软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(3):1043−1056 [doi: 10.13328/j.cnki.jos.006457]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向大规模二部图的分布式 Tip 分解算法*

周 旭[1], 翁同峰[1], 杨志邦[1], 李博仁[1], 张 吉[3], 李肯立[1,2]

[1](湖南大学 信息科学与工程学院, 湖南 长沙 410082)

[2](国家超级计算长沙中心, 湖南 长沙 410082)

[3](之江实验室, 浙江 杭州 311100)

通信作者: 翁同峰, E-mail: wengtongfeng@hnu.edu.cn

摘 要: Tip 分解作为图数据管理领域的热点研究问题, 已被广泛应用于文档聚类和垃圾邮件组检测等实际场景中. 随着图数据规模的爆炸式增长, 单机内存已无法满足其存储需求, 亟需研究分布式环境下 Tip 分解技术. 现有分布式图计算系统的通信模式无法适用于二部图, 为此, 首先提出一种基于中继的通信模式, 以实现分布式环境下处理二部图时消息的有效传递; 其次, 提出分布式 butterfly 计数算法(DBC)和 tip 分解算法(DTD), 特别地, 为解决处理大规模二部图时 DBC 面临的内存溢出问题, 提出了一种可控的并行顶点激活策略; 最后, 引入基于顶点优先级的消息剪枝策略和消息有效性剪枝策略, 通过减少冗余通信和计算开销, 进一步提高算法效率. 实验平台部署于国家超算中心高性能分布式集群上, 在多个真实数据集上的实验结果验证了所提算法的有效性和高效性.

关键词: 二部图; butterfly 计数; 分布式系统; tip 分解

中图法分类号: TP311

中文引用格式: 周旭, 翁同峰, 杨志邦, 李博仁, 张吉, 李肯立. 面向大规模二部图的分布式 Tip 分解算法. 软件学报, 2022, 33(3): 1043–1056. http://www.jos.org.cn/1000-9825/6457.htm

英文引用格式: Zhou X, Weng TF, Yang ZB, Li BR, Zhang J, Li KL. Distributed Algorithm for Tip Decomposition on Large Bipartite Graphs. Ruan Jian Xue Bao/Journal of Software, 2022, 33(3): 1043−1056 (in Chinese). http://www.jos.org.cn/1000-9825/6457.htm

Distributed Algorithm for Tip Decomposition on Large Bipartite Graphs

ZHOU Xu[1], WENG Tong-Feng[1], YANG Zhi-Bang[1], LI Bo-Ren[1], ZHANG Ji[3], LI Ken-Li[1,2]

[1](College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China)

[2](National Supercomputing Center in Changsha, Changsha 410082, China)

[3](Zhejiang Lab, Hangzhou 311100, China)

Abstract: Tip decomposition has a pivotal role in mining cohesive subgraph in bipartite graphs. It is a popular research topic with wide applications in document clustering, spam group detection, and analysis of affiliation networks. With the explosive growth of bipartite graph data scale in these scenarios, it is necessary to use distributed method to realize its effective storage. For this reason, this work studies the problem of tip decomposition on a bipartite graph in the distributed environment for the first time. Firstly, a new relay based communication mode is proposed to realize effective message transmission when the given bipartite graph is decomposed in distributed environment. Secondly, the distributed butterfly counting algorithm (DBC) and tip decomposition algorithm (DTD) are designed. In particular, a controllable parallel vertex activation strategy is proposed to solve the problem of memory overflow when DBC decomposes large-scale bipartite graphs. Finally, the message pruning strategy based on vertex priority and message validity pruning strategy are introduced to further improve the efficiency of the algorithm by reducing redundant communication and computing overhead. The

* 基金项目: 国家自然科学基金(61772182, 61802032, 69189338, 62172146, 62172157); 之江实验室开放课题(2021KD0AB02); 国防科技大学信息系统工程重点实验室基金

本文由“数据库系统新型技术”专题特约编辑李国良教授、于戈教授、杨俊教授和范举教授推荐.

收稿时间: 2021-06-30; 修改时间: 2021-07-31; 采用时间: 2021-09-13; jos 在线出版时间: 2021-10-21

experiment is deployed on the high performance distributed computing platform of National Supercomputing Center. The effectiveness and efficiency of the proposed algorithms are verified by experiments on several real datasets.

