# 图模式挖掘综述

詹宇 [1)]

[1)](华中科技大学计算机学院

**摘 要** 在当今数据驱动的时代，图数据已成为信息表示的一种强大工具，它在社交网络分析、生物信息学、网络安全等众多领域发挥着至关重要的作用。图模式挖掘（Graph Pattern Mining, GPM）是一项从图数据中找到特定的子图结构的计数。GPM 技术可以分为定制和通用两种类型。定制 GPM 专注于特定问题，优化算法以在特定类型的图中寻找特定模式，这种方法在准确性和效率方面往往更有优势。相反，通用 GPM 提供了更多的灵活性，能够适应多种不同的图和模式类型，但可能在特定场景下不如定制方法高效。此外，GPM 的实现方式可分为分布式和单机两种。分布式 GPM 能够处理大规模图数据，利用并行计算资源进行高效的模式挖掘，尤其适用于数据量庞大且分布在多个节点的情况。而单机 GPM 更适合中小规模数据集，它减少了网络通信的延迟，简化了系统的管理和维护。在挖掘策略上，GPM 可分为以嵌入为中心和以模式为中心两种主要方法。以嵌入为中心的方法通过将图数据嵌入到低维空间中来简化问题，这对于使用机器学习算法和进行大规模图数据分析尤为有用。然而，这种方法可能会丢失一些高维空间中的信息，并且其结果的解释性也较差。与之相对的是以模式为中心的方法，这种策略直接在原始图上操作，以发现模式和子图，它在解释性和准确性方面具有优势，但在处理非常大的数据集时可能会遇到性能瓶颈

关键词 模式挖掘；数据挖掘；知识图谱；图

## An overview of graph pattern mining

Yu Zhan[1)]

[1)](School of Computer Science, Huazhong University of Science and Technology

**Abstract** In today's data-driven era, graph data has become a powerful tool for information representation, which plays a vital role in many fields such as social network analysis, bioinformatics, cyber security, etc. Graph Pattern Mining (GPM) is a count to find specific subgraph structures from graph data.

GPM techniques can be categorized into two types: custom and generic. Custom GPM focuses on specific problems and optimizes algorithms to find specific patterns in specific types of graphs; this approach tends to be more advantageous in terms of accuracy and efficiency. On the contrary, generic GPM offers more flexibility and can accommodate many different graph and pattern types, but may not be as efficient as customized approaches in specific scenarios. In addition, GPM implementations can be categorized as distributed or standalone. Distributed GPM is able to handle large-scale graph data and utilize parallel computing resources for efficient pattern mining, which is especially suitable for cases where the data volume is huge and distributed across multiple nodes. While stand-alone GPM is more suitable for small and medium-sized datasets, it reduces the delay of network communication and simplifies the management and maintenance of the system. In terms of mining strategies, GPM can be divided into two main approaches: embedding-centric and pattern-centric. Embedding-centered approaches simplify the problem by embedding graph data into a low-dimensional space, which is particularly useful for using machine learning algorithms and performing large-scale graph data analysis. However, this approach may lose some of the information in the high-dimensional space and its results are less interpretable. In contrast, the pattern-centered approach, a strategy that operates directly on the original graph to

discover patterns and subgraphs, offers advantages in terms of interpretability and accuracy, but may encounter performance bottlenecks when dealing with very large datasets
**Key words** Pattern Mining;Data Mining;Knowledge Graph;Graph

# 1 引言

随着计算机技术的迅猛进步，人类社会产生了海量的数据。据估计，全球数据量每 20 个月就会增加一倍[1]。同时，计算机的应用领域正在不断扩展，带来了数据类型和结构的多样化。这种数据的海量性和多样性给应用系统带来了前所未有的挑战。

数据量的急剧膨胀并没有给使用者在知识获取方面带来理想中的便捷，这就产生了“数据丰富但知识匮乏”的现象。为了有效地解决这一问题，自 20 世纪 90 年代起, 数据挖掘技术开始逐渐兴起。数据挖掘，亦称作知识发现（简称 KDD），指的是从数据集中提取隐含的、有潜在价值的、人类可理解的模式[2]。通过揭示新的概念和规律，数据挖掘帮助人们进行科学分析和做出决策。自从这一概念提出以来，数据挖掘便受到了学术界和工业界的高度关注。在过去十多年中，学术界提出了众多数据挖掘的理论、技术和方法，而工业界也为各种应用领域开发了大量数据挖掘产品。如今，数据挖掘技术已被广泛应用于商业、金融、互联网、信息检索等多个领域，并取得了巨大的成功。其中一个典型的成功案例就是谷歌的 PageRank 技术[3], 该技术根据网页间的链接关系来评估网页的重要性，相比于仅评估网页内容的方法，它能够提供更高的准确度。这项技术在搜索引擎领域已得到广泛应用，影响深远。

