

大规模二分图中 bi-triangle 的分图枚举优化研究

朱星坡¹, 王晓阳²

(1. 复旦大学软件学院, 上海 200433; 2. 复旦大学计算机学院, 上海 200433)

摘要: 二分图中 bi-triangle 的枚举是局部聚类系数(CC)计算等图分析任务的核心操作。随着实际二分图数据规模不断扩大,其数据量已超出单机处理能力,亟需依托分布式算法实现高效枚举。然而,现有分布式分图枚举算法 GP 存在子图组合数据量庞大、消息过载及重复枚举等问题。对此,基于 bi-triangle 拓扑特性定制分图策略,提出两种优化算法。方法 1 将 bi-triangle 视为由 3 个 wedge 结构组成,以 wedge 组为基本单位生成子图,并提出基于 A 型和 V 型 wedge 组拼接的子图组合构造机制,显著减少子图组合的数量和数据规模,最终以 wedge 三元组枚举 bi-triangle。此外,为解决消息过载和重复枚举问题,方法 1 提出基于分布式存储系统的子图读取机制和顶点有序性的去重机制。方法 2 将 bi-triangle 视为由 2 个 zedge 结构组成,先以 wedge 组为基本单位执行第一次分图,再通过“压缩 zedge 组”的构造与还原机制完成第二次分图,最终以 zedge 二元组枚举 bi-triangle,实现比方法 1 更低阶的计算复杂度。实验结果表明,与 GP 算法相比,方法 1 平均只需处理 0.5% 的子图组合数据,所用时间不到原来的 2.2%;方法 2 平均处理约 3.3% 的数据量,时间压缩至原来的 1.0% 以内。

关键词: 二分图; 6-环; 分布式分图枚举; wedge 组; zedge 组

中图分类号: TP391

文献标志码: A

DOI: 10.19678/j.issn.1000-3428.0252080

Optimization Research on Graph Partitioning Enumeration of bi-triangle in Large Bipartite Graphs

ZHU Xingpo¹, WANG Xiaoyang²

(1. School of Software, Fudan University, Shanghai 200433, China;

2. School of Computer Science, Fudan University, Shanghai 200433, China)

【Abstract】 Bi-triangle enumeration in bipartite graphs is a core operation for graph analysis tasks such as local Clustering Coefficient (CC) computation. As the scale of real-world bipartite graph data continues to increase, the data volume has exceeded the processing capacity of a single machine, necessitating efficient enumeration through distributed algorithms. However, existing distributed graph partitioning algorithm, called GP, suffer from issues such as large subgraph combination data volume, message overload, and redundant enumeration. To address these problems, two optimization algorithms based on the topological characteristics of bi-triangles are proposed and the graph partitioning strategies are customized accordingly. Method 1 treats a bi-triangle as composed of three wedge structures and generates subgraphs using wedge groups as the basic unit. It introduces a subgraph combination construction mechanism based on the concatenation of A-type and V-type wedge groups, significantly reducing the number and data size of subgraph combinations, and ultimately enumerates bi-triangles via wedge triplets. Additionally, to address message overload and redundant enumeration, Method 1 proposes a subgraph-reading mechanism based on a distributed storage system and a deduplication mechanism based on vertex ordering. Method 2 considers a bi-triangle as composed of two zedge structures. It first performs initial graph partitioning using wedge groups as the basic unit, then completes the second partitioning through a construction and restoration mechanism of "compressed zedge groups", ultimately enumerating bi-triangles via zedge pairs, achieving lower computational complexity than Method 1. Experimental results demonstrate that, compared with the GP algorithm, Method 1 processes only 0.5% of the subgraph combination data on average and the enumeration time is less than 2.2% of the original, whereas Method 2 processes approximately 3.3% of the data on average and the time is compressed to within 1.0% of the original.

【Key words】 bipartite graph; bi-triangle; distributed graph partitioning enumeration; wedge group; zedge group

基金项目: 复旦大学校企合作项目(SGH2301326)。

作者简介: 朱星坡,男,硕士,主研方向为图计算优化;王晓阳,教授、博士。

收稿日期: 2025-01-21

修回日期: 2025-03-27

E-mail: xinpzhu18@fudan.edu.cn

0 引言

在图的相关研究中,二分图因其顶点集的特殊性而备受关注^[1-2]。由于顶点集可以划分为两个互不相交的子集,且所有边仅连接不同子集中的顶点,因此二分图尤其适合建模现实中不同属性对象之间的关系网络^[3-4],如蛋白质与基因的相互作用网络、作者与文章的合著关系网络、用户与商品的购买行为网络等,广泛应用于疾病分析^[5-6]、文档聚类^[7-8]以及推荐系统^[9-10]等领域。

在二分图研究中,结构件的计数和枚举^[11-13]是一项关键工作,该任务有助于分析网络顶点的相关性,揭示网络功能与架构^[14]。bi-triangle(6-环,二分图 2 个顶点集各提供 3 个顶点)是二分图中的重要结构件,用于评估二分图同一顶点集内 3 个顶点之间的关联紧密度^[15-17],在多种二分图相关的实际场景中广泛存在。

本文的研究目标是在大规模二分图中实现 bi-triangle 的高效分布式枚举。动机来源于两方面:

一是随着移动互联网、云计算、人工智能大模型等技术的进步,各类联网应用不断积累数据,导致现实中二分图的规模持续扩大,超过了单机存储和处理能力的上限,需借助分布式计算集群实现高效存储和处理。

二是枚举 bi-triangle 在二分图结构挖掘和理论分析任务中具有关键作用。以下是两个典型例子:

Truss 子图计算:文献[18]研究了异构信息网络(HIN)的 Truss 子图计算问题,提出了 k -Btruss 和 k -Ctruss 两种模型及其对应求解算法。理论分析和实验结果表明, k -Ctruss 模型具有更强的内聚性,能更准确地对 HIN 的稠密程度进行刻画。在 HIN 为二分图的情况下,文献[18]中设计的 k -Ctruss 计算算法的第 1 步需要枚举出图中全部的 bi-triangle,从而为后续算法的构建提供基础。

聚类系数(CC)计算:CC 是衡量图顶点集集成团程度的量化评估指标。文献[15]基于 bi-triangle 提出了计算二分图局部 CC 的方法:对于二分图的某顶点 u ,设 B_u 表示包含 u 的 bi-triangle 的数量, L_u 表示二分图中长度为 4 且包含 u 的路径总数,则局部聚集系数 $CC(u) = (3 \times B_u) / L_u$ 。显然,计算 $CC(u)$ 的关键步骤是准确枚举包含顶点 u 的所有 bi-triangle。

针对 bi-triangle 的分布式枚举问题,可参考已有成熟算法设计出两种解决方案。

第一种是基于图转换的枚举算法。该算法首先

将二分图转换为简单图,使 bi-triangle 转化为简单图中的带权三角形,随后采用简单图的三角形分布式枚举算法^[19-21]进行处理。然而,这种转换会导致图规模呈指数级增长^[22],极易引发内存溢出,显著增加存储和计算成本,因而在实际应用中缺乏实用性,优化空间极为有限。

第二种是分图枚举算法 GP。GP 算法的处理过程包括分图与枚举两个阶段。在分图阶段,原始的大规模图数据被随机划分为 p 份($p > 6$),并通过跨节点的消息传递构建有限的子图组合,以确保覆盖原始大图中所有可能的 bi-triangle。该消息传递机制已广泛应用于多种图结构的分布式解析任务中^[23-25]。在枚举阶段,每个子图组合调用基于邻接点集的 bi-triangle 枚举算法^[18]来枚举 bi-triangle。

然而,对于 bi-triangle 的分布式枚举任务,GP 算法作为解决方案存在以下问题:

1)高构造开销:bi-triangle 由 6 条边构成,为避免枚举遗漏,GP 算法构造的子图组合数量高达 $C(p, 6)$,而总数据量是原始图网络大小的 $C(p-1, 5)$ 倍。由于该算法依赖节点间的信息传递机制生成子图组合,对应的数据传输量也处于同一数量级。如此大规模的消息传递容易导致消息队列拥塞、缓存溢出等消息过载现象,这种情况在大规模二分图场景中尤为明显。

2)冗余枚举:每个子图组合代表一个枚举任务,但在 $C(p, 6)$ 个子图组合中,大量组合不包含任何 bi-triangle,导致无效枚举。此外,同一个 bi-triangle 可能存在于多个子图组合中,由于各子图组合的处理过程相互独立,且缺乏去重机制,容易造成重复枚举。

GP 算法在分图阶段采用随机划分策略,导致子图组合的数量与规模过于庞大,这是产生上述性能瓶颈的主要原因。对此,本文提出 WP(wedge-based Graph Partitioning bi-triangle Enumeration)和 ZP(zedge-based Graph Partitioning bi-triangle Enumeration)两种优化算法,两者基于 bi-triangle 的拓扑结构特性,设计了定制化的分图策略,从而减少子图组合的数量和规模,实现更加高效的 bi-triangle 分布式枚举。

WP 算法将 bi-triangle 视为由 3 个 wedge 结构组成,枚举时以 wedge 组为基本单位,生成 A 型与 V 型 wedge 组子图,并将其存储至分布式存储系统。在子图组合构造阶段,计算节点直接从存储系统读取子图,无需消息依赖,实现数据解耦合,从根源上消除消息过载问题。子图组合的构造通过拼接

A 型与 V 型 wedge 组实现,不仅有效控制了子图组合的数量与规模,而且还聚合了 wedge 组拐点与其二阶邻接点。在子图组合中,WP 算法通过求拐点与其二阶邻接点的邻接点交集,并结合 wedge 组拐点的有序性,确保每个 bi-triangle 仅以 wedge 三元组形式在一个子图组合中被枚举。

ZP 算法将 bi-triangle 视为由两个 zedge 结构组成,枚举时首先以 wedge 组为基本单位执行第一次分图,随后通过“压缩 zedge 组”的构造与还原机制,将起点和终点哈希值相同的 zedge 聚合到同一子图组合中,完成第二次分图,最后从分布式存储系统中读取每个 zedge 组子图组合,拼接其中起点和终点相同的两个 zedge,枚举出 bi-triangle。

在真实二分图数据集上的实验结果表明,相较于 GP 算法,在子图组合数据量上,WP 算法平均减少了 99.5%,ZP 算法平均减少了 96.7%;在枚举时间上,WP 算法至少节省了 97.8%,ZP 算法至少节省了 99.0%。

1 相关工作

本章对与本文研究具有相关性的工作进行梳理,主要包括 bi-triangle、butterfly 以及二分图稠密子图。

1.1 bi-triangle

bi-triangle 作为本文的研究对象,最早由文献[15]提出。该研究从理论上证明了使用 bi-triangle 计算二分图 CC 的合理性与高效性。随后,文献[16]探讨了 bi-triangle 的精确计数问题,建立了以顶点邻接点集大小作为参数的 bi-triangle 数量计算公式。在改进算法中,文献[16]在二分图的两个顶点集间优先处理度数更大的顶点以减少计算量。文献[26]对变种 bi-triangle 展开研究,提出了相对应的准确数量计算定理及公式,并通过共享邻接点集的交集结果优化计算效率。文献[17]则提出一种基于概率的 bi-triangle 数量估算方法,该方法首先通过顶点度数与图边数量的比值确定采样概率,然后结合文献[16]的公式并引入概率修正参数估算 bi-triangle 的数量。需要指出的是,文献[16-17]是基于特定拓扑公式在单机环境下实现计算的,无法做到分布式枚举 bi-triangle。此外,文献[18]在研究其他图问题时提出单机环境下的 bi-triangle 枚举方案:对同一顶点集内的 3 个点,计算邻接点交集,利用三重嵌套循环遍历交集元素,生成候选三元组,最终与对应顶点形成 bi-triangle 结构。