Key words: bipartite graph; butterfly counting; distributed system; tip decomposition

二部图是一种特殊的图类型，图中顶点集可分为两个不相交子集 U 和 V，仅不同子集中的顶点间存在边，即 $G=(U,V,E)$，满足$\{e=(u,v)\in E|u\in U\wedge v\in V\}$．近年来，针对二部图中内聚子图的挖掘技术受到广泛关注，并被应用于文档聚类[1]、作者-论文关系分析、用户-产品关系分析[2]和垃圾邮件组检测[3]等实际应用中．

目前，针对简单图上 k-core[4,5]和 k-truss[6]等稠密子图结构的分解技术已存在大量研究．这些技术可用于分析由二部图转换获得的简单图[7]，但会造成原始结构信息的丢失．此外，将二部图转化为简单图时，图的规模将增加 6 个数量级[8]，进而造成大量额外空间存储开销．因此，现有针对简单图的分解技术无法适用于大规模二部图数据，需研究针对二部图的稠密子图分解算法．

Butterfly 子图是二部图中最小的紧密子图结构，其包含分别来自于两个顶点集合(U/V)的 4 个顶点组成的二部团，各顶点对应的 butterfly 子图数量即为该顶点的 butterfly 度．可见，任意两个同侧顶点共享的 butterfly 子图越多，则说明二者联系越紧密．该结构可用于衡量二部图中同侧顶点间的稠密关系．k-tip 被定义为一种紧密社区 $H=(U',V',E')$，满足任意 $u\in U'$存在于至少 k 个 butterfly 子图中．对于一个特定顶点 u，若其只能存在于 k-tip 而不能存在于 k+1-tip 中，则 u 的 tip 值为 k．本文研究的 tip 分解算法用于计算给定图中所有顶点的 tip 值[9]．

Sariyuce 等人[8]提出了一种基于层次剥离算法用于分解给定的二部图．该算法迭代删除 butterfly 度最小的顶点，而顶点被删减时的 butterfly 度即为其 tip 值．如图 1(a)所示，V 中有 4 个顶点 v_1, v_2, v_3 和 v_4，U 中有 5 个顶点 u_1, u_2, u_3, u_4 和 u_5，其中，顶点 u_1, u_2, v_1 和 v_2 构成了一个 butterfly 子图(即(2,2)-二部团)．若 G 中每个顶点 $u\in U$，至少存在于一个 butterfly 子图中，则 G 为 1-tip 社区．类似地，图 1(c)和图 1(d)中的子图分别为 2-tip 和 3-tip 社区．集合 U 中每个顶点的 tip 值如图 1(e)所示．

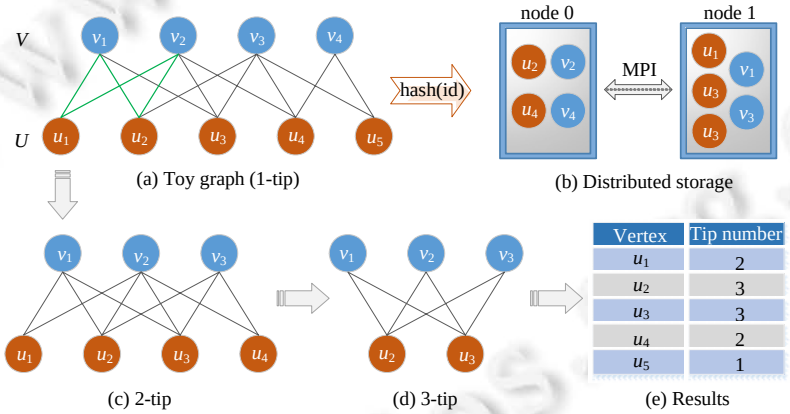


图 1　二部图的分布式 tip 分解示例图

挑战：现有的 butterfly 子图计数(简称 butterfly 计数)算法和 tip 分解算法[10−12]建立在单机内存可以满足原始图数据和中间计算结果存储需求的基础上．随着二部图规模的不断增长，单机内存已难以满足大规模二部图数据的存储需求．为此，需对大规模图数据进行分布式存储，并研究相应的分布式二部图处理技术；其次，由于二部图结构特殊性，两个顶点集合中存在大量 id 相同的顶点，而现有分布式图计算系统主要针对简单图，其通信模式依赖于顶点 id，因而在处理二部图时会出现消息发送/接收错乱的情况，导致错误的计算结果；最后，二部图中同侧顶点间无直接边连接，而 tip 分解旨在挖掘同侧顶点间的稠密关系，需通过二跳消息传输才能实现 butterfly 计数算法和 tip 分解算法，这将导致大量的通信开销，进而影响分解的效率．