模式挖掘（Pattern Mining）[4]是图挖掘中一项主要的任务，其主要目的是从图数据中找到特定的子图结构，这样的子图结构称为模式（Pattern）。用户通常只对某种类型的模式感兴趣，而不是整个模式集，即只是所有频繁子图集合的某些子集. 这种“特殊模式”是根据模式的性质或某些特定的约束(限制)判断.

在社交网络分析、生物信息学、化学结构分析等领域，图模式挖掘能有效地揭示隐藏在复杂网络结构中的重要特征和关系。通过分析用户行为的图结构，图模式挖掘可以用于提升推荐系统和广告的精准度和效果。在金融和安全领域，图模式挖掘能够识别异常模式，帮助检测欺诈行为。

# 2 定制和通用图模式挖掘

## 2.1 定制图模式挖掘

早期的研究关于图模式挖掘主要着眼于通过特定的算法改进来满足特定需求。例如，kClist 算法[5]以一种创新的方式采用了基于核心值的方法，专门为处理稀疏图中的团块计数设计了一种高效算法。

在团块计数任务中，Chiba 和 Nishizeki 提出的经典算法[6]一直被认为是解决问题的最有效方法之一。然而，原始算法存在顺序依赖性，这限制了并行化实现的效率。为了改进这一点，新的算法采用了 Node-Parallel 和 Edge-Parallel 的变体设计，以及每个线程处理一个子图的方法，从而使不同部分能够独立并行处理，提高了算法的并行性能。

原始算法按照非递增度的顺序处理节点，虽然这种方法简化了运行时间的分析，但在实际操作中可能不是最佳选择。kClist 算法允许接受来自图中节点的任何总排序方式 $\eta$ 。根据分析和实验，核心排序 (core ordering) 被认为是性能较好的排序方式，但新算法也考虑了其他可能的节点排序策略。

此外，新算法还考虑了输入图的有向版本，以引导避免不必要的计算，这有助于减少操作次数，从而提高整体算法的效率。

总之，这个新算法不仅在理论上提供了对于稀疏图的最佳渐近运行时间上界，而且在实验中表现出比现有最先进算法更快的性能，同时具备出色的并行性。

“网络模式”，最早由 Milo 等人于 2002 年提出[7]。寻找网络模式（Network Motif Discovery）与发现频繁子图（Frequent Subgraph Mining）在图数据分析领域是两个不同的任务。

发现频繁子图的目标是查找在一组图中经常出现的子图，这些子图可能是网络中的常见组成部分，但没有对其重要性的假设。这个任务旨在找到在给定图集中频繁出现的子图。

寻找网络模式的目标是发现网络中的特定模式或子结构，这些模式被认为在网络中反复出现，具有某种重要性或意义。网络模式通常用于识别网络的重要功能或特性。

在网络模式分析的领域，现有的子图探索方法主要分为两种截然不同的途径：一是枚举出所有特定大小的子图，随后确定这些子图之间的同构关系[8, 9]；二是根据已知的目标子图（例如，通过生成所有可能的特定大小子图），执行专门的搜索过程，从而单独计算每个子图出现的频次[10]。

这两种策略各有利弊。第一种方法涉及到对图中所有子图的全面考虑，即便在随机图中，我们的主要关注点仍是原始网络中出现的子图频率。这可能导致我们花费大量时间探索那些我们并不关心的子图。而第二种方法则聚焦于单一子图，但在同一图部分对相同类型子图的重复搜索可能造成时间浪费。

G-tries 的主要创新在于提出了一种介于这两种方法之间的新型数据结构。通过事先确定我们感兴趣的子图集合（通过对原始子图进行普查计算），G-tries 将这些子图集合储存在一种树形数据结构中，这一结构充分利用了常见的拓扑特征。G-tries 识别并利用共有的祖先树节点来表示共同的子结构。随后，G-tries 运用这棵树来高效计算随机图中子图的频率，同时避免了因图的自同构带来的对称性问题。当我们构建出部分子图时，我们就能确定哪些子图集合仍是符合这一特定子图的潜在候选，而当该子图构建完成时，我们已经可以确认它与我们所寻找的子图是同构的。我们将这种创新的数据结构命名为 g-tries。

