
    更新候选集：candidates $\leftarrow$ candidates $\setminus \{v_f|(v_i, v_f) \in E\}$
end while
执行信息素蒸发 Evaporate
评估所有蚂蚁当前找到的最佳团
基于粒子群优化（PSO）通过 (3.3)(3.4) 更新信息素 $\Delta\tau$

4 实验

4.1 实验准备

4.1.1 实验环境和数据集

实验在 MACOS(apple silicon) 下运行，主要运用 python 实现，开发工具是 PyCharm 2022.3.3 (Professional Edition)。本文的实验部分使用了若干真实的社会网络数据集，这些数据集均从斯坦福大学的 SNAP 平台下载，格式为 txt 文件，要求节点，边数量合适且符合社交网络分布要求。其中 soc-Epinions1 这是一个普通消费者评论网站 Epinions.com 的谁信任谁的在线社交网络。网站成员可以决定是否相互“信任”以构建成信任网络。Ego-Facebook 数据集是基于 Facebook 用户的“圈子”（也被称为“朋友列表”）构建而成的，常用于社交网络研究。

数据集	soc-Epinions1	ego-Facebook
节点数	77360	4039
边数	905468	88234
平均度数	11.704	21.84

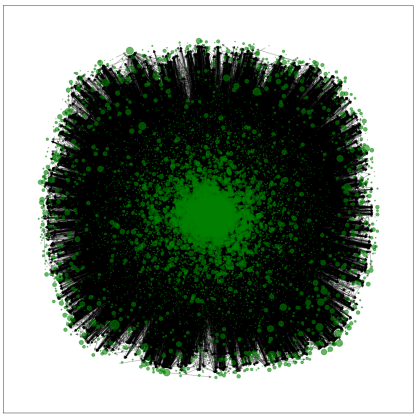

图 4.1: soc-Epinions1

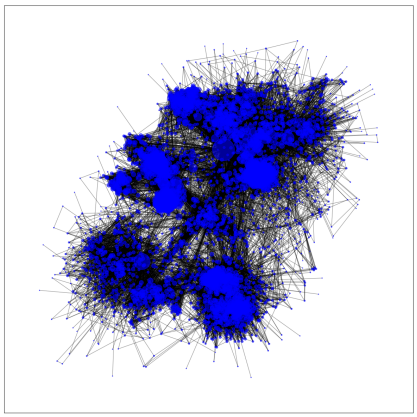

图 4.2: facebook

图 4.3: 数据集可视化

4.1.2 参数设置

参数设置如下所示。需要注意的是，选择不同的值结果也是不同的。

ACO-PSO 算法使用了 30 只蚂蚁，其中 $\rho = 0.95$，$\Phi = 0.0002$，$\Delta\tau_{initial} = 0$，$\tau_{min} = 0.01$，$\tau_{max} = 6$。同时，PSO 的参数初始化为 $V = 0$，$c_1 = c_3 = 0.3$，$c_2 = 1 - c_1$。α 和 ρ 的值根据实验结果和迭代次数 t 设定如下。

$$\alpha(t) = \begin{cases} 1 & t \leq 100 \\ 2 & 100 < t \leq 400 \\ 3 & 400 < t \leq 800 \\ 4 & t > 800 \end{cases} \tag{4.1}$$

$$\rho(t+1) = \begin{cases} (1-\phi)\rho(t) & \text{if } \geq (t) > 0.95 \\ 0.95 & \text{if } \rho(t) > 0.95 \end{cases} \tag{4.2}$$