1.2 butterfly

butterfly(4-环)作为一种与 bi-triangle 结构相

似的闭环拓扑结构,在二分图分析领域已形成较为完整的研究体系。其研究发展大致可分为 3 个阶段:基础算法优化,大规模场景扩展以及属性融合分析。

在基础算法研究阶段,文献[27]首次提出通过计算每个顶点 u 与其二阶邻接点 α (距离 u 为 2 跳的顶点)的共同邻居数实现 butterfly 结构计数。随后,文献[28]改进该方法,采用哈希表结构仅处理序号大于 u 的二阶邻接点,从而减少重复计算。文献[29]引入顶点度优先排序策略,通过动态调整顶点处理顺序并结合哈希存储,进一步降低计算复杂度。

随着大规模二分图场景的出现,多种解决方案开始涌现。文献[30]提出了基于 wedge 结构聚类的 butterfly 并行计数算法。文献[31]构建了面向流式数据的实时计数框架。文献[25]首次对 butterfly 的分布式计数算法进行了研究,提出两阶段消息传递机制:顶点首先将信息广播至一阶邻接点,然后由一阶邻接点转发至二阶邻接点,通过消息聚合完成 butterfly 计数。此后,文献[32]提出基于分布式哈希表的设计,其实现机制包含两个核心阶段:首先,各计算节点维护本地哈希表,其键为顶点 u 与其二阶邻接点 α 组成的顶点对,值为 u 和 α 的共同邻居基数;随后通过全局 shuffle 操作跨节点交换哈希表数据,在总表中对每个顶点对的共同邻居数执行组合计算,从而统计出完整的 butterfly 结构数量。

近年来,研究者将 butterfly 计算拓展到更复杂的图结构场景中。例如,文献[33]研究了不确定二分图中的 butterfly 计数问题,文献[34]则探索了时序二分图场景中的 butterfly 分析方法。

1.3 二分图稠密子图

稠密子图挖掘是二分图分析中的一个重要研究方向,其核心在于通过不同的拓扑特征来定义子图稠密性。例如: (r, s) -core 模型基于顶点度数约束,被定义为二分图中满足一类顶点度不小于 r 、另一类顶点度不小于 s 的极大子图;在基于 butterfly 特征的 k -bitruss 模型中,则要求每个顶点参与的 butterfly 结构数量不少于 k 。针对模型求解,研究者提出了多种优化策略:文献[35-36]采用层次化索引技术,分别实现了 (r, s) -core 和 k -bitruss 的高效分解;文献[23]提出了 butterfly 完全子图的概念,为 k -bitruss 的分解提供了一种高效的计算框架。

随着应用场景的复杂化,传统模型逐渐暴露出两个主要局限:其一,对数据噪声敏感;其二,仅关注直接邻居关系,难以捕捉多跳关联的细粒度结构特

征。为此,学界提出了多维度的模型泛化方案:文献[37]设计 k -缺陷二团模型,通过允许 k 条边的缺失增强鲁棒性,并创新性地融合子图划分、上界剪枝、动态更新和分支优化 4 项关键技术提升计算效率;文献[38]则突破结构分析框架,将节点属性信息转化为余弦距离构建增强型二部图,进而采用谱聚类算法实现语义增强的社区发现。

2 背景知识

本章分为两部分,其中第 2.1 节对本文涉及的图结构给出定义和图示(如图 1~3 所示),第 2.2 节则介绍 GP 算法分布式枚举 bi-triangle 的整体流程。

2.1 结构图示

二分图的结构特性是其顶点集可分为两个互不相交的集合,因此逻辑表述为 $G=(A \cup V, E)$, 其中 A 和 V 共同组成 G 的顶点集,满足 $A \cap V = \emptyset$, E 是 G 的边集,满足 $(a, v) \in E | a \in A \wedge v \in V$ 。对边 (a, v) 而言, a 和 v 是其两个端点,互称为邻接点。顶点 a 全部邻接点构成的集合简称为邻接点集,记作 $N(a)$,其大小 $|N(a)|$ 表示 a 的度,记作 $d(a)$ 。离顶点 a 的距离为 2 跳的顶点称为 a 的二阶邻接点,其构成的集合记作 $N_2(a)$,集合大小 $|N_2(a)|$ 表示 a 的二阶度数,记作 $d_2(a)$ 。此外,二分图 G 中所有顶点的平均度数记作 d_{ag} 。二分图的结构示例如图 1(a)所示。

定义 1(bi-triangle^[15-16]) 在 G 中,由顶点 $a_1, a_2, a_3 \in A$ 和顶点 $v_1, v_2, v_3 \in V$ 组成的环形路径称为 bi-triangle。bi-triangle 的结构示例如图 1(b)所示,其对应路径为 $a_1-v_1-a_2-v_2-v_3-a_3-v_3-a_1$ 。

定义 2(bi-triangle 模式异构体) 在 G 中,由顶点 $a_1, a_2, a_3 \in A$ 和顶点 $v_1, v_2, v_3 \in V$ 组成的 bi-triangle 共有 6 种不同的连接模式,这些模式互称为模式异构体,如图 2 所示。

定义 3(wedge^[16,29-30]) 在 G 中,由 u, x_1, x_2 3 个顶点连接而成共包含 $(u, x_1), (u, x_2) \in E$ 2 条边的结构,称为 wedge,如图 3(a)所示,记作 $\langle x_1, u, x_2 \rangle$ 。其中, u 是 x_1 与 x_2 共同的一阶邻接点,称为拐点。

定义 4(wedge 组) 在 G 中,由所有拐点相同的 wedge 聚合形成的图结构,称为 wedge 组,如图 3(b)所示,记作 $\omega=(u, N(u))$,其中 u 是 ω 的拐点。在类别上, wedge 组分为 A 型 wedge 组 ω_A 和 V 型 wedge 组 ω_V ,前者满足 $u \in A$,后者满足 $u \in V$ 。

定义 5(wedge 组子图) 在 G 中,仅由 wedge 组构成的子图称为 wedge 组子图,记作 W 。

定义 6(zedge) 在 G 中,满足 $q \in A, t \in V$ 且序号关系是 $q.id < u_1.id$ 的三步路径 $q-u_1-u_2-t$ 称为 zedge,如图 3(c)所示,记作 $\langle q, u_1, u_2, t \rangle$ 。其中, q 和 t 分别是 zedge 的起点和终点, u_1 和 u_2 分别是 zedge 的第一拐点和第二拐点,而 (u_1, u_2) 则是 zedge 的拐点边。

定义 7(zedge 组) 在 G 中,所有拐点边相同的 zedge 聚合形成的图结构称为 zedge 组,如图 3(d)所示,记作 (Q, u_1, u_2, T) 。其中, Q 和 T 分别是 zedge 组的起点集和终点集。起点集满足 $Q = N(u_1) \setminus u_2$ 且其中顶点的序号均小于 u_2 ,而终点集满足 $T = N(u_2) \setminus u_1$ 。

定义 8(zedge 组子图) 在 G 中,仅由 zedge 组构成的子图称为 zedge 组子图,记作 Z 。

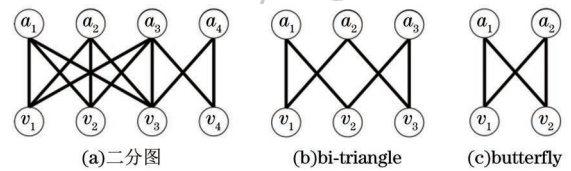


图 1 二分图、bi-triangle 以及 butterfly 结构示例

Fig.1 Bipartite graph, bi-triangle, and butterfly structure examples

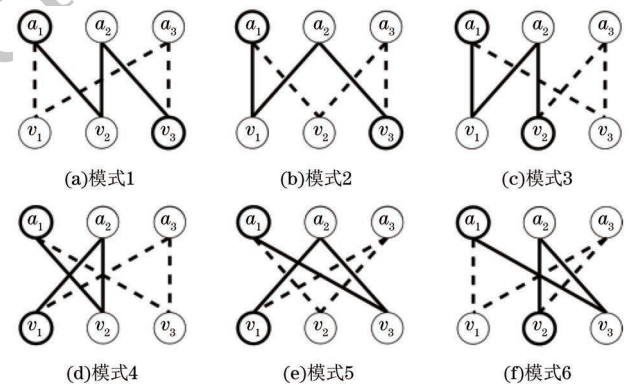


图 2 同一组 6 个顶点的 6 种不同 bi-triangle 连接模式

Fig.2 Six different bi-triangle connection patterns of the same set of six vertices

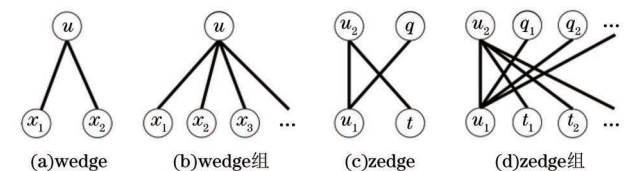


图 3 wedge、wedge 组、zedge、zedge 组结构示例

Fig.3 wedge, wedge group, zedge, and zedge group structure examples

2.2 GP 算法

GP 算法的伪代码如算法 1 所示,分为分图和枚举两个阶段,其中分图阶段进一步包括子图生成(第 1 行)和子图组合构造(第 2 行)。

算法 1 GP 算法输入 $G=(A \cup V, E)$ 输出 G 中全部的 bi-triangle1. 将原始大图随机划分为 p 份子图 //分图阶段的子图生成步骤

2. 计算节点相互发送消息构造子图组合 //分图阶段的子图组合构造步骤

3. 将子图组合中来自顶点集 A 的点 3 个为一组输入到算法 2 //枚举阶段

然而,对于 bi-triangle 的分布式枚举任务,GP 算法作为解决方案存在以下问题:

在分图阶段(第 1 行和第 2 行),GP 算法对输入的二分图边集不做任何预处理,而是直接将其随机划分为 p 份子图。这 p 份子图分布在分布式集群的各个节点内存中。随后,各节点通过消息传递交换子图,构造 $C(p, 6)$ 个 6-子图组合。该设计基于边分布的覆盖性原理:任意一个 bi-triangle 由 6 条边组成,这些边可能分布在 k 个不同的子图中($1 \leq k \leq 6$),而每个可能的 k -子图集合必然被至少一个 6-子图组合所包含。基于此,在随后的枚举阶段,GP 算法可通过分布式处理所有 6-子图组合,枚举二分图中所有的 bi-triangle。

在枚举阶段(第 3 行),每个 6-子图组合作为一个枚举任务,将其中来自顶点集 A 的顶点 3 个为一组输入算法 2 进行处理,算法 2 的输出则是所有包含这 3 个点的 bi-triangle。具体而言:算法 2 首先计算输入的 3 个顶点的邻接点集的交集;随后通过嵌套的三重循环,从交集中生成三元组。当三元组中的元素互异时,以 wedge 三元组的形式枚举出 bi-triangle。图 2 中的所有模式异构体均可视为由 3 个 wedge 结构组成。例如,图 2 中的模式 2 可分解为 $\langle a_1, v_1, a_2 \rangle$ 、 $\langle a_1, v_2, a_3 \rangle$ 与 $\langle a_2, v_3, a_3 \rangle$ 3 个 wedge,而其他连接模式亦可按相同方式分解。通过枚举 wedge 三元组,算法 2 在枚举 bi-triangle 的同时,可有效区分不同的模式异构体。

算法 2 基于邻接点集的 bi-triangle 枚举算法^[18]输入 a_1, a_2, a_3 输出 包含 a_1, a_2, a_3 的全部 bi-triangle

```

1. for  $v_1$  in  $N(a_1) \cap N(a_2)$ 
2.   for  $v_2$  in  $N(a_1) \cap N(a_3)$ 
3.     for  $v_3$  in  $N(a_2) \cap N(a_3)$ 
4.       if  $v_1 \neq v_2 \neq v_3$  then 枚举出:
         ( $\langle a_1, v_1, a_2 \rangle$ ,  $\langle a_1, v_2, a_3 \rangle$ ,  $\langle a_2, v_3, a_3 \rangle$ )
5. endfor to line 3 2 1

```

3 WP 算法

本章分为三部分,其中第 3.1 节介绍 WP 算法

分布式枚举 bi-triangle 的具体细节,第 3.2 节对 WP 算法的性能进行分析,第 3.3 节则对比 GP 算法与 WP 算法。

3.1 WP 算法

为解决 GP 算法在分布式枚举 bi-triangle 时存在的高构造开销和冗余枚举等问题,本节提出 WP 算法,该算法通过 3 项策略对 GP 算法进行了优化。