这种方法显著压缩了具有共同子结构的子图集合的表征，并且在计算其他较大图中所有子图频率时，相比以往的方法表现出了显著的优势。

### 2.2 通用图模式挖掘

与特定算法的改进相反，一些研究更关注在更一般情况的数据挖掘,比如 Cyclosa[11]。

在之前的研究中，许多创新技术被提出来缩减搜索空间和探索的部分实例数量。例如，AutoMine[12]引入了一种便捷的编译器，它能够自动生成优化的匹配和计算顺序，这对减小搜索空间产生了显著影响。

然而，Cyclosa 发现即便在这样优化的搜索空间中，进行集合操作时，依然面临大量固有的重复计算问题。具体来说，这些重复计算分为两类：显式冗余和隐式冗余。显式冗余发生在一个集合交集被用来重复计算连接到相同子图实例的不同模式顶点。而隐式冗余则是指同一交集在处理不同子图实例时的重复出现。这些重复计算通常占据了超过80%的运行时间，极大地影响了性能。

更复杂的是，无论是在挖掘单一模式还是多个模式时，这些显式和隐式冗余都遍布于整个运行过程中，使得它们难以被追踪和重复利用。现有系统虽然遵循结构相等原则，并尝试通过将不同顶点的相同集合公式统一为单一公式来减少部分显式冗余[13-15]，但还是有超过 60%的冗余问题未能得到有效解决。

Cyclosa 在集合操作中观察到两种计算相似性，为识别和重用显式和隐式冗余提供了机会。第一种相似性被称为静态相似性，它揭示了模式中集合操作的不同操作数之间存在的结构类似性。具体来说，这意味着一个输入操作数能够在多个操作符中复用，而这些操作符产生的输出结果可能包含潜在的重复内容。静态相似性可以在执行前进行分析。Cyclosa 提出了一种解耦所有集合操作的操作数和操作符的方法，并将它们重新组织成一个有向流，Cyclosa 称之为集合数据流。这个数据流是由输入与输出之间的联系所引导的。集合数据流中的每个节点代表一个集合操作数或操作符。通过在数据流中仅保留独一无二的操作数，Cyclosa 可以消除显式冗余。与此同时，通过标识数据流中输入源节点的重叠，Cyclosa 可以指出不同操作符之间的隐式冗余。

第二种相似性，动态相似性，涉及到在运行时发生的所有集合交集输入之间的相似性，这需要在执行后进行分析。具体而言，Cyclosa 发现大多数计算集中在少量高度参与的顶点上。这一现象意味着冗余计算主要发生在这些顶点。因此，我们可以通过追踪这些高参与度的顶点来缓存和重用隐式冗余的结果。

Cyclosa 展现了几个显著的特性。首先，它引入了一种创新的集合数据流表达和高效的构建方法，用于为任意模式生成集合数据流。Cyclosa 采用了轻量级的成本估计模型来保持优化过程的成本，并引入了冗余概率来指导计算结果的有效重用。

其次，Cyclosa 开发了一种基于数据流的执行模型，该模型在运行时揭示了捕捉和重用冗余的潜力。通过这种数据流驱动的执行方式，数据访问和

计算过程得到了解耦，从而最大化了并行数据重用的潜能。

然后，Cyclosa 还设计了一个内存优化的数据管理基础设施，自动存储具有高冗余概率的计算结果。它利用评估出的结果重用概率来实现智能缓存策略，有效控制内存消耗。此外，这一基础设施能够与数据流执行引擎高效协同工作，确保重复计算所需结果的可用性。

以前的优化也可以整合到 Cyclosa 中。

# 3 分布式和单机图模式挖掘

## 3.1 分布式图模式挖掘

分布式图模式挖掘系统在处理大规模数据时有出色的能力。随着数据量的不断增加，分布式系统能够轻松地通过增加额外的节点来扩展其处理能力，而无需替换现有的硬件设备。它具备动态调整资源的能力，可以根据实际需求来灵活分配和回收资源，以适应不同的工作负载。

分布式图模式挖掘系统通过并行处理数据的方式，显著提高了计算效率。即使在某个节点发生故障的情况下，系统仍能够继续运行，得益于其出色的容错性，这进一步提高了整体可靠性。此外，数据的多节点备份策略使得故障恢复变得更加简便和快速。