ACO 算法使用 30 只蚂蚁，其中参数设置为：α= (4.1), β=1, ρ=0.85, ϵ=0.01, q_0=0.8

对于所有的有向图和无向图，我们将对其边设置 0 到 1 的随机值以用作独立级连模型的参数。

4.2 实验结果

Facebook 结果（每条边概率随机）

K	算法类型	影响力扩散系数	收敛时间 (seconds)
5	ACO-IM	810	0.6
	ACO-PSO-IM	813	1.3
10	ACO-IM	3549	35.3
	ACO-PSO-IM	3698	15.7
20	ACO-IM	1863	160.9
	ACO-PSO-IM	1897	55.5
30	ACO-IM	3308	153.7
	ACO-PSO-IM	3318	116.5

soc-Epinions1 结果（每条边概率相等）

K	算法类型	影响力扩散系数	收敛时间 (seconds)
5	ACO-IM	10673	27.8
	ACO-PSO-IM	10720	185.3
10	ACO-IM	10684	70.1
	ACO-PSO-IM	10768	2298.7
15	ACO-IM	17470	347.3
	ACO-PSO-IM	17454	1952.6

上面两张图表给出了影响力扩散系数和收敛时间，影响力扩散系数越大，说明找到的种子集影响的范围越大，效果越好。而收敛时间则代表运行的效率，时间越短说明得出解的速度越快，效率越高。影响扩散系数大，收敛时间短则程序性能越好。

因为每次运行所选取的每条边的系数不同，所以只能在每次运行结果之间比较。如facebook 结果，大部分情况下，ACO-PSO-IM 无论在求解速度和算法虽然大部分时间 ACO-PSO-IM 算法的影响力扩展和收敛速度都优于 ACO-IM 算法。但在最大影响力集合与启发值较大的集合重合度较高时（如解集 k 较小，或者每条边的概率值相等），ACO-IM 算法有更好的表现。如下所示，在 Facebook 数据 $k = 10$ 的一次计算中，虽然 ACO-IM 更早收敛，但 ACO-PSO-IM 算法却得出了更优解，也就是更大影响力的集合，说明这个算法拥有更好的搜索能力。

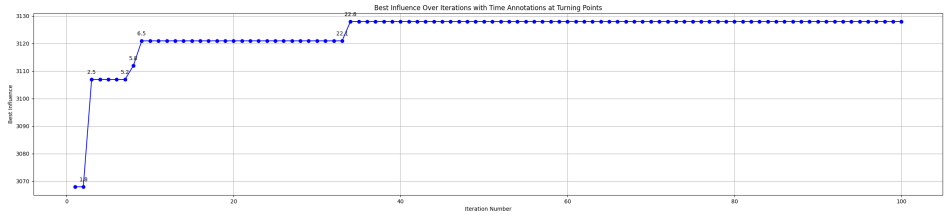

图 4.4: ACO-IM

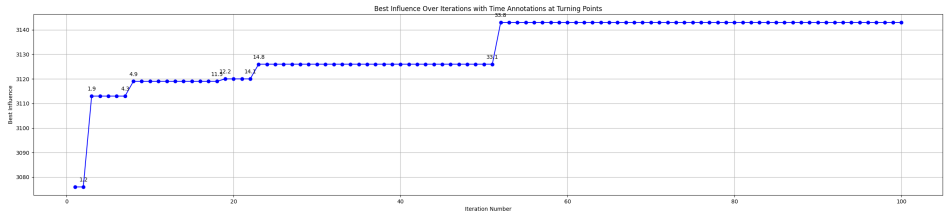

图 4.5: ACO-PS-IM

4.3 本章小结

本文提出了一种新的混合算法，使用 ACO 和 PSO (ACO-PSO) 来寻找最大团问题。传统算法复杂度较高，而混合算法只是改变了更新信息素的过程。事实证明，新算法能够简单、快速地改进基本 ACO 算法。在流行的社交网络数据集上的仿真结果表明，与 ACO 算法相比，所提出的算法具有好的结果。

5 总结与展望

5.1 总结

在深入研究社交网络中的影响力最大化问题（IM）时，蚁群优化算法（ACO）和粒子群优化算法（PSO）作为两种备受瞩目的启发式优化技术，已被学术界和工业界广泛采纳和应用。这两种算法在解决如何在复杂的社交网络中识别并选择最具影响力的个体，以实现信息传播的最大化问题上，均展现出了显著的效能。