策略 1 以 wedge 组为基本分图单位。

与 GP 算法通过随机划分二分图边集生成子图的方式不同,WP 算法基于 bi-triangle 可以分解为 3 个 wedge 的特性,将 wedge 聚合为 wedge 组,并以 wedge 组作为基本分图单位,从而构建 wedge 组子图,实现拓扑结构信息的有效聚合。整个过程的伪代码如算法 3 所示。

算法 3 的输入包括二分图边集 E 、子图数量 p 和类型 TYPE。首先,算法遍历输入的边集 E ,根据 TYPE 决定以 A 型或 V 型 wedge 组生成子图,即将每条边 (a, v) 转换为键值二元组 $\langle a, v \rangle$ 或 $\langle v, a \rangle$ 并存入集合 M 。随后,算法对 M 按键聚类,生成 wedge 组 $(u, N(u))$,其中 u 表示 wedge 组拐点, $N(u)$ 为其邻接点集。在此基础上,通过哈希函数 $\text{hash}(u, p)$ 计算 $(u, N(u))$ 的分组编号,将其转换为三元组 $(\text{hash}(u, p), u, N(u))$ 并存入集合 W 。最后,算法根据哈希键对 W 进行聚类,此时原始二分图 G 转换为由 p 份子图组成的 Ω_{TYPE} 。算法 3 的底层逻辑是根据边的端点聚类,计算量为 $|E|$ 。

算法 3 wedge 组子图生成算法输入 E, p, TYPE //TYPE $\in \{ 'A', 'V' \}$ (枚举值 //集合)

输出 根据 TYPE 构造出的 wedge 组子图

```

1.  $M \leftarrow \emptyset$ 
2. foreach  $(a, v)$  in  $E$  in distributed do
3.    $M.add(\text{TYPE} = 'A' \langle a, v \rangle : \langle v, a \rangle)$ 
4. endfor to line 2
5. 根据键对  $M$  中的键值二元组作聚类,同时合并键相同的键值二元组
6.  $W \leftarrow \emptyset$ 
7. foreach  $(u, N(u))$  in  $M$  in distributed do
8.    $W.add((\text{hash}(u, p), u, N(u)))$ 
9. endfor to line 7
10. 根据键对  $W$  中的键值二元组作聚类,得到  $\Omega_{\text{TYPE}} = W_{\text{TYPE}}^0 \cup W_{\text{TYPE}}^1 \cup \dots \cup W_{\text{TYPE}}^{p-1}$ 

```

策略 2 以分布式存储系统为子图交换媒介。

GP 算法通过消息传递机制交换子图,该过程涉及合并打包、缓存排队、序列化和网络传输等多个

步骤。在消息数据量较小时,该机制可实现节点间的低延迟通信。然而,GP 算法在枚举 bi-triangle 时,需要构造的子图组合总数据量是原始图的 $C(p-1, 5)$ 倍,对应的消息传输量也处于同一数量级。这种大规模的消息传递极易导致消息队列堵塞与缓存溢出,进而造成消息发送和接收失败。此外,消息的序列化和反序列化过程进一步增加 CPU 开销,拖慢计算速度。

相比之下,WP 算法采用将子图存入分布式存储系统的策略。各节点在构造子图组合时,直接从分布式存储系统中读取所需子图至内存进行处理,实现了计算节点之间的数据解耦。与依赖消息传递的 GP 算法相比,WP 算法在构造子图组合时仅保留网络传输过程,不仅规避了消息队列拥堵和缓冲区溢出问题,还能支持更大规模的子图组合构造。在大规模子图组合构造场景下,该策略比消息发送策略更具优势。

策略 3 基于 A 型和 V 型 wedge 组拼接的二阶邻接点聚合。

由图 2 展示的 6 种模式异构体可见,二分图中同一顶点集内的 3 个顶点构成 bi-triangle 的必要条件是它们互为二阶邻接点。然而,GP 算法采用随机划分策略生成子图,导致子图间拓扑结构缺乏关联性,使得枚举阶段输入算法 2 的大量顶点三元组不满足该条件,产生无效枚举。

为减少无效枚举,WP 算法提出一种基于 A 型和 V 型 wedge 组拼接的优化策略。具体而言:首先通过算法 3 构造出 Ω_A 与 Ω_V ,并按照策略 2 将其存入分布式存储系统;接着, Ω_A 驻留内存,对于 $\omega = (u, N(u)) \in \Omega_A$,算法从分布式存储系统中读取以 $N(u)$ 中顶点为拐点的 V 型 wedge 组进行拼接,从而完整地获取 u 的二阶邻接点集 $N_2(u)$;最后,将三元组 $(u, \alpha, \beta) (\alpha, \beta \in N_2(u))$ 输入到算法 2 进行枚举,由于 α 和 β 均来自 $N_2(u)$,因此该三元组满足互为二阶邻接点的概率显著提高,从而大幅度减少无效枚举,同时通过约束 $u.id < \alpha.id < \beta.id$,借助顶点的有序性,三者构成的 bi-triangle 将仅被枚举一次。

采用上述策略的 WP 算法伪代码如算法 4 所示:

第 1 行调用算法 3,生成 Ω_A 与 Ω_V (策略 1),并将其存储在分布式存储系统中(策略 2)。随后, Ω_A 驻留在集群节点的内存中,对于其中任意 A 型 wedge 组 $\omega_A = (a, N(a))$,读取以 $fnr \in N(a)$ 为拐点的 V 型 wedge 组来聚合 $N_2(a)$ (策略 3)。完成

聚合后,再次读取以 $snr \in N_2(a)$ 为拐点的 A 型 wedge 组,为后续枚举所需取交集操作提供数据支持。具体而言:

第 6~11 行对每个 fnr 执行操作:先将 $\text{hash}(fnr, p)$ 添加到 index_{fnr}^i 中,该哈希值用于确定 fnr 为拐点的 V 型 wedge 组所在子图在分布式存储系统中的位置;再将 fnr 添加到 filter_{fnr}^i 。

第 12~23 行根据 index_{fnr}^i 中的索引,从分布式存储系统中读取对应的 V 型 wedge 组子图。为避免读取拐点不属于 W_A^i 的 V 型 wedge 组,使用 filter_{fnr}^i 进行过滤:若拐点存在于 filter_{fnr}^i ,则读取整个 wedge 组并存入 map_V ,同时计算拐点的一阶邻接点 snr 的哈希值 $\text{hash}(snr, p)$,将其添加到索引集合 index_{snr}^i 。该哈希值用于定位以 snr 为拐点的 A 型 wedge 组,并将 snr 添加到 filter_{snr}^i ;否则,跳过对拐点一阶邻接点的读取。

第 24~31 行根据 index_{snr}^i 中的索引,从分布式存储系统中读取索引对应的 A 型 wedge 组子图。若拐点存在于 filter_{snr}^i ,则读取整个 wedge 组并存入 map_A ,否则跳过对拐点一阶邻接点集的读取。

第 32~43 行在完成子图组合的构造后,对于 W_A^i 中每一个 A 型 wedge 组 $\omega_A = (a, N(a))$ 执行相同操作:首先将 $N_2(a)$ 中所有序号大于拐点 a 的二阶邻接点存入 H_u ;然后将 H_u 中的顶点两两配对成二元组,与拐点 a 一同输入算法 2,以枚举三者组成的 bi-triangle。其中,二元组中各点的邻接点集通过查询 map_A 获取。

算法 4 WP 算法

输入 $G = (A \cup V, E), p$

输出 G 中全部的 bi-triangle

1. 使用算法 3 构造出 Ω_A 与 Ω_V ,并作分布式存储
2. foreach W_A^i in Ω_A in distributed do $\parallel \Omega_A$ 驻留内存
3. $\text{map}_A, \text{map}_V \leftarrow \emptyset$ //分别存储 W_A^i 从分布式存储系统中读取的 A 型与 V 型 wedge 组结构
4. $\text{index}_{fnr}^i, \text{index}_{snr}^i \leftarrow \emptyset$ //分别存储 W_A^i 需要读取的 $\parallel$ V 型与 A 型 wedge 组子图在外存中的索引
5. $\text{filter}_{fnr}^i, \text{filter}_{snr}^i \leftarrow \emptyset$
6. foreach $\omega_A = (a, N(a))$ in W_A^i //对驻留内存的 W_A^i 中的 A 型 wedge 组子图进行遍历
7. foreach fnr in $N(a)$ do
8. $\text{index}_{fnr}^i.add(\text{hash}(fnr, p))$ //计算 fnr 为拐点 $\parallel$ 的 V 型 wedge 组索引,元素加入时去重
9. $\text{filter}_{fnr}^i.add(fnr)$ //从外存读取 V 型 wedge $\parallel$ 组子图时,用于判断是否读取,并在元素加入时进行去重
10. endfor to line 7

```

11.   endfor to line 6 //需要读取的 V 型 wedge 组子图
//索引收集完成
12.   foreach j in indexinri do
13.       在分布式存储介质中定位 WVi
14.       foreach ωV=(v, N(v)) in WVi do //从分布式
//存储系统中读取 V 型 wedge 组子图
15.           if filterinri.contains(v) then
16.               foreach snr in N(v) do
17.                   indexsnri.add(hash(snr, p))
18.                   filtersnri.add(snr)
19.               endfor to line 16
20.               mapV.put(v, N(v))
21.           else 跳过对 N(v) 的读取
22.       endfor to line 14 //需要读取的 A 型 wedge 组
//子图索引收集完成
23.   endfor to line 12 //V 型 wedge 组读取完成
24.   foreach k in indexsnri do
25.       在分布式存储介质中定位 WAk
26.       foreach ωAk=(u, N(u)) in WAk do //从分布式
//存储系统中读取 A 型 wedge 组子图
27.           if filtersnri.contains(u) then
28.               mapA.put(u, N(u))
29.           else 跳过对 N(u) 的读取
30.       endfor to line 26
31.   endfor to line 24 //A 型 wedge 组读取完成
32.   foreach ωA=(a, N(a)) in WAi do //再次对计算
//节点内存中的 A 型 wedge 组进行遍历
33.       Ha←∅//存放 a 的二阶邻接点
34.       foreach hop1 in N(a)
35.           N2(a)=mapV.get(hop1)
36.           foreach hop2 in N2(a)
37.               if hop2.id>a.id then
38.                   Ha.add(hop2)
39.           else 跳过对 hop2 的处理
40.       endfor to line 36
41.   endfor to line 34 //a 的二阶邻居收集完成
42.   将 Ha 中的点两两组合,与 a 一起输入算法 2,
以枚举三者构成的 bi-triangle //在算法 2 中, a 的二阶邻接
//点的邻接点集通过查询 mapA 获取
43.   endfor to line 32 //a 参与构成的 bi-triangle 枚举
//完成
44. endfor to line 2

```

3.2 WP 算法性能分析

本节从子图组合数量、子图组合数据量和枚举计算量 3 个维度对 WP 算法进行性能分析。

1) 子图组合数量。

在 WP 算法中,驻留内存的 Ω_A 从分布式存储系统中读取子图构成子图组合,因此 WP 算法构造

的子图组合数量是 $|\Omega_A|=p$ 。

2) 子图组合数据量。

将 WP 算法产生的子图组合数据量记作 K_{WP} ,满足 $K_{WP}=K_V+K_A$,其中 K_V 和 K_A 分别是 WP 算法两次读取 wedge 组子图的网络数据量。 K_V 和 K_A 具体数值分析如下:

WP 算法以 wedge 组子图为存储颗粒度,在读取分布式存储系统中的目标 wedge 组时,必须首先加载其所在 wedge 组子图。为提高读取效率,WP 算法通过构建索引集实现批量读取。如算法 4 第 8 行和第 17 行所示, W_A^i 将自身所有 wedge 组拐点的一阶邻接点和二阶邻接点与子图数量参数 p 作哈希映射,形成 $index_{fnr}^i$ 和 $index_{snr}^i$ 。由于两类邻接点的数量均远远大于 p , $index_{fnr}^i$ 和 $index_{snr}^i$ 中的映射结果涵盖区间 $[0, p-1]$ 内的所有整数值。因此, W_A^i 需要读取的 V 型和 A 型 wedge 组子图网络结构分别对应 $\sum W_V^i=\Omega_V$ 与 $\sum W_A^i=\Omega_A$ (其中整数 i 的值从 0 取到 $p-1$)。