为应对上述挑战，本文探索分布式环境下面向大规模二部图的 butterfly 子图计数和 tip 分解算法，其中，

分布式存储环境可以有效地解决大规模二部图的原始数据存储以及中间计算结果的存储需求. 基于分布式计算环境, 本文设计了一种基于中继的通信模式, 通过中继顶点转发消息, 从而保证消息传递的有效性, 以解决现有分布式图计算系统依赖顶点 id 进行消息传递的局限. 具体来说, 给定一个二部图 $G=(U,V,E)$, 计算 U 中所有顶点的 tip 值. 将 V 中的顶点作为中继顶点, 其不需要执行计算, 仅负责转发接收到的消息, 而 U 中的顶点根据需要执行不同的计算功能. 由于中继顶点(V 侧顶点)可直接获取待分解侧顶点(U 侧顶点)信息, 如顶点的度、tip 值等, 则可根据这些属性信息设计消息剪枝策略以减少额外的通信开销, 进而减少下一轮迭代中的计算量. 据此, 本文提出了一种基于中继通信方式的分布式 butterfly 计数算法, 并依据顶点优先级引入一种消息剪枝策略来减少冗余通信开销; 随后, 基于 butterfly 计数结果设计了一种基于层次剥离的分布式 tip 分解算法来计算所有顶点的 tip 值. 在 tip 分解过程, 通过维护一个 butterfly 树 *BFTree*, 将顶点按 butterfly 度分层存储; 最后, 基于 *BFTree* 的最小键剥离对应叶子中的顶点, 当 *BFTree* 为空时, 即可得到所有顶点的 tip 值.

本文的主要贡献如下:

- 设计了一种基于中继的通信模式, 并构建了一个适用于二部图紧密子图分析的分布式图计算系统.
- 提出了一种高效的分布式 butterfly 计数算法(distributed butterfly counting (DBC)). 特别地, 为解决 DBC 分解大规模二部图数据时内存溢出问题, 引入了可控的并行顶点激活策略, 从而实现有效计算 butterfly 计数的同时提高算法并行度.
- 提出了一种基于 butterfly 树 *BFTree* 的分布式 tip 分解算法(distributed tip decomposition (DTD)), 为进一步提高算法效率, 引入基于顶点优先级的消息剪枝策略以减少冗余通信和计算开销.
- 在国家超级计算长沙中心部署分布式图计算系统并实现本文所设计算法, 在多个真实数据集上验证了所提算法的有效性和高效性.

本文第 1 节总结相关工作, 介绍分布式图计算系统、butterfly 计数和 tip 分解问题的相关算法. 第 2 节给出本文使用到的相关定义与问题定义. 本文提出的基于中继的通信模式和分布式 butterfly 计数算法在第 3 节中提出. 第 4 节阐述分布式 tip 分解算法. 实验结果与分析在第 5 节中给出. 最后, 在第 6 节对本文的研究进行总结与展望.

1 相关工作

本节主要介绍分布式图计算系统、二部图 butterfly 计数算法和 tip 分解算法的相关工作.

1.1 分布式图计算系统

分布式图计算系统多基于 BSP 模型设计, 是一种适用于图计算任务的计算模式. 如图 2 所示, 超步作为基本迭代单元, 它由 3 个步骤组成: 接收消息、计算和发送消息. 在每个超步的末尾, 都有一个用于同步消息的全局栅栏.

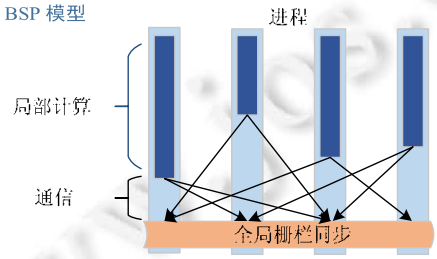


图 2 BSP 模型

基于 BSP 模型、Pregel[13]、Giraph[14]、GraphX[15]和 Pregel+[16]等以顶点为中心计算模式的图计算系统相继提出. 面向大规模图数据, 于戈等人[17]就图数据管理与处理机制两个方面, 综述了大规模图数据处理中计算模型、通信机制、分割策略、索引结构和容错管理等关键问题. 陆李等人[18]提出了度变异系数和分片通达

度两个特征参数, 用于优化 CPU+GPU 异构环境下图计算系统研究平台和负载特征感知的在线分割算法的性能. 张程鹏等人[19]根据分布式图计算框架中存在的不确定性因素和鲁棒性问题, 提出了基于成本、效率和质量这 3 个维度的容错技术评估框架.