影响力最大化问题的研究通常聚焦于两大核心方面：模型优化与算法优化。模型优化旨在通过不断精细化和完善模型，使其更加贴近实际社交网络的结构和动态特性；而算法优化则聚焦于提升计算效率，确保在有限的时间内能够找到高质量的解决方案。

通过结合蚁群优化算法（ACO）的全局搜索能力和粒子群优化算法（PSO）的局部搜索优势，研究者们期望能够开发出更为高效且准确的影响力最大化算法，以应对社交网络分析领域日益增长的需求和挑战。

研究的主要发现和贡献：

1. 算法适应性和融合：研究表明，ACO 和 PSO 可以有效地适应社交网络的复杂性和动态性。通过调整算法参数，如信息素挥发率和粒子速度更新规则，可以在社交网络环境中获得更好的性能。ACO 和 PSO 的结合使用，即蚁群和粒子群的混合算法，已经被证明能有效提高解的质量和算法的稳定性，尤其是在面对大规模网络时。
2. 模型的创新和应用：在信息传播模型的研究领域，研究者们一直在努力提升模型对现实世界中信息传播过程的模拟精度。这包括引入全新的传播模型，以及对现有模型如独立级联模型和线性阈值模型进行细致且富有成效的改进。这些创新旨在使算法能够更加精准地捕捉社交网络中的复杂传播机制，从而更贴近真实世界的动态。将蚁群优化算法（ACO）和粒子群优化算法（PSO）应用于影响力最大化问题，不仅体现了这些启发式优化技术的强大能力，还展现了它们在处理网络复杂交互和节点间非线性关系时的独特优势。这种应用使得算法能够在市场营销、公共健康信息传播等实际场景中发挥更大的作用，实现更广泛的社会影响。通过这些创新模型和优化技术的应用，我们能够更加精准地预测和控制信息传播过程，

为决策者提供有力的支持。

3. 性能评估和优化：通过大量实验和比较研究，ACO 和 PSO 在不同类型的社交网络数据集上的表现被详细分析。结果显示，这些算法不仅在找到最有影响力的节点集合方面有效，而且在计算效率上也具有明显优势。针对特定问题和数据特征的参数调整和算法优化，进一步提高了算法在实际应用中的适用性和效果。

5.2 展望

尽管蚁群优化算法（ACO）和粒子群优化算法（PSO）在影响力最大化问题上已经取得了显著的进展，但这一领域仍面临诸多挑战和潜在的改进空间。一种值得探索的改进策略是引入动态惯性因子。与固定值相比，动态惯性因子在 PSO 搜索过程中能够展现出更加灵活和适应性强的特性。这种动态惯性因子可以设计为线性变化，也可以根据 PSO 性能的特定测度函数进行动态调整，以期达到更快的收敛速度和更优的解质量。

此外，跨学科的合作将为这一领域的研究带来新的活力。通过结合不同学科的理论和方法，我们可以开发出更加全面和深入的理论模型以及实用工具，以更好地理解和利用社交网络中的影响力动态。这种跨学科的研究方法将有助于我们更全面地理解复杂网络行为，并为优化网络干预策略提供更多创新的方法和深入的见解。

因此，未来的研究可以进一步探索动态惯性因子的设计和应用，同时加强跨学科的合作，以期在影响力最大化问题上取得更加显著的进展。

参考文献

[1] CHRISTAKIS N A, FOWLER J H. The spread of obesity in a large social network over 32 years[J]. New England journal of medicine, 2007, 357(4): 370-379.

[2] CHRISTAKIS N A, FOWLER J H. The collective dynamics of smoking in a large social network[J]. New England journal of medicine, 2008, 358(21): 2249-2258.

[3] KEMPE D, KLEINBERG J, TARDOS É. Maximizing the spread of influence through a social network[C]//Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. 2003: 137-146.

[4] KEMPE D, KLEINBERG J, TARDOS É. Influential nodes in a diffusion model for social networks[C]//Automata, Languages and Programming: 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005. Proceedings 32. 2005: 1127-1138.