与高性能的单机服务器相比，分布式系统通常能够以更低的成本实现相同的任务。此外，它还能够更有效地管理能源消耗，使其成为一个经济实惠而高效的选择。

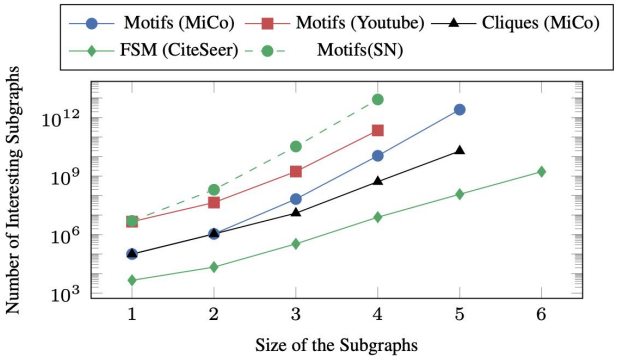


图 1：在不同数据集上，图挖掘问题（模式计数、团查找、FSM：频繁子图挖掘）中中间状态的指数增长

将图模式挖掘算法扩展到中等规模的图形非常困难。在图形中，可能出现的模式及其子图的集合会随着原始图形的大小呈指数级增长，从而导致计算和中间状态的急剧增加。图 1 展示了在 Arabesque[16]文中评估的一些图挖掘问题和数据集中不同大小的“有趣”子图数量呈指数增长的情况。即使是具有少量边的图形也可能快速生成数以亿计的有趣子图。

Arabesque 是专为分布式图挖掘而设计的第一个嵌入式探索系统。从概念上讲，Arabesque 从“像一个顶点思考”（TLV）[17]的范式转移到“像一个嵌入思考”（TLE），在这里，嵌入指的是表示更一般的模式子图实例的子图。

Arabesque 定义了一个高级的过滤-处理计算模型。给定一个输入图形，系统会自动而有序地访问所有需要由用户定义的算法探索的嵌入，以分布式方式执行这一探索。系统将其所探索的所有嵌入传递给应用程序，主要包括两个功能：filter，用于指示是否应处理一个嵌入，以及 process，用于检查一个嵌入并可能生成一些输出。例如，在查找团块的情况下，filter 功能将修剪不是团块的嵌入，因为它们的任何扩展都不可能是团块，并且 process 功能将输出所有已探索的嵌入，这些嵌入在构建时都是团块。Arabesque 还支持根据用户定义的指标跨多个嵌入进行探索空间的修剪。

Arabesque 的嵌入中心 API 有助于实现高度可扩展性。该系统通过在工作节点之间均匀分布嵌入来实现扩展，从而避免了热点问题。通过明确表示嵌入是探索的基本单位，Arabesque 能够使用基于嵌入规范性的快速无协调技术，以避免冗余工作并减少通信成本。它还使我们能够使用一种名为“过估计有向无环图”（ODAG）的新型数据结构高效存储嵌入，并为基于模式的聚合设计了新的两级优化，这是图挖掘算法中常见的操作。

## 3.2 分布式图模式挖掘

尽管分布式系统在处理大规模数据和复杂计算任务方面具有显著的优势，但单机系统仍然具有一些独特的优点和好处。

首先，单机图模式挖掘系统在系统管理方面更加简化和易于处理。它们通常不需要复杂的网络配置或多节点协调，因此更容易设置和维护。相对于分布式系统，单机系统的硬件和软件复杂性也更低。

其次，单机系统在成本效益方面具有明显的优势。对于中小规模的数据处理任务来说，单机系统的初始投资通常远低于建立完整分布式处理环境所需的费用。此外，由于其管理和维护的简便性，

单机系统在长期运行中可能会带来更低的运维成本。

然后，单机系统在性能优化方面也有其独特之处。由于所有数据和计算资源都位于同一物理位置，单机系统能够更有效地利用缓存和内存，从而减少数据访问的延迟。此外，它们避免了分布式计算中常见的网络通信开销，对于不需要大规模并行处理的任务来说，这可能意味着更高的效率。

数据一致性和同步是另一个重要方面。单机系统不需要涉及跨多个节点的数据复制和同步，因此更容易维护数据的一致性。同时，不需要处理分布式环境中常见的数据版本控制和同步问题。

RStream 是基于 X-Stream[18]的单机图挖掘系统。它仅支持基于边诱导的嵌入探索。在解决一些基于顶点的应用程序时，如模式计数和团块发现，它需要更多的迭代和更多的磁盘 I/O。