进而, WP 算法需累计读取 p 次 Ω_V 和 Ω_A 。在此过程中,拐点包含在过滤器中的 wedge 组会被完整读取,而其余 wedge 组则仅读取其拐点信息。该过程等效于读取 p 次顶点集合,而在这 p 次读取中,对于集合 V 和 A 中的顶点,需访问其邻接点集 F 次。因此, $K_V=\sum_{v \in V} (F(v) \cdot d(v) + p - F(v))$, 而 $K_A=\sum_{a \in A} (F(a) \cdot d(a) + p - F(a))$, 其中 $F(u)$ 的计算方法是为顶点 u 构造位长为 p 的二进制编码,其中第 i 位为 1 表示顶点 u 出现在 $filter_{fnr}^i$ ($u \in V$) 或 $filter_{snr}^i$ ($u \in A$) 中,若未出现则该位为 0。统计编码中值为 1 的位数,即为 $F(u)$ 的值。

在 K_V 和 K_A 的数值表达式中, $F(v)$ 和 $F(a)$ 会随着顶点度数的不同而发生变化,无法得出 K_{WP} 的精确数值。可使用二分图顶点的平均度数 d_{ag} 来近似表示每个顶点的邻接点集大小,从而简化分析,为 K_{WP} 提供一个合理的上限估计:

若 $p \leq d_{ag}$ 、 $F(v)$ 、 $F(a) \leq p$, 使得 $K_V \leq p \cdot \sum_{v \in V} (d(v))$ 、 $K_A \leq p \cdot \sum_{a \in A} (d(a))$, 则 $K_{WP} \leq 2 \cdot p \cdot |E|$ 。

若 $p > d_{ag}$ 、 $F(v)$ 、 $F(a) \leq d_{ag}$, 使得 $K_V \leq d_{ag} \cdot \sum_{v \in V} (d(v))$ 、 $K_A \leq d_{ag} \cdot \sum_{a \in A} (d(a))$, 则 $K_{WP} \leq 2 \cdot d_{ag} \cdot |E|$ 。

3) 枚举计算量。

WP 算法包括分图和枚举两个阶段。分图阶段

进一步划分为子图生成和子图组合构造两部分。

在子图生成部分,算法 4 中第 1 行调用算法 3,将 G 转换为 Ω_A 与 Ω_V ,计算量为 $2 \cdot |E|$ 。

在子图组合构造部分,算法 4 中第 6~11 行,对于任意 $\omega_A = (a, N(a)) \in W_A^i$,遍历 $fnr \in N(a)$,将该操作从 W_A^i 推广至 Ω_A 的所有子图,对应计算量为 $\sum_{a \in A} (d(a))$ 。算法 4 中第 12~23 行,读取以 $fnr \in N(a)$ 为拐点的 V 型 wedge 组,考虑 Ω_A 的所有子图,对应计算量是 $\sum_{v \in V} (F(v) \cdot d(v) + p - F(v))$ 。算法 4 中第 24~31 行,读取以 $snr \in N_2(a)$ 为拐点的 A 型 wedge 组,考虑 Ω_A 的所有子图,计算量是 $\sum_{a \in A} (F(a) \cdot d(a) + p - F(a))$ 。

在枚举阶段,算法 4 中第 32~41 行收集拐点 a 的二阶邻接点,考虑 Ω_A 的所有子图,计算量是 $\sum_{v \in V} (d^2(v))$ 。算法 4 中第 42 行枚举 W_A^i 中的 bi-triangle,将该操作从 W_A^i 推广至 Ω_A 的所有子图,计算量是 $\sum_{a \in A} (d_2^2(a) \cdot d_{ag}^3)$ 。

3.3 与 GP 算法的对比

在分布式枚举 bi-triangle 时,WP 算法相较于 GP 算法实现了显著的性能优化。首先,WP 算法将子图组合数量从 GP 算法的指数级 $O(p^6)$ 降至线性级 $O(p)$ 。其次,GP 算法构造的子图组合数据量为 $C(p-1, 5) \cdot |E|$,而 WP 算法将对应指标上限降低至 $2 \cdot \min(p, d_{ag}) \cdot |E|$ 。最后,通过引入有序邻接点筛选机制,WP 算法解决了 GP 算法中存在的重复枚举问题。具体而言,对于任意 wedge 组的拐点 $u \in W_A^i$, u 凭借 W_A^i 读取的 V 型 wedge 组子图构建出其二阶邻接点集 $N_2(u)$ 的子集 H_u (其中每个元素的序号均大于 $u.id$)。随后, H_u 中的元素两两组合,生成有序对 (α, β) (满足 $\alpha.id < \beta.id$),并与 u 一起输入算法 2 枚举 bi-triangle。由于 u 只在 W_A^i 中被处理一次,加之 $u.id < \alpha.id < \beta.id$,三者构成的任意 bi-triangle 将仅被枚举一次。

尽管 WP 算法消除了结果重复,但其基于 wedge 组的枚举计算方式,却存在 wedge 结构被重复构造的问题。对于 wedge 组拐点 $u \in W_A^i \wedge \alpha \in W_A^j \wedge \beta \in W_A^k$,若 $u.id < \alpha.id < \beta.id$ 且三者互为二阶邻接点,则交集操作 $N(\alpha) \cap N(\beta)$ 将在 u 构建 H_u , α 构建 H_α 枚举 bi-triangle 时被重复执行,进而导致 wedge 结构 $\langle \alpha, \gamma, \beta \rangle$ 被冗余构造,其中 $\gamma \in N(\alpha) \cap N(\beta)$ 。由于 u 与 α 位于不同的节点内存中,交集操作的结果无法跨计算节点共享,因此该冗

余计算无法避免,降低了枚举效率。

4 ZP 算法

本章提出 ZP 算法,其计算复杂度较 WP 算法显著降低,包括三部分:第 4.1 节详述 ZP 算法的分布式 bi-triangle 枚举实现,第 4.2 节进行算法性能分析,第 4.3 节则对比 WP 算法与 ZP 算法。

4.1 ZP 算法

ZP 算法以策略 4 为主要设计构想:

策略 4 通过“压缩 zedge 组”的构造和还原机制,实现分图。

如图 2 所示,bi-triangle 的 6 种模式异构体均可分解为两个 zedge (图中实线和虚线表示的三步路径),两者起点和终点 (图中加粗标记的点,其中 a_1 为起点) 相同,而拐点边不同。基于这一特性,本节提出 ZP 算法。ZP 算法枚举 bi-triangle 的核心思想是将起点和终点相同的 zedge 逐一加入同一容器。每次新加入一个 zedge 时,比较它和容器中已有 zedge 的拐点边是否有相同的端点,若没有,则这两个 zedge 构成一个 bi-triangle。在分布式环境中,这种枚举方法成功的关键在于将起点和终点相同的 zedge 组划分到相同子图中。为实现该目标,ZP 算法采用构造压缩 zedge 组的方法。所谓“压缩”,是指用哈希映射值代替 zedge 组的起点集和终点集。具体而言:首先,对于任意 $(a, v) \in E$,将 $N(v)$ 与 $N(a)$ 中的元素与参数 p 进行哈希映射,分别映射为集合 set_Q^p 与 set_T^p ;接着,将 set_Q^p 与 set_T^p 中的元素两两组合,任意二元组 (q^p, t^p) 与 (a, v) 构成压缩 zedge 组;最后,根据 q^p, t^p 分别从 $N(v)$ 与 $N(a)$ 中回溯映射值与之相等的原始数据,重构完整的 zedge 组,从而聚合具有相同起点和终点的 zedge 组。

ZP 算法的伪代码包括分图阶段的算法 5 和枚举阶段的算法 6。算法 5 首先通过构造 wedge 组实现第一次分图,随后进行压缩 zedge 组的构造与还原,并以还原后的 zedge 组为基本单位执行第二次分图。分图完成后,算法 6 负责 bi-triangle 的分布式枚举。

下面首先对算法 5 的伪代码进行说明,其中涉及的哈希值均以 p (子图数量) 作为哈希参数:

第 1~12 行:数据准备阶段。该阶段分为两步完成:第一步复用 WP 算法中的子图生成和 V 型 wedge 组读取操作,将所读取的 V 型 wedge 组存入 map_V (第 4 行);第二步是对 wedge 组拐点的邻接点集进行分类,分类依据是邻接点集元素与 p 的哈希值。具体而言:对于 $\omega_A = (a, N(a)) \in W_A^i$,将

$fnr \in N(a)$, 以 $\text{hash}(fnr, p)$ 为键追加存入 map_{fnr}^a , 再以拐点 a 为键将 map_{fnr}^a 存入 map_A^h (第 5~8 行)。对于 $\omega_v = (v, N(v)) \in \text{map}_V$ 的处理类似: 将 $N(v)$ 中元素与 p 求哈希值后追加存入 map_{snr}^v , 接着以拐点 v 为键将 map_{snr}^v 存入 map_V^h (第 9~12 行)。

第 13~23 行: 构造压缩 zedge 组。对于 $\omega_A = (a, N(a)) \in W_A^i$, 首先, 计算 $N(a)$ 中所有点与 p 的哈希值并将该值存入 set_T^h (第 16 行); 然后, 对于 $N(fnr)(fnr \in N(a))$, 筛选出其中序号小于 a 的顶点, 将其与 p 的哈希值存入 set_Q^h (第 19 行); 最后, 将 set_Q^h 与 set_T^h 中的元素进行两两组合, 以生成的二元组为键, 边 (fnr, a) 为值存入 map_Z^h : 若该键已存在, 在其对应的值集合中追加新值; 否则, 新建列表并将新值加入其中 (第 20 行和第 21 行)。

第 24~45 行: 还原 zedge 组。对于 map_Z^h 中键 (q^p, t^p) 关联的每个压缩 zedge 组 $z^p = (q^p, u_1, u_2, t^p)$, 首先, 通过 $\text{map}_V^h.get(u_1)$ 获取 u_1 的邻接点集 map_{snr}^v , 并从中提取所有与 p 的哈希值为 q^p 且序号小于 u_2 的点, 构成起点集合 set_Q (第 29~34 行); 接着, 通过 $\text{map}_V^h.get(u_2)$ 获取 u_2 的邻接点集 map_{fnr}^a , 并从中提取所有与 p 的哈希值为 t^p 的点, 构成终点集合 set_T (第 35~40 行); 最后, z^p 被还原为 $z = (\text{set}_Q, u_1, u_2, \text{set}_T)$, 当键 (q^p, t^p) 被处理完后, W_A^i 中全部的起点哈希值为 q^p , 终点哈希值为 t^p 的 zedge 组将被划分到 $Z_{(q^p, t^p, W_A^i, \text{id})}$ 。在 map_Z^h 的键被全部处理后, ZP 算法完成第二次分图。