[5] CHEN W, ZHANG Y, LI X. Network representation learning[J]. Big Data Res, 2015, 1(3): 8-22.

[6] MANDALA S R, KUMARA S R, RAO C R, et al. Clustering social networks using ant colony optimization[J]. Operational Research, 2013, 13: 47-65.

[7] XU X, MA J, LEI J. An improved ant colony optimization for the maximum clique problem[C]//Third international conference on natural computation (ICNC 2007): vol. 4. 2007: 766-770.

[8] DORIGO M, MANIEZZO V, COLORNI A. Ant system: optimization by a colony of cooperating agents[J]. IEEE transactions on systems, man, and cybernetics, part b (cybernetics), 1996, 26(1): 29-41.

[9] FENET S, SOLNON C. Searching for maximum cliques with ant colony optimization[C]//Applications of Evolutionary Computing: EvoWorkshops 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM Essex, UK, April 14–16, 2003 Proceedings. 2003: 236-245.

[10] SOLNON C, FENET S. A study of ACO capabilities for solving the maximum clique problem[J]. Journal of Heuristics, 2006, 12: 155-180.

[11] HUIYING W, JIALI G. Solving Maximum Clique problem on Ant Colony Optimization[J]. Computer Application and Software, 2010, 27(10): 107-113.

[12] RONG C. Improved Ant Colony Algorithm for Maximum Clique Problem[J]. Microprocessors, 2011(1): 64-66.

[13] SOLEIMANI-POURI M, REZVANIAN A, MEYBODI M R. Finding a maximum clique using ant colony optimization and particle swarm optimization in social networks[J]. arXiv preprint arXiv:1311.7213, 2013.

[14] SOLEIMANI-POURI M, REZVANIAN A, MEYBODI M R. An ant based particle swarm optimization algorithm for maximum clique problem in social networks[J]. State of the art applications of social network analysis, 2014: 295-304.

[15] SALAVATI C, ABDOLLAHPOURI A. Identifying influential nodes based on ant colony optimization to maximize profit in social networks[J]. Swarm and Evolutionary Computation, 2019, 51: 100614.

[16] BARNES J A. Class and committees in a Norwegian island parish[J]. Human relations, 1954, 7(1): 39-58.

[17] KOUKI P. Resolution, Recommendation, and Explanation in Richly Structured Social Networks[M]. University of California, Santa Cruz, 2018.

[18] 陈卫. 大数据网络传播模型和算法[M]. 大数据网络传播模型和算法, 2020.

[19] LESKOVEC J, ADAMIC L A, HUBERMAN B A. The dynamics of viral marketing [J]. ACM Transactions on the Web (TWEB), 2007, 1(1): 5-es.

[20] RICHARDSON M, DOMINGOS P. Mining knowledge-sharing sites for viral marketing[C]//Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. 2002: 61-70.

[21] MA C, ZHU C, FU Y, et al. Social user profiling: A social-aware topic modeling perspective[C]//Database Systems for Advanced Applications: 22nd International Conference, DASFAA 2017, Suzhou, China, March 27-30, 2017, Proceedings, Part II 22. 2017: 610-622.

[22] XU T, ZHU H, ZHAO X, et al. Taxi driving behavior analysis in latent vehicle-to-vehicle networks: A social influence perspective[C]//Proceedings of the 22nd ACM

SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016: 1285-1294.

[23] ZHAO X, XU T, LIU Q, et al. Exploring the Choice Under Conflict for Social Event Participation[C]//NAVATHE S B, WU W, SHEKHAR S, et al. Database Systems for Advanced Applications. Cham: Springer International Publishing, 2016: 396-411.

[24] BACKSTROM L, LESKOVEC J. Supervised random walks: predicting and recommending links in social networks[C]//Proceedings of the fourth ACM international conference on Web search and data mining. 2011: 635-644.