为了探索所有可能的嵌入，RStream 执行了关系代数中的全连接操作，这会产生大量的中间数据。例如，在 MiCo（100K 个顶点，1.1M 条边）[19]上运行 RStream 的 4-模式计数会产生约 1.64TB 的中间数据，而 MiCo 中的 4-模式数量约为 110 亿。

Arabesque 和 RStream 都使用了一个图形库，bliss[20], 这是一个用于计算图同构问题的开源工具。Bliss 构建了一个搜索树，并为每个模式计算哈希值。如果两个模式包含相同的哈希值，则它们是自同构的。然而，构建搜索树会频繁分配和释放内存，从而减慢了模式哈希处理的速度，并消耗了大量内存。例如，在运行 3-FSM（37 个标签）的专利图上（支持 1），分配和释放内存的开销超过了 53%，总共消耗了 16.1GB 内存，用于处理 25083 个模式。

Kaleido[21]是一个单机外存图挖掘系统，受到 Arabesque 的启发。它以嵌入为核心的计算模型为基础，提供了通用的编程 API，可适用于大多数图挖掘应用。每一次基于顶点的嵌入扩展，实际上都相当于将中间数据升级一个维度。因此，Kaleido 将第 i 次探索产生的中间数据看作是一个 i 维的张量。受到稀疏矩阵中的压缩稀疏列（CSC）[22]启发，Kaleido 设计了一种逐级简洁的中间嵌入数据结构，它被命名为压缩稀疏嵌入（CSE）。每一轮探索都会升级中间数据并扩展 CSE 的层级。

在图挖掘问题中，由于中间数据呈指数级增长，计算大型嵌入变得异常困难。例如，在一个小型图 CiteSeer（3.3K 个顶点，4.7K 条边）上进行 7 轮嵌入探索, 嵌入的大小分别为 3.3K、4.5K、24.5K、352.2K、7.7M、168.2M、3.5G。Kaleido 的关注点则集中在高效解决小嵌入（少于 9 个顶点）中的图同构问题上。

基于同构特性, Kaleido 解释了同构嵌入具有相同的特征值。为了应对标记图同构问题，Kaleido 将每个模式中的顶点标签信息、度数信息以及邻接矩阵的特征值结合起来，以检查嵌入同构性。为了应对输入图规模或探索深度增加时的中间数据存储需求，Kaleido 设计了一种混合存储方案，将 CSE 的大型中间数据层级存储在磁盘上。为了平衡磁盘上的中间数据处理负载, Kaleido 预测下一轮迭代中嵌入候选的容量，并根据预测将探索任务均匀分配给各个线程。在处理磁盘上的中间数据时，Kaleido 采用了滑动窗口策略，以确保并行计算性能，并减轻 I/O 开销的影响。

# 4 嵌入为中心和模式为中心图模式挖掘

## 4.1 嵌入为中心图模式挖掘

图模式挖掘系统主要分为两种，一种常见的方法是列举所有子图（通常在一定深度下），以检查子图是否满足模式约束，这被称为以嵌入为中心的范式[16, 23]。这种方法易于开发和并行化。然而，它导致高内存消耗和由于大量中间部分实例而造成的计算资源浪费[12, 24]。例如上文中提到的 Arabesque 和 RStream。

## 4.2 模式为中心图模式挖掘

最近，先进的图模式挖掘系统采用了以模式为中心的范式来克服效率低下的问题[17]。其主要思想是使用图模式的结构信息来过滤不会导致正确最终匹配的中间体。这是通过将图模式转换为一系列集合操作（如交集、并集等）并按照模式顶点的匹配顺序在嵌套循环中执行来实现的。

这一方式由 AutoMine [12] 和 Peregrine [24]开创，通过模式约束的引导，避免了不必要的嵌入存储和处理。这些系统的核心是高效的匹配引擎，执行快速的集合操作[25]。

GraphZero [26]是 AutoMine 的增强版本，通过引入破除对称性的优化，定义了对称顶点之间的偏序关系，以消除多个模式之间的显式冗余。

GraphZero 首先认识到子图匹配中的计算冗余与查询模式内的对称性相关。这种冗余通过自同构来体现，自同构是在模式内保持其结构的映射。为了打破这种对称性，GraphZero 引入了一种方法，对模式中的顶点 ID 施加限制。例如，系统可能要求一个匹配的顶点的 ID 高于另一个。这些限制旨在减少有效自同构的数量，有效地限制了冗余。论文提出了生成这些限制的算法。它计算了模式的自同构群，并迭代每个顶点，建立了基于它们的 ID 的二元关系（如“大于”）。这些关系用于限制图中匹配顶点的方式，确保每个实例都被精确计数一次。该方法得到了理论证明的支持，证明了这些限制确保了每个模式实例的独特和高效映射，从而消除了冗余。