算法 5 ZP 算法分图阶段算法

输入 $G = (A \cup V, E), p$

输出 Z // zedge 组子图

1. 用算法 3 构造 Ω_A 与 Ω_V , 并将 Ω_V 作分布式存储
2. foreach W_A^i in Ω_A in distributed do
3. $\text{map}_V, \text{map}_A^h, \text{map}_V^h \leftarrow \emptyset$
4. 沿用 WP 算法 6 至 19 行: 从分布式存储读取 V 型 wedge 组子图并存入 map_V
5. foreach $\omega_A = (a, N(a))$ in W_A^i // 对驻留内存的 W_A^i // 中的 A 型 wedge 组子图进行遍历
6. $\text{map}_{fnr}^a \leftarrow \emptyset$
7. 将 $fnr \in N(a)$ 以 $\text{hash}(fnr, p)$ 为键追加存入 map_{fnr}^a , 然后执行 $\text{map}_A^h.put(a, \text{map}_{fnr}^a)$
8. endfor to line 5 // 构造 map_A^h 完成
9. foreach $\omega_v = (v, N(v))$ in map_V // 对 map_V 中 V 型 // wedge 组子图进行遍历
10. $\text{map}_{snr}^v \leftarrow \emptyset$
11. 将 $snr \in N(v)$ 以 $\text{hash}(snr, p)$ 为键追加存入

map_{snr}^v , 然后执行 $\text{map}_V^h.put(v, \text{map}_{snr}^v)$

12. endfor to line 9 // 构造 map_V^h 完成
13. $\text{map}_Z^h \leftarrow \emptyset$ // 存储压缩 zedge 组
14. foreach $\omega_A = (a, N(a))$ in W_A^i // 对驻留内存的 // W_A^i 中的 A 型 wedge 组子图再次进行遍历
15. $\text{set}_T^h \leftarrow \emptyset$ // 存储压缩终点
16. 将全部 $fnr \in N(a)$ 求出 $\text{hash}(fnr, p)$, 然后将该值存入 set_T^h
17. foreach fnr in $N(a)$
18. $\text{set}_Q^h \leftarrow \emptyset$ // 存储压缩起点
19. 将 $N(fnr)$ 中序号小于 a 的元素依次与 p 求一个哈希值, 然后将该值存入 set_Q^h
20. 将 set_Q^h 与 set_T^h 中的元素两两组合, 任意一个二元组用 (q^p, t^p) 表示
21. 以 (q^p, t^p) 为键, 边 (fnr, a) 为值追加存入 map_Z^h // (fnr, a) 是拐点边, 而 fnr 是压缩 zedge 组的第一拐点, // a 是第二拐点
22. endfor to line 17
23. endfor to line 14 // 以 W_A^i 中每条边为拐点边构造压缩 zedge 组完成
24. foreach z^p in map_Z^h // 访问单个压缩 zedge
25. $q^p = z^p.key.1, t^p = z^p.key.2$
26. $Z_{(q^p, t^p, W_A^i, \text{id})} \leftarrow \emptyset$ // 存储还原 zedge 组
27. foreach e in $z^p.val$ do // 拐点边
28. $u_1 = e.1, u_2 = e.2$ // 获取压缩 zedge 拐点边 // 中的第一拐点和第二拐点
29. $\text{set}_Q \leftarrow \emptyset$
30. $\text{map}_{snr}^v = \text{map}_V^h.get(u_1)$ // u_1 的邻接点集形 // 成的哈希表, 其中键是各元素与 p 的哈希值
31. foreach q in $\text{map}_{snr}^v.get(q^p)$ // 满足哈希值为 // q^p 的 u_1 的邻接点
32. if $q.id < u_2.id$ then $\text{set}_Q.add(q)$
33. else 跳过处理 // 起点需要小于 u_2
34. endfor to line 31
35. $\text{set}_T \leftarrow \emptyset$
36. $\text{map}_{fnr}^a = \text{map}_A^h.get(u_2)$ // u_2 的邻接点集形 // 成的哈希表, 其中键是各元素与 p 的哈希值
37. foreach t in $\text{map}_{fnr}^a.get(t^p)$ // 满足哈希值为 // t^p 的 u_2 的邻接点
38. if $t \neq u_1$ then $\text{set}_T.add(t)$
39. else 跳过处理 // 终点要和 u_1 不同
40. endfor to line 37
41. $z = (\text{set}_Q, u_1, u_2, \text{set}_T)$ // 起点集为 set_Q , 终 // 点集为 set_T , 第一拐点为 u_1 , 第二拐点为 u_2 的 zedge 组, // set_Q 中元素的序号均小于 u_2 , 且与 p 的哈希值均为 q^p , // set_T 中元素与 p 的哈希值均为 t^p
42. $Z_{(q^p, t^p, W_A^i, \text{id})}.add(z)$ // 存储 W_A^i 中全部的起 // 点集与 p 的哈希值均为 q^p , 终点集与 p 的哈希值均为 t^p 的

//zedge 组

43. endfor to line 27 //以 e 为拐点边的压缩 zedge

//组还原完成

44. 将 $Z_{(q^p, t^p, W_A^i, id)}$ 作分布式存储 //起点哈希值为

// q^p , 终点哈希值为 t^p 的 zedge 组聚合完成

45. endfor to line 24

46. endfor to line 2

两次分图完成后, ZP 算法通过算法 6 实现 bi-triangle 的分布式枚举。值得注意的是, map_Z^p 中键 (q^p, t^p) 的数量上限是 p^2 , 因此 W_A^i 生成的分图 $Z_{(q^p, t^p, W_A^i, id)}$ 数量也至多为 p^2 。又由于存在 $|\Omega_A|$ 份 W_A^i , 因此分布式存储系统中的每一个 zedge 组子图组合 $Z_{(q^p, t^p)}$ 满足:

$$Z_{(q^p, t^p)} = Z_{(q^p, t^p, 0)} \cup Z_{(q^p, t^p, 1)} \cup \dots \cup Z_{(q^p, t^p, |\Omega_A|-1)}$$

算法 6 的核心思想是将枚举任务划分为 p^2 个子任务, 每个子任务处理一个 $Z_{(q^p, t^p)}$ 子图组合。在具体处理过程中, 对于 $Z_{(q^p, t^p)}$ 中的每个 zedge, 以其起点和终点构成的序对为键, 拐点边为值存入哈希表。当新插入的 zedge 与哈希表中已有的 zedge 对应同一键时, 比较两者拐点边, 若两者拐点边的第一拐点与第二拐点均不相同, 则以 zedge 二元组的形式枚举出对应的 bi-triangle。算法 6 的伪代码如下:

第 1~6 行: 任务生成。将枚举任务划分为 p^2 个子任务, 并将每个子任务 $task_{(i,j)}$ 加入任务集合 Task。

第 7~22 行: 分布式执行每个子任务。 $task_{(i,j)}$ 从分布式存储介质中定位对应的 zedge 组子图 $Z_{(i,j)}$ 并开始读取。对读取到的每一个 zedge 组 (set_Q, u_1, u_2, set_T) , 依次遍历 set_Q 和 set_T 中的所有点对组合, 以 (q, t) 为键查询哈希映射 map_Z (第 11~16 行)。当键已存在时, 将当前值与已存储的值进行两两比较, 以 zedge 二元组枚举出对应的 bi-triangle (第 17~20 行)。比较完成后将 (u_1, u_2) 作为值存入 map_Z (第 21 行)。

算法 6 ZP 算法枚举阶段算法

输入 p //算法 5 中的子图数量参数

输出 G 中全部的 bi-triangle

1. Task $\leftarrow \emptyset$

2. for $i=0$ to $p-1$ do //构造子任务

3. for $j=0$ to $p-1$ do

4. 将 $task_{(i,j)}$ 加入 Task

5. endfor to line 3

6. endfor to line 2 //子任务构造完成, 每个子任务负责

//枚举起点哈希值为 i , 终点哈希值为 j 的 zedge 组子图中的
//bi-triangle

7. foreach $task_{(i,j)}$ in Task in distributed do

8. $map_Z \leftarrow \emptyset$ //将读取的 zedge 逐个加入该容器

9. 从分布式存储介质中定位 $Z_{(i,j)}$

10. for $k = 0$ to $|\Omega_A| - 1$ do //k 对应算法 5 中的

// W_A^i, id

11. foreach z in $Z_{(i,j,k)}$ do //开始枚举

12. $z = (set_Q, u_1, u_2, set_T)$ //读取 zedge 组

13. foreach q in set_Q do

14. foreach t in set_T do

15. $key = (q, t), val = (u_1, u_2)$

16. $edges = map_Z.get(key)$

17. 将 val 与全部 $e \in edges$ 作比较, 当 e 与 val 两个端点均不相同时:

18. $z_1 = \langle q, u_1, u_2, t \rangle$

19. $z_2 = \langle q, e, 1, e, 2, t \rangle$

20. 枚举 (z_1, z_2)

21. 根据 key 将 val 加入 map_Z

22. endfor to line 14, 13, 11, 10, 7

4.2 ZP 算法性能分析

本节从子图组合数量、子图组合数据量和枚举计算量 3 个维度对 ZP 算法进行性能分析。

1) 子图组合数量。

ZP 算法的子图组合数量等于算法 5 中 map_Z^p 键数量的最大值, 即为 p^2 。

2) 子图组合数据量。

将 ZP 算法的子图组合数据量记作 K_{ZP} , 满足 $K_{ZP} = K_V + K_Z$, 其中 K_V 为 ZP 算法沿用 WP 算法中读取 V 型 wedge 组操作所产生的数据量, 等于 $\sum_{v \in V} (F(v) \cdot d(v) + p - F(v))$ (参见第 3.2 节), 而 K_Z 为算法 6 从分布式存储系统中读取的 zedge 组子图组合的数据大小, 其具体数值分析如下:

ZP 算法构造 zedge 组子图的过程为: 对于 $(a, v) \in E$, 首先将 $N(v)$ 与 $N(a)$ 分别映射为 set_Q^p 与 set_T^p , 然后将其中的哈希值进行两两组合, 生成二元组 (q^p, t^p) , 并与 (a, v) 形成压缩 zedge 组。由此, 该结构的总数是 $|set_Q^p| \cdot |set_T^p| \cdot |E|$ 。为简化分析, 使用二分图顶点的平均度数 d_{ag} 近似表示每个顶点的邻接点集大小, 则有 $|set_Q^p|, |set_T^p| \leq \min(p, d_{ag})$ 。因此, 讨论两种情况: 当 $p \leq d_{ag}$ 时, 压缩 zedge 组的数量为 $p^2 \cdot |E|$, 而单个压缩 zedge 组从 map_V^h 和 map_A^h 回溯的数据大小总计约为 $2 \cdot d_{ag} / p$, 两者相乘后得到 $K_Z \approx 2 \cdot p \cdot d_{ag} \cdot |E|$; 当 $p > d_{ag}$ 时, 压缩 zedge 组的数量为 $d_{ag}^2 \cdot |E|$, 而单个压缩 zedge 组从 map_V^h 和 map_A^h 回溯数据时, 仅涉及两个顶点, 其数

据量可近似为常数 2 (即两个顶点对应的数据大小)。因此, $K_Z \approx 2 \cdot d_{ag}^2 \cdot |E|$ 。

3) 枚举计算量。

ZP 算法包括分图 (算法 5) 和枚举 (算法 6) 两阶段, 下面依次分析其计算量。

在算法 5 中, 首先, 调用算法 3 将 G 转换为 Ω_A 与 Ω_V (第 1 行), 该过程基于边的端点聚类, 计算量为 $2 \cdot |E|$; 随后, 复用 WP 算法的 V 型 wedge 组读取逻辑 (第 4 行), 由第 3.2 节可知其计算量是 $\sum_{a \in A} (d(a))$ 加上 $\sum_{v \in V} (F(v) \cdot d(v) + p - F(v))$; 接着, 对于 $\omega_A = (a, N(a)) \in W_A^i$, 遍历 $fnr \in N(a)$ 以构造出 map_A^h (第 5~8 行), 对 Ω_A 中所有子图的累计计算量为 $\sum_{a \in A} (d(a))$ 。类似地, 对于 $\omega_V = (v, N(v)) \in map_V$, 遍历 $N(v)$ 构造出 map_V^h (第 9~12 行), 其计算量是 $\sum_{v \in V} (F(v) \cdot d(v) + p - F(v))$ 。