[25] LIBEN-NOWELL D, KLEINBERG J. The link prediction problem for social networks[C]//Proceedings of the twelfth international conference on Information and knowledge management. 2003: 556-559.

[26] XU T, ZHU H, CHEN E, et al. Learning to annotate via social interaction analytics[J]. Knowledge and information systems, 2014, 41: 251-276.

[27] FORTUNATO S. Community detection in graphs[J]. Physics reports, 2010, 486(3-5): 75-174.

[28] FORTUNATO S, BARTHELEMY M. Resolution limit in community detection[J]. Proceedings of the national academy of sciences, 2007, 104(1): 36-41.

[29] GIRVAN M, NEWMAN M E. Community structure in social and biological networks [J]. Proceedings of the national academy of sciences, 2002, 99(12): 7821-7826.

[30] CHEN W, YUAN Y, ZHANG L. Scalable influence maximization in social networks under the linear threshold model[C]//2010 IEEE international conference on data mining. 2010: 88-97.

[31] GIUA A. Mathematical models for the diffusion of innovation in social networks[J]. Notes for the course"Control of Network Systems", Polytech Marseille, Aix-Marseille University,

[32] CHEN W, WANG Y, YANG S. Efficient influence maximization in social networks [C]//Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. 2009: 199-208.

[33] KUNDU S, MURTHY C, PAL S K. A new centrality measure for influence maximization in social networks[C]//Pattern Recognition and Machine Intelligence: 4th

International Conference, PReMI 2011, Moscow, Russia, June 27-July 1, 2011. Proceedings 4. 2011: 242-247.

[34] SINGH S S, SINGH K, KUMAR A, et al. ACO-IM: maximizing influence in social networks using ant colony optimization[J]. Soft Computing, 2020, 24(13): 10181-10203.

[35] DORIGO M. Optimization, learning and natural algorithms[J]. Ph. D. Thesis, Politecnico di Milano, 1992.

[36] KENNEDY J, EBERHART R. Particle swarm optimization[C]//Proceedings of ICNN'95-international conference on neural networks: vol. 4. 1995: 1942-1948.

[37] SHI Y, EBERHART R. A modified particle swarm optimizer[C]//1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360). 1998: 69-73.

附录

ACO-PSO-IM 实验代码

```
import networkx as nx
import random
import time
import matplotlib.pyplot as plt
import scipy as sp
```

算法 5.1: Python 标准库

```
def plot_influence_vs_iterations(best_influences, cumulative_times):
    plt.figure(figsize=(30, 6))
    iterations = list(range(1, len(best_influences) + 1))
    plt.plot(iterations, best_influences, marker='o', linestyle='-',
    color='b')
    for i in range(1, len(best_influences) - 1):
        if (best_influences[i] < best_influences[i + 1] or
    best_influences[i - 1] < best_influences[i]):
            x = iterations[i]
            y = best_influences[i]
            t = cumulative_times[i]
            plt.annotate(f'{t:.1f}', (x, y), textcoords="offset points",
    xytext=(0, 10), ha='center')
    plt.title('Best Influence Over Iterations with Time Annotations at
    Turning Points')
    plt.xlabel('Iteration Number')
    plt.ylabel('Best Influence')
    plt.grid(True)
    plt.show()
```