GraphZero 开始说明了，对于相同模式的不同调度，其性能可能存在显著差异。随后，引入了一种方法来确定有效的调度方式，其中每个顶点，除了第一个顶点外，都必须直接与至少一个在其之前的顶点相连。其中的关键洞察在于，某些调度方式在它们执行的集合操作序列方面是等效的。因此，我们的目标是辨认出这些等效的调度方式，并仅考虑每个等效组中的一个。

这一方法运用了群论中的自同构原理，将潜在的下一个顶点分为不相交的集合。在每个集合中，无论选择哪个顶点进行调度扩展，结果都会导致等效的调度。这种方式，使得我们避免了对所有排列方式的穷尽搜索，而只需关注不同的调度。

最后，我们采用递归方法生成各种不同的调度方式，并通过理论证明确保了该方法能够生成所有不同的调度方式，不会漏掉任何等效的调度。

GraphZero 还将基于方向的优化泛化到以前仅适用于团块模式的任意模式，增强了在更广泛应用领域的性能。

DecoMine [27]系统通过一个多步骤的过程，将复杂的图模式分解成更简单的子模式，并高效地挖掘这些子模式。首先，它通过识别一个顶点切割集来将复杂的图模式分解成较小的子模式。去除这些顶点将模式分割成多个连接组件，然后与切割集结合以形成更简单的子模式。然后系统枚举与切割集匹配的嵌入。对于这些嵌入中的每一个，它计算子模式嵌入的数量。这些子模式嵌入在概念上联合在一起，以确定完整模式嵌入的数量。最后消除无效嵌入，这一步涉及识别和消除无效的嵌入。当有效的子模式嵌入联合起来不能形成整个模式的有效嵌入时，就会出现这种情况。

**Algorithm 1** DecoMine Algorithm Template

```
1: Input: Pattern p, Cutting Set V_C, Matching orders of all subpatterns
2: Decompose p using V_C to generate K subpatterns, and the shrinkage patterns
3: pattern_cnt ← 0
4: for all e_C = (v_0, v_1, ..., v_{|V_C|-1}) matching the cutting set V_C do
5:     num_shrinkages ← new HashTable()
6:     clear(num_shrinkages)
7:     M ← 1
8:     for all i ← 1...K do
9:         M_i ← the number of pe extending e_C and matching the i-th subpattern
10:        M ← M × M_i
11:    pattern_cnt ← pattern_cnt + M
12:    for all e extending e_C and matching one of the shrinkage patterns do
13:        pattern_cnt ← pattern_cnt − 1
14:        for all i ← 1...K do
15:            pe_i ← extract_subpattern_embedding(e, i)
16:            num_shrinkages[pe_i] ← num_shrinkages[pe_i] + 1
17:    for all i ← 1...K do
18:        for all pe extending e_C and matching the i-th subpattern do
19:            count ← M/M_i − num_shrinkages[pe]
20:            if count > 0 then
21:                process_partial_embedding(pe, count)
```

图 2：DecoMine 算法模板

图 2 展示了一种算法模板，编译器可利用此模板为顶点割集 $V_C$ 和匹配顺序的组合生成具体的图模式分解算法。匹配顺序定义为($o_{vc}$,$o_1$, ..., $o_k$, $o_{s1}$, ..., $o_{sn}$)，其中 $o_{vc}$ 为第 4 行中使用的 $V_C$ 的匹配顺序；每个 $o_i$ 确定了从 $V_C$ 的匹配扩展到第 i 个子模式（共 k 个）的顺序，如第 9 行所用；每个 $o_{si}$ 确定了扩展到第 j 个收缩模式（共 n 个）的顺序，如第 12 行所用。当应用程序使用 process_partial_embedding 时，该模板便生成算法。对于模式计数，模式计数将在不执行第 14 至 21 行的情况下返回。对于每个 $e_c$，在迭代中执行三个计算步骤（第 6 至 21 行）。

## 参 考 文 献


[1] S. Salamone. Data Storage Trends Require New Management Solutions. http://m.zdnet.com.au/120265411.htm.

[2] Advances in knowledge discovery and data mining[C]. American Association for Artificial Intelligence, 1996.