在算法 5 中, 第 13~23 行构造压缩 zedge 组: 对于 $\omega_A = (a, N(a)) \in W_A^i$, 首先, 遍历 $fnr \in N(a)$ 构造 set_A^p (第 16 行), 累计计算量为 $\sum_{a \in A} (d(a))$; 随后, 遍历拐点 a 的二阶邻接点集 $N_2(a)$ 构造 set_Q^p (第 19 行), 对应计算量为 $\sum_{v \in V} (d^2(v))$; 最后, 在第 24~45 行中, 每个压缩 zedge 组被扩展并写入分布式存储系统。对于 Ω_A 产生的压缩 zedge 组, 当 $p \leq d_{ag}$ 时, 其数量为 $p^2 \cdot \sum_{a \in A} (d(a))$, 且 set_Q^p 和 set_T^p 的大小为 d_{ag}/p , 对应的回溯数据操作计算量为 $2 \cdot d_{ag}/p$ (第 29~34 行), 因而该步骤的总计算量是 $2 \cdot p \cdot d_{ag} \cdot \sum_{a \in A} (d(a))$; 当 $p > d_{ag}$ 时, 压缩 zedge 组数量为 $d_{ag}^2 \cdot \sum_{a \in A} (d(a))$, 而 set_Q^p 和 set_T^p 均只包含一个顶点的哈希值, 对应回溯数据计算量近似为常数 2 (第 35~40 行), 总计算量为 $2 \cdot d_{ag}^2 \cdot \sum_{a \in A} (d(a))$ 。

算法 6 对算法 5 生成的每个 zedge 组执行两步操作。第 1 步 (第 13~15 行) 将 zedge 组起点集合与终点集合的元素配对; 第 2 步 (第 16~20 行) 则访问每个二元组对应的所有 zedge。假设二元组的起点是 a , 终点是 v , 第 2 步中对应的 zedge 数量为 $\sum_{y \in N(v) \cap N_2(a)} |N(a) \cap N(y)|$, 该值不超过 $d_2(a) \cdot d_{ag}$, 后续分析采用该上界作为近似估计。当 $p \leq d_{ag}$ 时, 算法 5 生成的 zedge 组的数量为 $p^2 \cdot \sum_{a \in A} (d(a))$, 且起点集合和终点集合的大小均为 d_{ag}/p , 因而总计算量为 $\sum_{a \in A} (d(a) \cdot d_2(a) \cdot d_{ag}^3)$; 当 $p > d_{ag}$ 时,

zedge 组的数量变为 $d_{ag}^2 \cdot \sum_{a \in A} (d(a))$ 且起点集合和终点集合的大小为常数 1, 但总计算量仍然是 $\sum_{a \in A} (d(a) \cdot d_2(a) \cdot d_{ag}^3)$ 。

4.3 与 WP 算法的对比

本节从执行流程、算法性能和适用场景 3 个维度对比分析 WP 算法和 ZP 算法, 旨在明确两者的核心差异及适用边界。

1) 执行流程对比。

WP 与 ZP 算法在执行流程上具有相同的初始步骤: 首先构造出 Ω_A 和 Ω_V , 并将其存入分布式存储系统, 同时将 Ω_A 驻留内存; 随后从分布式存储系统中选择性读取部分 Ω_V 。然而, 两者的后续处理逻辑存在显著差异。WP 算法首先基于 Ω_V 的结构信息, 进一步选择性读取 Ω_A , 相当于内存中的 Ω_A 依次与分布式存储中的 Ω_V 和 Ω_A 进行数据交互; 接着构造顶点三元组, 并将其输入算法 2, 以枚举 bi-triangle 结构。相比之下, ZP 算法采用全然不同的策略: 它融合 Ω_A 和 Ω_V 生成 zedge 组子图 Z 并存入分布式存储系统, 随后在枚举阶段对 Z 中起点、终点相同的 zedge 进行拼接以枚举 bi-triangle。需要强调的是, 两种算法在初始阶段共享相同的预处理逻辑, 但后续处理路径完全不同。因此, ZP 算法并非通过包含或覆盖 WP 算法实现功能演进, 而是采用了一种全新的处理范式。

2) 算法性能对比。

在子图组合数量方面, WP 算法为 p , ZP 算法为 p^2 。在子图组合数据量上, WP 算法对应数值 $K_{WP} = K_V + K_A$, 而 ZP 算法对应数值 $K_{ZP} = K_V + K_Z$ 。由于 $K_A \leq \min(p, d_{ag}) \cdot \sum_{a \in A} (d(a))$, 而 $K_Z \approx 2 \cdot \min(p, d_{ag}) \cdot d_{ag} \cdot \sum_{a \in A} (d(a))$, 因此 K_A 小于 K_Z 。综上所述, 在子图组合数量和规模控制上, WP 算法优于 ZP 算法。

在计算复杂度上, WP 算法的最高项为 $\sum_{a \in A} (d_2^2(a) \cdot d_{ag}^3)$, 而 ZP 算法的最高项为 $\sum_{a \in A} (d(a) \cdot d_2(a) \cdot d_{ag}^3)$, 因此 ZP 算法的枚举效率优于 WP 算法。此外, 在 ZP 算法的枚举阶段中, 当拼接起点和终点相同的 zedge 枚举 bi-triangle 时, 以 a_i 为起点、 v_j 为终点的所有 zedge 仅在算法 6 处理二元组 (a_i, v_j) 时被访问, 而不会在处理其他二元组 (如 (a_i, v_k)) 时被重复访问。相比之下, WP 算法通过 wedge 拐点三元组 (a_i, a_j, a_k) 枚举 bi-triangle, 这需要计算 $N(a_j) \cap N(a_k)$, 当不同三元组共享公共元素时, 会

引发重复操作。例如,在处理 (a_j, a_k, a_l) 时, $N(a_j) \cap N(a_k)$ 会再次进行, wedge 结构 $\langle a_l, v, a_k \rangle$ 被重复访问,其中 $v \in N(a_j) \cap N(a_k)$,该冗余进一步降低了 WP 算法的枚举效率。

3) 适用场景对比。

根据本节的算法性能对比分析, WP 算法在子图组合数量和规模控制方面优于 ZP 算法,使其在通信成本敏感和资源受限的分布式系统中具有不可替代的实用价值。

在数据存储与计算分离的跨地域集群中,相比 ZP 算法,采用 WP 算法分布式枚举 bi-triangle,产生的带宽占用开销和传输费用更低。这是因为 WP 算法从存储节点读取的子图组合数据量为 $K_V + K_A$,而 ZP 算法则需读取 $K_V + K_Z$ (其中 $K_Z > K_A$)。以国际电商的跨境分布式集群为例,存储节点通常部署在散热冷却成本较低的地区(如高纬度寒冷地区),而计算节点则设在电价低廉的区域(如水电资源富集地区),从而有效降低集群的能耗成本。在该架构下,枚举 bi-triangle 分析用户的月度商品偏好时,由于任务对响应时间要求相对宽松,跨区域带宽成本成为影响算法部署决策的首要考量因素,而非计算速度。此时,尽管 WP 算法的枚举速度不及 ZP 算法,但其更低的子图组合数据传输费用使其成为更优选择。

在由低资源硬件构成的物联网边缘计算集群中, WP 算法能够顺利完成 bi-triangle 的分布式枚举,而 ZP 算法则可能因子图组合数据规模过大而难以满足实际需求。以智能制造工厂为例,在其设备与传感器构成的二分图中,通过枚举 bi-triangle 可分析设备对特定传感器的依赖程度,从而为制定周期性维护计划提供参考。为降低任务经济成本,该任务可利用工厂闲置的边缘网关集群完成,而无需搭建硬件成本高昂的专用分布式集群进行计算。然而,由于网关的内存和存储资源有限,难以处理大规模子图组合数据。若采用 ZP 算法,容易因子图组合数据规模过大而导致内存和存储同时溢出,甚至引发网关宕机。在该场景下, WP 算法凭借更小的子图组合数量与规模成为更合理的选择,能够有效避免溢出问题,确保任务的可行性和系统的稳定性。

综上所述,在通信成本和计算资源受限且对计算速度没有严格要求的场景中,尽管 WP 算法的枚举速度不及 ZP 算法,但凭借更低的经济成本和更高的可行性,其适用性更强。

与 WP 算法相比, ZP 算法的核心优势在于枚举

复杂度更低、计算速度更快。因此,对于需要保障 bi-triangle 枚举速度的场景, ZP 算法是更优选择。例如,在电商平台中,每日用户新购商品数据更新后,推荐系统需尽快完成 bi-triangle 枚举分析,以反映当日用户商品偏好的变化,从而为次日广告推荐提供数据支持。此时,计算速度更快的 ZP 算法更为适用。同样,在证券交易所的场景中, ZP 算法结合内存带宽充裕的高性能计算集群(如 GPU 加速环境),能够显著提升 bi-triangle 的枚举效率,从而快速识别异常账户变动,满足交易所对高效计算的需求。

5 实验测试与结果分析

本章测试 WP 算法和 ZP 算法的性能:第 5.1 节阐述实验配置,第 5.2 节分析实验结果。

5.1 实验配置

1) 实验环境。

实验在 Spark 分布式集群上进行,分布式存储系统使用 HDFS。集群由 9 台配置相同软硬件的计算节点组成。每个节点配备 Intel® Xeon® Bronze 3204 CPU@1.90 GHz 处理器,操作系统为 Ubuntu 4.15.0-166-generic 版本。集群使用的 Spark 版本为 3.3.0, Scala 版本为 2.12.12, Hadoop 版本为 3.3.3, Java 版本为 1.8.0。实验任务默认参数 p 为 100,配置 81 个 Spark executor,每个 executor 分配 3 个处理器核和 15 GB 内存。

2) 实验算法。

测试算法除本文介绍的 GP、WP 和 ZP 算法以外,还包括 PTE、DBC 和 Monarch 3 种对比算法。PTE 算法是当前性能领先的简单图三角形分布式枚举算法^[21]。二分图转化为简单图后,其中的 bi-triangle 转化为带权三角形,从而可以利用 PTE 算法进行分布式枚举。DBC 算法^[25]和 Monarch 算法^[32]则是近年被提出的 butterfly 分布式计数算法,这两种算法的核心处理步骤都是通过分布式聚合顶点与其二阶邻接点来构建关联网络。在完成这一步骤后, DBC 算法和 Monarch 算法可将顶点及其二阶邻接点的有序对输入算法 2 枚举 bi-triangle。

3) 实验数据集。

实验共使用 6 个二分图数据集,均来源于公开数据网站 Konect(<http://konect.cc>)。数据集的具体组成情况如表 1 所示。

4) 实验任务。

实验任务分为 4 组:

实验 1 子图组合数据量对比。该实验首先统

表 1 实验数据集信息

Table 1 Experimental dataset information

二分图名称	A	V	E	wedge
Marvel	6 486	19 428	25 914	12 869 214
YouTube	94 238	30 087	124 325	70 180 198
Team	901 106	34 461	935 627	336 279 376
BookCross	105 278	340 523	455 801	465 125 366
Genre	7 783	258 934	266 717	1 331 590 538
CiaoDvd	21 019	71 633	92 652	2 730 969 169

计各测试算法在分布式枚举 bi-triangle 过程中产生的子图组合数据量,然后计算该数据量与原始二分图规模的比值,以便有效比较不同算法在数据存储和传输效率上的性能差异。需要注意的是,PTE 算法需要先将二分图转换为普通图(这一转换过程会导致图规模呈指数级增长),其数据膨胀机制与其他直接处理原生二分图的实验算法存在本质差异。为避免因基准不一致而导致的结果失真,该实验未纳入 PTE 算法的相关数据。

实验 2 枚举时间对比。该实验旨在比较 6 种测试算法在实验数据集上完成 bi-triangle 分布式枚举所需的时间,以评估各算法的计算效率

实验 3 参数 p 的变动测试。该实验以 BookCross 数据集为测试数据,测试参数 p (即分图后子图的数量)变化时,WP 算法和 ZP 算法完成枚举所需时间的变化趋势,从而分析两者对参数 p 调整的敏感性。

实验 4 扩展性测试。该实验以 YouTube 和 BookCross 数据集为测试数据,通过统计 WP 算法和 ZP 算法在不同物理计算核配置下分布式枚举 bi-triangle 的运行时间,以评估两者的扩展性表现。

5.2 实验结果与分析

实验 1 的结果如图 4 所示。结果表明,在分布式枚举 bi-triangle 的过程中,不同测试算法构造的子图组合数据规模存在显著差异,其中 WP 算法生成的数据规模最小。

与 GP 算法相比,WP 算法以 wedge 组为基本分图单位,实现了子图组合数量与规模的大幅下降。实验数据显示,采用随机分图策略的 GP 算法的子图组合数据量平均是 WP 算法的 205 倍。