算法 5.2: 可视化结果程序

```
class ACO_IM:
    def __init__(self, graph, k, ants, steps, alpha, beta, rho, epsilon,
    q0):
        self.graph = graph
```

```
        self.k = k
        self.ants = ants
        self.steps = steps
        self.alpha = alpha
        self.beta = beta
        self.rho = rho
        self.epsilon = epsilon
        self.q0 = q0
        self.pheromones = {node: epsilon for node in self.graph.nodes()}
        self.best_set = None
        self.best_influence = 0

    def run(self):
        best_influences = []  # List to store the best influence of each iteration
        iteration_times = []  # List to store time taken for each iteration
        start_time = time.time()  # Record the start time of the algorithm
        for step in range(self.steps):
            all_solutions = []
            for ant in range(self.ants):
                solution = self.construct_solution(step)
                influence = self.simulate_influence_spread(solution)
                all_solutions.append((solution, influence))
                if influence > self.best_influence:
                    self.best_influence = influence
                    self.best_set = solution
            self.update_pheromones(all_solutions)
            iteration_end = time.time()  # End time of the iteration
            iteration_times.append(iteration_end - start_time)
            print(
                f"Iteration {step + 1}: Best Solution = {self.best_set}, Best Influence = {self.best_influence}, time = {iteration_end - start_time}")
            best_influences.append(self.best_influence)  # Store the best influence of this iteration
        return self.best_set, self.best_influence, best_influences,
```

```
    iteration_times

    def construct_solution(self, step):
        current_solution = set()
        while len(current_solution) < self.k:
            next_node = self.select_next_node(current_solution, step)
            current_solution.add(next_node)
        return current_solution

    def select_next_node(self, current_solution, step):
        unvisited = set(self.graph.nodes()) - current_solution
        alpha_t = self.get_alpha(step)
        if random.random() < self.q0:
            next_node = max(unvisited, key=lambda x: self.pheromones[x]
** alpha_t * self.heuristic(x) ** self.beta)
        else:
            weights = [self.pheromones[node] ** alpha_t * self.heuristic(
node) ** self.beta for node in unvisited]
            total_weight = sum(weights)
            probabilities = [w / total_weight for w in weights]
            next_node = random.choices(list(unvisited), weights=
probabilities)[0]
        return next_node

    def heuristic(self, node):
        # Out-degree of the node as heuristic
        if isinstance(self.graph, nx.DiGraph):
            return self.graph.out_degree(node)
        else:
            return len(self.graph[node])

    def get_alpha(self, t):
        if t <= 100:
            return 1
        elif t <= 400:
            return 2
        elif t <= 800:
            return 3
```

```
        else:
            return 4

    def simulate_influence_spread(self, influencers):
        influenced = set(influencers)
        active = list(influencers)
        while active:
            new_active = set()
            for node in active:
                neighbors = set(self.graph[node]) - influenced
                for neighbor in neighbors:
                    if random.random() < self.graph[node][neighbor]['diffusion_prob']:
                        # Use edge-specific diffusion probability
                        new_active.add(neighbor)
            influenced.update(new_active)
            active = new_active
        return len(influenced)

    def update_pheromones(self, solutions):
        for node in self.pheromones:
            self.pheromones[node] *= (1 - self.rho)
        for solution, influence in solutions:
            for node in solution:
                self.pheromones[node] += self.rho * (influence / self.best_influence)


class ACO_IM_PSO(ACO_IM):
    def __init__(self, graph, k, ants, steps, alpha, beta, rho, epsilon, q0, c1, c2, c3, phi= 0.0002, tau_max=6,
                 tau_min=0.01):
        super().__init__(graph, k, ants, steps, alpha, beta, rho, epsilon, q0)
        self.c1 = c1  # Cognitive coefficient
        self.c2 = c2  # Social coefficient
        self.c3 = c3  # Weight of previous velocity
        self.phi = phi
```

```
104         self.tau_max = tau_max
105         self.tau_min = tau_min
106         self.velocity = {node: 0.0 for node in self.graph.nodes()}
107         self.local_best_pheromone = self.pheromones.copy()
108         self.global_best_pheromone = self.pheromones.copy()
109 
110     def update_pheromones_with_pso(self, solutions):
111         for node in self.pheromones:
112             self.pheromones[node] *= (1 - self.rho)
113             self.pheromones[node] = max(self.tau_min, self.pheromones[
    node])
114         for solution, influence in solutions:
115             for node in solution:
116                 if influence > self.best_influence:
117                     self.global_best_pheromone[node] = self.pheromones[
    node]
118                 if influence > self.local_best_pheromone[node]:
119                     self.local_best_pheromone[node] = self.pheromones[
    node]
120 
121                 r1, r2 = random.random(), random.random()
122                 new_velocity = (self.c1 * r1 * (self.local_best_pheromone
    [node] - self.pheromones[node]) +
123                                 self.c2 * r2 * (self.
    global_best_pheromone[node] - self.pheromones[node]) +
124                                 self.c3 * self.velocity[node])
125 
126                 self.velocity[node] = new_velocity
127                 self.pheromones[node] += self.velocity[node]
128                 self.pheromones[node] = max(self.tau_min, min(self.
    tau_max, self.pheromones[node]))
129 
130     def run(self):
131         best_influences = []
132         iteration_times = []
133         start_time = time.time()
134         for step in range(self.steps):
135             all_solutions = []
```

```
            for ant in range(self.ants):
                solution = self.construct_solution(step)
                influence = self.simulate_influence_spread(solution)
                all_solutions.append((solution, influence))
                if influence > self.best_influence:
                    self.best_influence = influence
                    self.best_set = solution
            self.update_pheromones_with_pso(all_solutions)
            self.update_rho()  # Update rho after each iteration
            iteration_end = time.time()
            iteration_times.append(iteration_end - start_time)
            print(
                f"Iteration {step + 1}: Best Solution = {self.best_set},
    Best Influence = {self.best_influence}, time = {iteration_end -
    start_time}")
            best_influences.append(self.best_influence)
        return self.best_set, self.best_influence, best_influences,
    iteration_times

    def update_rho(self):
        if self.rho > 0.95:
            self.rho = 0.95
        else:
            self.rho *= (1 - self.phi)


def run_experiment(graph, algorithm_cls, **params):
    algorithm = algorithm_cls(graph=graph, **params)
    start_time = time.time()
    influencers, influence, best_influences, iteration_times = algorithm.
    run()
    elapsed_time = time.time() - start_time
    plot_influence_vs_iterations(best_influences, iteration_times)
    return influencers, influence, elapsed_time
```