与面向简单图的图计算系统[20−22]和图算法[23−25]相比, 为处理二部图而设计的系统和图算法要少得多. Chen 等人[26]提出了一个在 PowerGraph 上实现的以顶点为中心的系统 BiGraph, 目的在于为机器学习算法有效地划分二部图. Liu 等人[27]则设计了一种高效的在二部图上用于计算(α-β)-core 的索引构建和维护算法, 可以用于对动态二部图的分析. 但是这些系统与算法并没有为 butterfly 计数算法引入特殊的优化策略.

综上, 现有的图计算系统和图算法不能有效地用于处理 butterfly 计数和 tip 分解问题.

1.2 二部图butterfly计数和tip分解

Butterfly 子图是二部图中重要的图结构, 广泛应用于 k-tip 社区发现等二部图结构挖掘中. 本文主要研究二部图上 butterfly 计数和 tip 分解算法.

为了优化 butterfly 计数算法, Chiba 和 Nishizeki[28]设计了一种顶点优先的计数算法, 优化了图中边搜索过程. Wang 等人[9]进一步将缓存感知策略引入到顶点优先级算法中, 以降低时间复杂度. Sanei-Mehri 等人[11]则是提出了一种用于 butterfly 计数的近似算法, 并在准确性上给出了证明.

为了加速 tip 分解, Jessica Shi 和 Julian Sun[9]提出了 ParButterfly 框架, 基于 OpenMP 众核技术分别设计了用于 butterfly 计数和 tip 分解问题的并行算法. Lakhotia 等人[12]提出了一种由粗粒度分解和细粒度分解组成的共享内存的并行 tip 分解算法, 获得了更高并行度, 提高多核系统的资源利用率.

上面提到的算法都基于如下假设: 单台机器的内存足够大, 可以存储给定的二部图及中间计算结果. 然而, 随着图规模的爆炸性增长, 单台机器的内存瓶颈已经不能被忽视. 多台机器不仅可以提供可扩展的分布式存储空间, 还可以拥有高性能的计算资源, 用于提高算法的并行性.

此外, 虽然现有的算法可以有效地处理 butterfly 计数和 tip 分解问题, 但如果应用于分布式系统, 顶点重标记和重排序操作将引入大量的通信开销. 例如, 在图 1(a)中, 顶点根据 id 的哈希值分布式存储在集群中(即图 1(b)). 现有算法需要按照度数降序重新标记 U 中的顶点. 在节点 0 中, u_4 的标号小于 u_2, 因为它们具有相同的度数, u_4 的 id 大于 u_2. 类似地, 在节点 1 中, 标号(u_3)<标号(u_5)<标号(u_1). 这些是每台机器中顶点的局部标号. 如果我们想要得到所有顶点的全局标号, 不同的机器需要对局部标号从最大到最小进行比较和重新排序. 该操作具有很高的时间复杂度并将产生大量的通信开销.

2 问题与定义

本节主要介绍有关二部图和 tip 分解的相关理论. 表 1 列出了所用符号名称及其描述.

表 1 符号概述

名称	描述
$G=(U,V,E)$	一个由顶点集 U, V 和边集 E 组成的二部图
N_u	顶点 u 的邻居
$d_G(u)$	顶点 u 的邻居数量
$H=(U',V',E')$	图 G 的子图,简称 H
$\bowtie_u$	一个包含顶点 u 的 butterfly(子图)
$d_{\bowtie}$	某个顶点所参与的 butterfly 数量
$d^u_{\triangleright\triangleleft}$	包含顶点 u 的 butterfly 数量, 即 butterfly 度
$\triangleright\triangleleft^{u_2}_{u_1}$	一个包含 u_1 和 u_2 顶点的 butterfly
T_u	顶点 u 的 tip 值
H^u_k	包含顶点 u 且 k 值最大的 k-tip