与 DBC 和 Monarch 算法相比,WP 算法在枚举 bi-triangle 时同时采用“聚合顶点与其二阶邻接点”的处理思路,但通过实现机制的优化,WP 算法生成的子图组合数据量更低。具体而言,对于任意 wedge 组 $\omega=(u, N(u))$,其中 $u \in A$,DBC 算法采

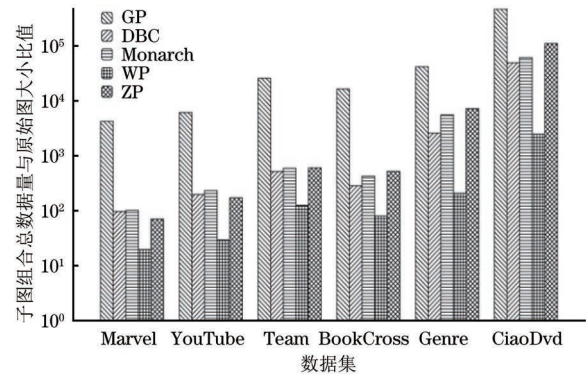


图 4 GP、DBC、Monarch、WP 和 ZP 算法在实验二分图数据集上分布式枚举 bi-triangle 产生的子图组合数据量

Fig. 4 Subgraph combination data size generated by GP, DBC, Monarch, WP, and ZP algorithms for distributed bi-triangle enumeration on experimental bipartite graph datasets

用两阶段消息传递机制处理该 wedge 组,在第一阶段,拐点 u 向所有 $\gamma \in N(u)$ 发送消息,在第二阶段,接收方再将 u 的标识转发给满足条件 $u. id < \alpha. id$ 的二阶邻接点 $\alpha \in N(\gamma)$ 。而 Monarch 算法将 $N(u)$ 中的元素两两配对,以聚合这些元素及其二阶邻接点。理论上,DBC 和 Monarch 两种算法产生的子图组合数据量级均为 $O\left(\sum_{a \in A} (d^2(a))\right)^{[25,32]}$ 。相比之下,WP 算法首先将以 $\gamma \in N(u)$ 为拐点的 V 型 wedge 组预存入分布式存储系统,在后续聚合 u 和 $N(\gamma)$ 时,直接从该系统中读取预存数据。根据第 3.2 节的分析,WP 算法产生的子图组合数据量级最大为 $O\left(\min(p, d_{ag}) \cdot \sum_{a \in A} (d(a))\right)$ 。在顶点度数均匀分布的理想情况下,两种数据量级大致相当。但在实际的二分图网络中,顶点度数通常呈现不均匀分布,高度数顶点在求和平方时会显著放大数据量级,从而使得 $O\left(\sum_{a \in A} (d^2(a))\right)$ 远大于 $O(\min(p, d_{ag}) \cdot \sum_{a \in A} (d(a)))$ 。实验结果验证了这一分析:WP 算法平均构造规模仅为 DBC 的 16.7%、Monarch 的 13.2%。

与 ZP 算法相比,WP 算法在所有实验二分图数据集中构造的子图组合数据量始终更低。这一性能差异主要源于两种算法采用了不同的分图策略:WP 算法仅以 wedge 组为基本单位进行一次分图,而 ZP 算法则先后以 wedge 组和 zedge 组为基本单位执行两次分图。

实验 2 的结果如图 5 所示。结果表明,在 bi-triangle 分布式枚举任务中,不同测试算法的耗时差异显著,其中 ZP 算法耗时最短。

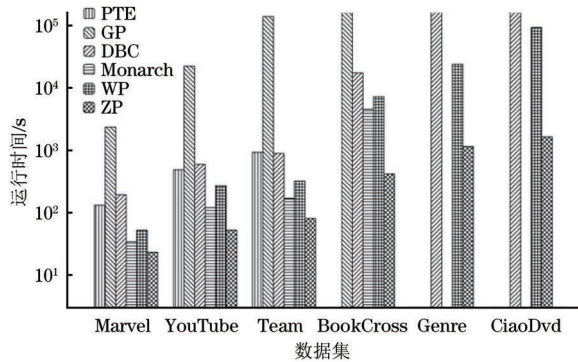


图 5 PTE、GP、DBC、Monarch、WP 和 ZP 算法在实验二分图数据集上分布式枚举 bi-triangle 所用时间

Fig.5 Time taken by PTE, GP, DBC, Monarch, WP, and ZP algorithms for distributed bi-triangle enumeration on experimental bipartite graph datasets

PTE 算法和 GP 算法在任务执行过程中表现出了明显的内存与效率瓶颈。使用 PTE 算法枚举 bi-triangle 时需要将二分图转换为普通图,这一过程中图的规模会呈指数级增长,极易导致内存溢出。受限于此,PTE 算法仅能完成前 3 个小规模数据集的计算,且效率低下,运行时间平均为 WP 算法的 2.4 倍,ZP 算法的 8.8 倍。GP 算法则依赖节点间的消息传递来构造子图组合。由于所需的子图组合数量高达 $C(p, 6)$,随着数据集规模的增大,消息量呈指数级增长,引发消息过载和重复计算问题。在 BookCross 数据集中,GP 算法耗时超过 72 h,因而作截断统计;在 Genre 和 CiaoDvd 数据集上,GP 算法则因消息缓存溢出导致任务失败。实验证明这两种算法难以应对大规模 bi-triangle 枚举任务。

DBC 算法虽然与 WP 算法一样能够处理全部实验数据集,但效率不如 WP 算法。DBC 算法在运行过程中,每个顶点需要与其二阶邻接点进行两次消息传递。为了控制消息传递规模,防止内存溢出,DBC 算法采用了分轮次计算策略,即每轮仅处理部分节点。这种设计虽然有效避免了单次内存峰值,却严重制约了算法的枚举效率。以 BookCross 数据集为例,DBC 算法的枚举耗时达到 WP 算法的 2.4 倍,ZP 算法的 11 倍。而在 Genre 和 CiaoDvd 数据集上,DBC 算法因超过 72 h 的运行时间而失去了实际应用价值,进一步凸显了其在大规模二分图处理中的局限性。

Monarch 算法的枚举效率仅次于 ZP 算法,但其适用性有限,无法应对所有实验数据集。在前 4 个数据集上,该算法耗时保持次优水平(如其在 Team 数据集上的耗时仅为 WP 算法的 53%)。然而,随着实验数据集规模的扩大,算法的性能受到显

著制约。具体而言,Monarch 算法在枚举过程中需要为每个计算节点维护哈希表,并通过 shuffle 聚合操作将其合并为总表进行计算。该机制在前 4 个实验数据集上表现良好,但随着实验数据规模的增加,shuffle 过程的内存需求呈指数级增长,导致在 Genre 和 CiaoDvd 数据集上,因产生内存溢出而无法完成枚举任务。这种内存瓶颈严重限制了 Monarch 算法在大规模二分图上的适用性。

WP 算法在 72 h 内完成了全部数据集的枚举任务。通过第 3.1 节的 3 条优化策略,WP 算法实现了全数据集覆盖,其效率在前 3 个数据集上比 GP 算法平均提升了 187 倍。尽管在处理后 3 个大规模数据集的耗时呈指数级增长,但与 DBC 和 Monarch 算法相比,WP 算法的增速更为平缓,且未出现内存异常。尤其是在 Genre 和 CiaoDvd 数据集上,当 DBC 算法因多轮计算耗时超过 72 h、Monarch 算法因内存溢出失效时,WP 算法仍能稳定完成任务,充分展现了其在大规模二分图数据处理中的可靠性。

ZP 算法在所有测试集中枚举所需时间最少。从时间复杂度来看,WP、DBC、Monarch 3 种算法虽然采用了不同的分布式计算策略,但最终枚举 bi-triangle 的核心操作均是将顶点及其二阶邻接点的有序对输入算法 2 进行处理,因此它们的计算复杂度主项相同,均为 $\sum_{a \in A} (d_2^2(a) \cdot d_{ag}^3)$ 。相比之下,ZP 算法以 zedge 组为枚举单元,其计算复杂度主项是 $\sum_{a \in A} (d(a) \cdot d_2(a) \cdot d_{ag}^3)$,具有显著的性能优势。具体到实验数据,相较于 GP 算法,ZP 算法平均提升了 751 倍效率;在 BookCross 数据集中,ZP 算法耗时较 DBC 算法缩短 97.6%;在 Monarch 算法能够处理的前 4 个数据集中,ZP 算法的平均耗时缩减了 58.0%;此外,在 BookCross、Genre 和 CiaoDvd 数据集中,ZP 算法的枚举效率分别是 WP 算法的约 17、21 和 57 倍,这证明了其在大规模二分图上分布式枚举 bi-triangle 的高效性。

在实验 1 中,WP 算法构造的子图组合数据量在所有对比算法中最低;在实验 2 中,ZP 算法的枚举耗时在所有对比算法中最短。这一结果验证了第 4.3 节中关于 WP 和 ZP 算法性能的预期,即 WP 算法在子图组合数量和规模控制方面优于 ZP 算法,而 ZP 算法在计算速度上表现更快。为更具体地探究这一性能差异,图 6 统计了实验数据集中 WP 与 ZP 算法的子图组合数据量比值(ZP/WP)及枚举时间比值(WP/ZP)。数据显示,在分布式枚举过程中,WP 算法构造的子图组合数据量平均为 ZP 算法

的 14.4%，而 ZP 算法的枚举耗时平均仅为 WP 算法的 16.8%。此外，图 6 进一步表明，相较于 WP 算法，ZP 算法在子图组合数据量上的增长幅度与枚举时间的减少幅度处于同一数量级。这意味着，ZP 算法计算效率的提升伴随更高的数据存储和传输成本。该实证结果为第 4.3 节中关于 WP 算法和 ZP 算法场景适用性的分析提供了有力支持：在通信成本受限的分布式系统中，WP 算法的低数据传输特性更具优势；而在计算资源充裕且对计算速度要求较高的场景中，ZP 算法的高效计算能力则更为适用。

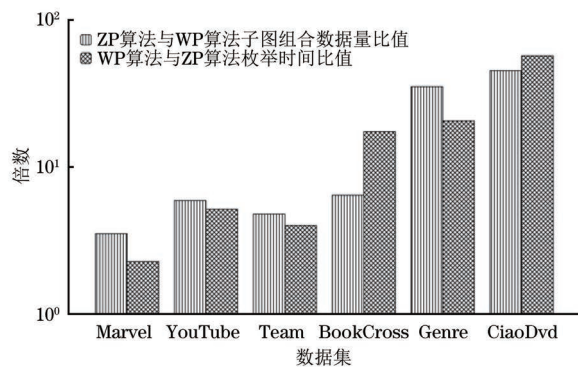


图 6 在实验二分图数据集上 WP 与 ZP 算法子图组合数据量及枚举时间比值

Fig. 6 Subgraph combination data size and enumeration time ratio between WP and ZP algorithms on experimental bipartite graph datasets

实验 3 的结果如图 7 所示。实验结果表明，WP 和 ZP 两种算法的运行时间随子图数量变化呈现先下降后上升的趋势。具体而言，ZP 算法和 WP 算法的运行时间曲线在初期较高，随后随着子图数量的增加逐渐下降，并在某一范围内达到最低点。当子图数量较少时，计算集群的运算核未被完全占用，导

致算力未得到充分利用。随着子图数量增加，计算资源逐渐得到更高效的利用，运行时间因此降低。然而，当子图数量继续增加并超过集群的承载能力时，子任务的执行开始出现积压，从而导致运行时间再次上升。

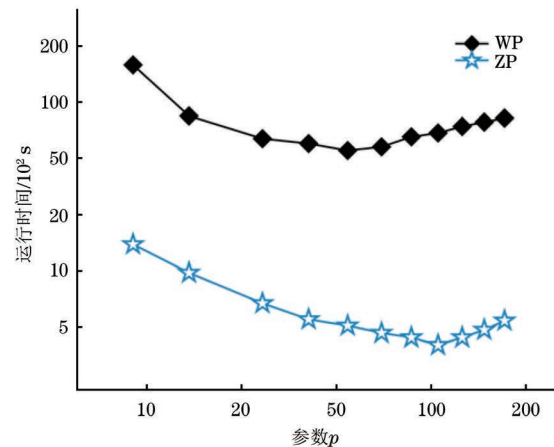


图 7 在 BookCross 数据集上 WP 和 ZP 算法配置不同参数 p 时分布式枚举 bi-triangle 所用时间

Fig. 7 Distributed bi-triangle enumeration time of WP and ZP algorithms with different parameter p configurations on the BookCross dataset