算法 5.3: ACO-IM

```
def add_diffusion_probabilities(graph, min_prob=0.1, max_prob=0.1):
    for u, v in graph.edges():
```

```
            graph[u][v]['diffusion_prob'] = random.uniform(min_prob,max_prob)
def main():
    graph_path = 'soc-Epinions1.txt'
    G_graph = nx.read_edgelist(graph_path, create_using=nx.DiGraph())

    # Add random diffusion probabilities to each edge
    add_diffusion_probabilities(G_graph)

    aco_params = {
        'k': 10,
        'ants': 30,
        'steps': 300,
        'alpha': 1,  # Initial value
        'beta': 1,
        'rho': 0.85,  # Initial value
        'epsilon': 0.01,
        'q0': 0.7
    }

    pso_aco_params = {
        'k': 10,
        'ants': 30,
        'steps': 300,
        'alpha': 1.0,  # Initial value
        'beta': 1,
        'rho': 0.95,  # Initial value
        'epsilon': 0.01,
        'q0': 0.7,
        'c1': 0.3,
        'c2': 0.7,
        'c3': 0.3
    }

    G_aco_results = run_experiment(G_graph, ACO_IM, **aco_params)
    print("soc-Epinions1.txt Graph - ACO-IM Results:")
    print(f"Best Influencers: {G_aco_results[0]}")
    print(f"Influence Spread: {G_aco_results[1]}")
    print(f"Computation Time: {G_aco_results[2]:.2f} seconds\n")
```

```
41      G_pso_aco_results = run_experiment(G_graph, ACO_IM_PSO, **
        pso_aco_params)
42      print("soc-Epinions1.txt Graph - ACO-PSO-IM Results:")
43      print(f"Best Influencers: {G_pso_aco_results[0]}")
44      print(f"Influence Spread: {G_pso_aco_results[1]}")
45      print(f"Computation Time: {G_pso_aco_results[2]:.2f} seconds\n")
46
47
48  if __name__ == "__main__":
49      main()
```

算法 5.4: main()