给定无向二部图 $G=(U,V,E)$, 其中, U 和 V 是两个不相交的顶点集, E 表示图中边集合, 满足$\{e=(u,v)\in E|u\in U\wedge v\in V\}$. 任意顶点 $u\in U$ 的邻接点仅存在于 V 中, 表示为 N_u, 其中, $d_g(u)=|N_u|$为 u 的度. 由于同侧顶点之间

没有直接边相连, 可通过 butterfly 子图衡量同一集合中顶点间的紧密关系, 即 $d_{\bowtie}$对应于简单图中度的概念, 包含顶点 u 的 butterfly 数表示为 $d^u_{\triangleright\triangleleft}$. 对于同一 butterfly 中的两个顶点 u_1和 u_2, 使用 $\triangleright\triangleleft^{u_2}_{u_1}$ 来表示它们之间可通过 butterfly 子图相连, $|\triangleright\triangleleft^{u_2}_{u_1}|$可用于度量两个顶点 u_1和 u_2之间的紧密程度.

定义 1(butterfly 子图, butterfly). 给定二部图 $G=(U,V,E)$, 存在子图 $H=(U',V',E')$, 其中, $U'=\{u_1,u_2\}\subseteq U$, $V'=\{v_1,v_2\}\subseteq V$, $H=(U',V',E')$是 butterfly 子图当且仅当 u_1和 u_2是 v_1和 v_2的共同邻居(即 H 是一个(2,2)-二部团).

一般来说, 社区由一组密切相关的同质实体组成. k-core 是简单图中一种紧密子图结构, 其中任意顶点至少有 k 个同社区邻接点. 在二部图中, 同一侧的顶点属于同一类实体, 而各顶点仅维系位于异侧的邻接点, 因此无法通过顶点的度获取同侧顶点间的连接关系. 如果两个顶点在至少一个相同的 butterfly 中, 可将 u_1映射为 u_2的邻接点. 本文主要研究 U 侧顶点的内聚结构, 在此基础上, k-tip 社区的定义如下.

定义 2(k-tip). 给定一个二部图 $G=(U,V,E)$, 子图 $H=(U',V',E')$是一个 k-tip 当且仅当满足如下 3 个条件.

- 连通性. k-tip 中属于 U'的每对顶点都可以通过 1 个或多个 butterfly 直接或者间接相连.
- 紧密性. 每个顶点 $u\in U'$至少存在于 k 个 butterfly 中.
- 最大性. 不存在其他 k-tip 包含 H.

定义 3(tip 值). 给定一个二部图 $G=(U,V,E)$, 顶点 $u\in U$ 只能存在于 k-tip 而不能存在于任何 k+1-tip, 则 u 的 tip 值为 k, 记作 T_u, 相应的 k-tip 表示为 H^u_k.

根据以上对二部图的分析和相关定义, 本文主要研究以下两个问题.

问题 1(butterfly 计数). 给定一个二部图 $G=(U,V,E)$, 对于每个顶点 $u\in U$, 统计包含 u 的 butterfly 个数, 即 H^u_k.

问题 2(tip 分解). 给定一个二部图 $G=(U,V,E)$, 对于每个顶点 $u\in U$, 计算 u 的 tip 值.

如图 1 所示, 顶点 u_1, u_2, v_1, v_2形成一个 butterfly. 对于顶点 u_2, 有 5 个 butterfly 包含它, 包括 1 个 butterfly $\triangleright\triangleleft^{u_2}_{u_1}$、3 个 butterfly $\triangleright\triangleleft^{u_3}_{u_2}$、1 个 butterfly $\triangleright\triangleleft^{u_4}_{u_2}$. 根据所有顶点的 butterfly 计数结果, 通过迭代处理 butterfly 度最小的顶点, 直到图为空, 即可获得每个顶点的 tip 值. 同时, 可分别构建出 1-tip, 2-tip 和 3-tip 社区.

3 分布式 butterfly 计数

尽管 tip 分解类似于简单图中的 core 分解, 但由于二部图结构的特殊性, 不能将 core 分解算法直接应用于前者. 为了构造同侧每对顶点之间的关系, 需枚举一个二部图中所有 butterfly(即找到适用于所有 $u_1\in U$ 和 $u_2\in U$ 的 $\triangleright\triangleleft^{u_2}_{u_1}$). 本节将提出一种基于顶点中心计算模式的分布式 butterfly 计数算法(DBC). 此外, 由于原始二部图分布式存储在集群中, 不能直接将现有的单机 butterfly 计数算法应用到分布式系统中. 为应对现有分布式图计算系统无法适用于二部图的局限性, 本文引入一种灵活的基于中继的通信模式.