实验 4 的结果如图 8 所示(彩色效果见《计算机工程》官网 HTML 版,下同)。实验结果表明,随着计算物理核数量的增加,WP 和 ZP 算法的运行时间均呈下降趋势,且下降幅度整体接近线性,表现出良好的扩展性。具体而言,WP 算法在两个数据集上表现出较好的初期扩展性,但随着核数的增加,运行时间下降幅度逐渐减小,尤其在低核数时,下降速率放缓。相比之下,ZP 算法展现出更均匀的扩展性,尤其是在高核数条件下,运行时间持续稳定下降,其扩展性显著优于 WP 算法。

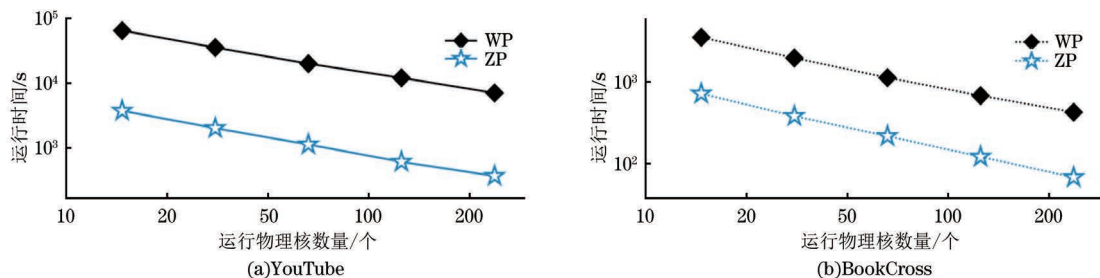


图 8 在 YouTube 和 BookCross 数据集上 WP 和 ZP 算法配置不同计算物理核时分布式枚举 bi-triangle 所用时间

Fig. 8 Distributed bi-triangle enumeration time of WP and ZP algorithms with different computational physical core configurations on YouTube and BookCross datasets

6 结束语

在分布式枚举 bi-triangle 任务中,现有 GP 算法的效率低下。对此,本文首先提出 WP 算法,该

算法设计了以 wedge 组为单位分图、分布式存储子图以及拼接 wedge 组聚合二阶邻接点等策略;接着本文提出以 zedge 组作为分图与枚举单位的 ZP 算法,该算法先后以 wedge 组和 zedge 组为基本单位

执行两次分图,具有比 WP 算法更低的枚举复杂度。实验结果表明,WP 算法和 ZP 算法在冗余控制、子图数据量削减和枚举效率等方面均显著优于 GP 算法。值得注意的是,这两种算法在子图划分时基于顶点哈希值。未来研究可进一步探讨如何结合顶点的度数信息实现更优的负载均衡。此外,还可探索将 WP 和 ZP 算法与机器学习等技术相结合,推动其在高效分布式枚举任意二分图结构中的应用。

参考文献

- [1] HUANG J J, SHEN H W, CAO Q, et al. Signed bipartite graph neural networks[C]//Proceedings of the 30th ACM International Conference on Information & Knowledge Management. New York, USA: ACM Press, 2021: 740-749.
- [2] WU Z T, SONG C Y, CHEN Y Q, et al. A review of recommendation system research based on bipartite graph[C]//Proceedings of MATEC Web of Conferences. [S. l.]: EDP Sciences, 2021: 05010.
- [3] BORGATTI S P, EVERETT M G. Network analysis of 2-mode data[J]. Social Networks, 1997, 19(3): 243-269.
- [4] LATAPY M, MAGNIEN C, VECCHIO N D. Basic notions for the analysis of large two-mode networks[J]. Social Networks, 2008, 30(1): 31-48.
- [5] XIE G B, LI J M, GU G S, et al. BGMSDDA: a bipartite graph diffusion algorithm with multiple similarity integration for drug-disease association prediction[J]. Molecular Omics, 2021, 17(6): 997-1011.
- [6] CHI C, YE Y T, CHEN B, et al. Bipartite graph-based approach for clustering of cell lines by gene expression-drug response associations[J]. Bioinformatics, 2021, 37(17): 2617-2626.
- [7] CHEN W, WANG H J, LONG Z G, et al. Fast flexible bipartite graph model for co-clustering[J]. IEEE Transactions on Knowledge and Data Engineering, 2022, 35(7): 6930-6940.
- [8] ZHANG H, NIE F P, LI X L. Large-scale clustering with structured optimal bipartite graph[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2023, 45(8): 9950-9963.
- [9] MAIER C, SIMOVICI D. Bipartite graphs and recommendation systems[EB/OL]. [2024-12-07]. <https://www.jait.us/uploadfile/2022/0428/20220428061814827.pdf>.
- [10] GURUKAR S, PANCHAN, ZHAI A, et al. MultiBiSage: a Web-scale recommendation system using multiple bipartite graphs at pinterest[EB/OL]. [2024-12-07]. <https://arxiv.org/abs/2205.10666>.
- [11] YANG J Y, PENG Y, ZHANG W J. (p, q) -biclique counting and enumeration for large sparse bipartite graphs[J]. Proceedings of the VLDB Endowment, 2021, 14(2): 141-153.
- [12] CHEN L, LIU C F, ZHOU R, et al. Efficient maximal biclique enumeration for large sparse bipartite graphs[J]. Proceedings of the VLDB Endowment, 2022, 15(8): 1559-1571.
- [13] SUN R J, WU Y P, CHEN C, et al. Maximal balanced signed biclique enumeration in signed bipartite graphs[C]//Proceedings of the IEEE 38th International Conference on Data Engineering (ICDE). Washington D. C., USA: IEEE Press, 2022: 1887-1899.
- [14] MILO R, SHEN-ORR S, ITZKOVITZ S, et al. Network motifs: simple building blocks of complex networks[J]. Science, 2002, 298(5594): 824-827.
- [15] OPSAHL T. Triadic closure in two-mode networks: redefining the global and local clustering coefficients[J]. Social Networks, 2013, 35(2): 159-167.
- [16] YANG Y X, FANG Y X, ORLOWSKA M E, et al. Efficient bi-triangle counting for large bipartite networks[J]. Proceedings of the VLDB Endowment, 2021, 14(6): 984-996.
- [17] ZHANG F Y, CHEN D C, WANG S B, et al. Scalable approximate butterfly and bi-triangle counting for large bipartite networks[J]. Proceedings of the ACM on Management of Data, 2023, 1(4): 1-26.
- [18] YANG Y X, FANG Y X, LIN X M, et al. Effective and efficient truss computation over large heterogeneous information networks[C]//Proceedings of the IEEE 36th International Conference on Data Engineering (ICDE). Washington D. C., USA: IEEE Press, 2020: 901-912.
- [19] ARIFUZZAMAN S, KHAN M, MARATHE M. A space-efficient parallel algorithm for counting exact triangles in massive networks[C]//Proceedings of the IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems. Washington D. C., USA: IEEE Press, 2015: 527-534.
- [20] AHMED A, ENNS K, THOMO A. Triangle enumeration for billion-scale graphs in RDBMS[C]//Proceedings of International Conference on Advanced Information Networking and Applications. Berlin, Germany: Springer, 2021: 160-173.
- [21] LIU X Z, SANTOSO Y, SRINIVASAN V, et al. Practical survey on MapReduce subgraph enumeration algorithms[C]//Proceedings of the International Conference on Emerging Networking, Data & Web Technologies. Berlin, Germany: Springer, 2022: 430-444.
- [22] SARIYÜCE A E, PINAR A. Peeling bipartite networks for dense subgraph discovery[C]//Proceedings of the 11th ACM International Conference on Web Search and Data Mining. New York, USA: ACM Press, 2018: 504-512.
- [23] WANG Y, XU R Q, JIAN X, et al. Towards distributed bitruss decomposition on bipartite graphs[J]. Proceedings of the VLDB Endowment, 2022, 15(9): 1889-1901.
- [24] LIU Q, LIAO X K, HUANG X, et al. Distributed (α, β) -core decomposition over bipartite graphs[C]//Proceedings of the IEEE 39th International Conference on Data Engineering (ICDE). Washington D. C., USA: IEEE Press, 2023: 909-921.
- [25] 周旭, 翁同峰, 杨志邦, 等. 面向大规模二部图的分布式 Tip 分解算法[J]. 软件学报, 2022, 33(3): 1043-1056.
- [26] ZHOU X, WENG T F, YANG Z B, et al. Distributed algorithm for Tip decomposition on large bipartite graphs[J]. Journal of Software, 2022, 33(3): 1043-1056. (in Chinese)
- [27] NIU J, ZOLA J, SARIYÜCE A E. Counting induced 6-cycles in bipartite graphs[C]//Proceedings of the 51st International Conference on Parallel Processing. New York, USA: ACM Press, 2022: 1-10.
- [28] WANG J, FU A W, CHENG J. Rectangle counting in large bipartite graphs[C]//Proceedings of the IEEE International Congress on Big Data. Washington D. C., USA: IEEE Press, 2014: 17-24.
- [29] SANEI-MEHRI S V, SARIYÜCE A E, TIRTHAPURA S. Butterfly counting in bipartite networks[C]//Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. New York, USA:

- ACM Press, 2018: 2150-2159.
- [29] WANG K, LIN X M, QIN L, et al. Vertex priority based butterfly counting for large-scale bipartite networks [J]. Proceedings of the VLDB Endowment, 2019, 12(10): 1139-1152.
- [30] SHI J, SHUN J. Parallel algorithms for butterfly computations[EB/OL]. [2024-12-07]. <https://arxiv.org/abs/1907.08607>.
- [31] PAPADIAS S, KAOUDI Z, PANDEY V, et al. Counting butterflies in fully dynamic bipartite graph streams [C] // Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE). Washington D. C., USA: IEEE Press, 2024: 2917-2930.
- [32] TANG Y X, BENDRE M, DAS M. Monarch: distributed butterfly counting for large-scale bipartite graph [C] // Proceedings of the IEEE International Conference on Big Data (BigData). Washington D. C., USA: IEEE Press, 2024: 799-804.
- [33] ZHOU A, WANG Y, CHEN L. Butterfly counting on uncertain bipartite graphs [J]. Proceedings of the VLDB Endowment, 2021, 14(2): 211-223.
- [34] CAI X W, KE X Y, WANG K, et al. Efficient temporal butterfly counting and enumeration on temporal bipartite graphs [EB/OL]. [2024-12-07]. <https://arxiv.org/abs/2306.00893>.
- [35] LIU B G, YUAN L, LIN X M, et al. Efficient (α, β) -core computation: an index-based approach [C] // Proceedings of the World Wide Web Conference. New York, USA: ACM Press, 2019: 1130-1141.
- [36] WANG K, LIN X M, QIN L, et al. Towards efficient solutions of bitruss decomposition for large-scale bipartite graphs[J]. The VLDB Journal, 2022, 31(2): 203-226.
- [37] 代强强, 于瀚文, 李荣华, 等. 面向二部图的极大缺陷二团高效枚举算法[J]. 软件学报, 2025, 36(4): 1796-1810.
- DAI Q Q, YU H W, LI R H, et al. Efficient algorithms for maximal defective biclique enumeration on bipartite graphs [J]. Journal of Software, 2025, 36(4): 1796-1810. (in Chinese)
- [38] 赵兴旺, 薛晋芳. 基于二部图表示的属性网络社区发现算法[J]. 计算机科学, 2023, 50(11): 107-113.
- ZHAO X W, XUE J F. Community discovery algorithm for attributed networks based on bipartite graph representation [J]. Computer Science, 2023, 50(11): 107-113. (in Chinese)

文字编辑 陆燕菲

栏目编辑 宋 圆