[3] S. Brin, L. Page. The Anatomy of a Large-scale Hypertextual Web Search Engine[J].Computer Networks and ISDN Systems, 1998, 30(1-7):107–117.

[4] 刘勇.图模式挖掘技术的研究[D].哈尔滨工业大学,2010.

[5] Danisch M, Balalau O, Sozio M. Listing k-cliques in sparse real-world graphs[C]//Proceedings of the 2018 World Wide Web Conference. 2018: 589-598.

[6] Chiba N, Nishizeki T. Arboricity and subgraph listing algorithms[J]. SIAM Journal on computing, 1985, 14(1): 210-223.

[7] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U.

Alon. Network motifs: simple building blocks of complex networks. Science, 298(5594):824–827, October 2002.

[8] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan,D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. Science,298(5594):824–827, October 2002.

[9] S. Wernicke. Efficient detection of network motifs. IEEE/ACM Trans. Comput. Biol. Bioinformatics, 3(4):347–359, 2006.

[10] J. Grochow and M. Kellis. Network motif discovery using subgraph enumeration and symmetry-breaking. pages 92–106. 2007.

[11] Gui C, Liao X, Zheng L, et al. Cyclosa:{Redundancy-Free} Graph Pattern Mining via Set Dataflow[C]//2023 USENIX Annual Technical Conference (USENIX ATC 23). 2023: 71-85.

[12] Mawhirter D, Wu B. Automine: harmonizing high-level abstraction and high performance for graph mining[C]//Proceedings of the 27th ACM Symposium on Operating Systems Principles. 2019: 509-523.

[13] Sporns O, Kötter R. Motifs in brain networks[J]. PLoS biology, 2004, 2(11): e369.

[14] Wasserman S, Faust K. Social network analysis: Methods and applications[J]. 1994.

[15] Watts D J, Strogatz S H. Collective dynamics of ‘small-world’networks[J]. nature, 1998, 393(6684): 440-442.

[16] Teixeira C H C, Fonseca A J, Serafini M, et al. Arabesque: a system for distributed graph mining[C]//Proceedings of the 25th Symposium on Operating Systems Principles. 2015: 425-440.

[17] Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large-scale graph processing[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. 2010: 135-146.

[18] Roy A, Mihailovic I, Zwaenepoel W. X-stream: Edge-centric graph processing using streaming partitions[C]//Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. 2013: 472-488.

[19] Elseidy M, Abdelhamid E, Skiadopoulos S, et al. Grami: Frequent subgraph and pattern mining in a single large graph[J]. Proceedings of the VLDB Endowment, 2014, 7(7): 517-528.

[20] Junttila T, Kaski P. Engineering an efficient canonical labeling tool for large and sparse graphs[C]//2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX). Society for Industrial and Applied Mathematics, 2007: 135-149.

[21] Zhao C, Zhang Z, Xu P, et al. Kaleido: An efficient out-of-core graph mining system on A single machine[C]//2020 IEEE 36th International Conference on Data Engineering (ICDE). IEEE, 2020: 673-684.

[22] Saad Y. Iterative methods for sparse linear systems[M]. Society for Industrial and Applied Mathematics, 2003.

[23] Dias V, Teixeira C H C, Guedes D, et al. Fractal: A general-purpose graph pattern mining system[C]//Proceedings of the 2019 International Conference on Management of Data. 2019: 1357-1374.

[24] Jamshidi K, Mahadasa R, Vora K. Peregrine: a pattern-aware graph mining system[C]//Proceedings of the Fifteenth European Conference on Computer Systems. 2020: 1-16.

[25] Besta M, Vonarburg-Shmaria Z, Schaffner Y, et al. Graphminesuite: Enabling high-performance and programmable graph mining algorithms with set algebra[J]. arXiv preprint arXiv:2103.03653, 2021.

[26] Mawhirter D, Reinehr S, Holmes C, et al. Graphzero: A high-performance subgraph matching system[J]. ACM SIGOPS Operating Systems Review, 2021, 55(1): 21-37.

[27] Chen J, Qian X. DecoMine: A Compilation-Based Graph Pattern Mining System with Pattern Decomposition[C]//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1. 2022: 47-61.

## 附录 研讨课问题及意见

问题：图中检测三角形的经典办法

回答:

1. Schank T, Wagner D. Finding, counting and listing all triangles in large graphs, an experimental study[C]//International workshop on experimental and efficient algorithms. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005: 606-609.