3.1 基于中继的通信模式

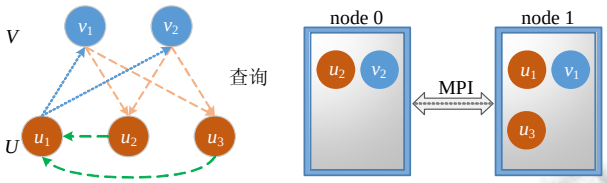
由于二部图结构的特殊性, 需要设计一种适合 2 跳邻居之间通信的消息传递方式. 给定一个二部图 $G=(U,V,E)$, 对于顶点 $u\in U$, 它只获得其属于 V 的邻居. 为了统计包含 u 的 butterfly(即 $\triangleright\triangleleft^{u'}_{u}|u'\in U$), 需要得到不同 2 跳邻居的共享邻居数.

在以顶点为中心的系统中, 顶点可通过接收来自邻接点的消息来激活. 基于这一思想, 以下将提出了一种用于分析分布式系统上二部图更灵活的通信模式.

策略 1(基于中继的通信模式). 首先, 人工激活 U 中的顶点, 并发送消息给 V 中的邻接点, 在下一个超步中这些邻接点将被激活; 随后, V 中被激活的顶点将通过消息发送激活 U 中已被人工激活顶点的二跳邻接点; 最后, U 中被激活的顶点互为二跳邻接点, 可直接相互发送消息.

如图 3 所示, 给定一个二部图 $G=(U,V,E)$, 其中, U 中包含 3 个顶点 u_1, u_2, u_3, V 中包含 2 个顶点 v_1和 v_2, 将这些顶点根据其 id 的哈希值分布式存储在集群中. 对于顶点 $u_1\in U$, 需要获取 U 中它的二跳邻接点和 V 中和

它的二跳邻接点之间的共享邻接点. u_1 首先被激活, 并向它的邻接点 v_1 和 v_2 发送消息; 然后, v_1 和 v_2 将 u_1 的 id 1 发送给 u_2, u_3(顶点 u_1 除外). u_2 和 u_3 将在下一个超步中接收到来自 v_1 和 v_2 的消息 1(u_1 的 id), 这意味着它们分别与 u_1 有两个共享邻接点; 之后, u_2 和 u_3 确定 u_1 是它们的二跳邻接点, 并分别向 u_1 发送它们的 id 值 2, 3 和共享邻接点的信息. 通过以上分析发现, 访问和返回二跳邻接点的信息共需 4 个超步.



(a) Sub bipartite graph (b) Distributed storage

图 3 基于中继的通信模式示例图

3.2 Butterfly计数算法

一个包含顶点 u_1 和 u_2 的 butterfly 可以看作是顶点间的一条抽象边, butterfly 的数量被定义为边的权重. 显然, 权重与 u_1 和 u_2 的共享邻居数量直接相关.

定理 1. 给定一个二部图 $G=(U,V,E)$, 对于两个顶点 $u_1\in U$ 和 $u_2\in U$, V 中有 n 个共享邻居, 那么同时包含 u_1 和 u_2 的 butterfly 数为 $n\times(n-1)/2$.

证明: 根据定义 1, butterfly 是一个(2,2)-二部团, 每侧包含两个顶点和其中所有 4 个可能的边. 对于两个顶点 u_1 和 u_2, 如果存在两个顶点 $v_1\in V$ 和 $v_2\in V$ 都与 u_1 和 u_2 相邻, 则这 4 个顶点可以构成一个 butterfly $\triangleright\triangleleft_{u_1,u_2,v_1,v_2}$. 因此, 任何两个共享邻居都可以与 u_1 和 u_2 形成 butterfly. 包含每对顶点的 butterfly 总数为

$$ButterflyCnt=C_n^2=\frac{2!}{(n-2)!\times 2!}=\frac{n\times(n-1)}{2},$$

其中, n 是共享邻居的数量. □

如图 1 所示, 根据基于中继的通信模式, 需要 4 个超步来获得各顶点与其二跳邻接点的共享邻接点数量. 顶点 u_1 和 u_2 有共享的邻接点 v_1 和 v_2, 存在 butterfly $\triangleright\triangleleft_{u_1,u_2,v_1,v_2}$ 包含它们. 对于顶点 u_2 和 u_3, 顶点 v_1, v_2 和 v_3 是它们的共享邻接点, 所以 u_1 和 u_2 同时参与 3 个 butterfly.