**Algorithm 1**: *forward*
**Input**: ordered list (high degree first) of vertices $(1,\ldots,n)$; Adjacencies $Adj(v)$
**Data**: Node Data: $A(v)$;
**for** $v \in V$ **do**
  $A(v) \leftarrow \emptyset$
**for** $s \in (1,\ldots,n)$ **do**
  **for** $t \in Adj(s)$ **do**
    **if** $s < t$ **then**
      **foreach** $v \in A(s) \cap A(t)$ **do**
        output triangle $\{v,s,t\}$ ;
      $A(t) \longleftarrow A(t) \cup \{s\}$;

这个算法是一个用于在图中计数三角形的算法，称为“forward”。它的输入是一个顶点列表，这些顶点按照度从高到低排序。每个顶点 v 都有一个邻接表 Adj(v)，即与 v 直接相连的顶点列表。

算法大致流程如下：

初始化：为图中的每个顶点 v 初始化一个空的集合 A(v)。这个集合用于存储可能构成三角形的顶点。

外层循环：遍历顶点列表，对于每个顶点 s，按顺序执行以下步骤：

内层循环：遍历顶点 s 的邻接表中的每个顶点 t。

如果 t 的顺序号大于 s（即 t 出现在 s 之后），则继续；这是为了确保每个三角形只被计数一次。

查找交集：对于邻接表中的每个顶点 t，找出集合 A(s) 和 A(t)的交集。这个交集包含了与 s 和 t 都直接相连的顶点 v，因此{s,t,v}形成了一个三角形。

输出三角形：当找到构成三角形的三个顶点时，输出三角形{s,t,v}。

更新集合：在集合 A(t)中加入顶点 s。这意味着顶点 t 已经与顶点 s 相连，任何将来发现与 t 相连的顶点都有可能与 s 形成新的三角形。

通过这种方式，算法有效地遍历图中的每个顶点，查找并计数所有可能的三角形。这种方法利用了顶点的度来优化搜索过程，并减少了不必要的比较，因为它只考虑顺序在当前顶点之后的顶点，从而避免了重复计数同一个三角形。

2. Tsourakakis C E. Fast counting of triangles in large real networks without counting: Algorithms and laws[C]//2008 Eighth IEEE International Conference on Data Mining. IEEE, 2008: 608-617.

本文提供了 EigenTriangle 算法， 用于计算图中总三角形的数量

**Algorithm 1** The EIGENTRIANGLE algorithm
**Require:** Adjacency matrix $A$ $(nxn)$
**Require:** Tolerance $tol$
**Output:** $\Delta'(G)$ global triangle estimation
  $\lambda_1 \leftarrow LanczosMethod(A,1)$
  $\vec{\Lambda} \leftarrow [\lambda_1]$
  $i \leftarrow 2$ {initialize $i$, $\vec{\Lambda}$}
  **repeat**
    $\lambda_i \leftarrow LanczosMethod(A,i)$
    $\vec{\Lambda} \leftarrow \left[\vec{\Lambda}\ \lambda_i\right]$
    $i \leftarrow i+1$
  **until** $0 \leq \frac{|\lambda_i^3|}{\sum_{j=1}^{i}\lambda_j^3} \leq tol$
  $\Delta'(G) \leftarrow \frac{1}{6}\sum_{j=1}^{i}\lambda_j^3$
  **return** $\Delta'(G)$

以下是算法的步骤：

初始化：使用邻接矩阵 A 和 Lanczos 方法计算最大特征值λ1，并将其作为初始值放入特征值估计向量Λ^Λ^中。初始化迭代变量 i 为 2。

迭代计算：通过 Lanczos 方法迭代计算邻接矩阵 A 的下一个特征值λi，并更新特征值向量Λ^，在每次迭代后将 i 递增 1。

终止条件：迭代直到第 i 个特征值的立方除以前 i 个特征值的立方之和的绝对值小于或等于容忍度 tol。这个条件检查新计算的特征值的重要性是否足够小，如果是，算法就会停止。

计算三角形数量：一旦算法停止，使用累积的特征值来估计三角形的数量。这是通过计算所有获得的特征值的立方之和的 1/6 来实现的，因为每个三角形在无向图中被计数了 6 次。

输出结果：返回估计的全局三角形数量Δ′(G)。

总结来说，这个算法是一个基于图的谱性质（特征值）来估计三角形数量的方法。通过计算和累积图的邻接矩阵的主要特征值的立方，它可以推断出图中三角形的数量。这种方法在计算上比直接枚举三角形要高效得多，特别是在处理大型稀疏图时。