将单台机器上现有的 butterfly 计数算法应用于分布式系统成本太高. 这是因为它们大多需要对所有顶点进行重新排序和编号, 这不仅会引入大量额外的通信开销, 还会产生大量的顶点迁移开销. 为了提高分布式 butterfly 计数算法的效率, 本文设计了一种基于顶点优先级的消息剪枝策略来减少冗余通信.

策略 2(基于顶点优先级的消息剪枝策略). 给定一个 $G=(U,V,E)$, 每个顶点 $u\in U$ 只需要统计与 id 大于自身的二跳邻接点所组成的 butterfly 数量. 因此, V 中激活的顶点只需将人工激活的顶点 u 的 id 发送给 id 大于它的邻居.

如图 3 所示: 当激活 u_2 时, v_1 和 v_2 将在下一个超步中接收消息并被激活. 根据策略 2, v_1 和 v_2 只需将 u_2 的 id(2)发送给 u_3, 从而避免了 u_1 和 u_2 间 butterfly 重复计算.

基于策略 1、策略 2 和定理 1, 分布式 butterfly 计数算法的主要思想是寻找 U 中每对顶点的共享邻居. 为了提高资源利用率, 引入并行的思想来提高算法效率.

如算法 1 所示, 待分解的二部图已经根据每个顶点 id 的哈希值存储在分布式机器中, 而 $U'\subset U$ 表示存储在当前机器中的那些顶点. 在初始化阶段, *ActiveVector*(第 2 行)用于存储已激活的顶点. Butterfly 计数阶段, 每次迭代需要 4 个超步. 首先, 从 U'中选择 m 个顶点, 并将它们放入 *ActiveVector* 中, 其中, m 限制了每台机器上并行处理的初始化顶点的数量. 注意, 此处 m 用于实现可控的并行顶点激活策略. 很显然, m 越大, 算法并行度越高, 但是通信量会增加, 因此可能导致内存溢出. 这些人工激活的顶点向它们的邻居发送消息(超步 1,

第 5 行–第 10 行). 其次, 根据策略 2(超步 2, 第 12 行–第 16 行), V 中从 U 接收到消息的顶点被激活, 并将人工激活的顶点的 id 转发给它们的邻居. 接着, 根据定理 1 计算 butterfly 的数量, 并将结果发送给相应的人工激活的顶点(超步 3, 第 18 行–第 22 行). 最后, 在第 1 个超步中激活的这些顶点从它们的 2 跳邻居接收消息, 并获得 butterfly 计数结果(超步 4, 第 24 行–第 27 行). 经过这 4 个超步, U 中的部分顶点可确定它们所在的 butterfly 数量. 当 U 中的所有顶点都经过相同的运算后, butterfly 计数算法结束.

算法 1. Distributed Butterfly Counting.

输入: 二部图 $G=(U,V,E)$.

输出: 对于每个顶点 $u\in U$ 的 butterfly 计数结果.

```
1.  /* 所有机器上的初始化阶段 */
2.  ActiveVector←0
3.  /* 所有机器上的 butterfly 计数阶段 */
4.  repeat
5.      Superstep 1: 人工激活顶点
6.      从 U′中选择 m 个顶点进行并行处理, 并放入 ActiveVector
7.      U′=U′\ActiveVector
8.      for u∈ActiveVector do
9.          for v∈N_u do
10.             向 v 发送消息, 并在下一个超步中激活它
11.     通信同步障碍
12.     Superstep 2: 转发中继消息
13.     V 中的顶点接收来自 U 的消息, 并将其放入 ActiveVector
14.     for v∈ActiveVector do
15.         for u′∈N_v do
16.             如果 u′.id 大于人工激活的顶点的 id, 将它们的 id 发送给 u′
17.     全局栅栏同步
18.     Superstep 3: 进行 2 跳邻居上的 butterfly 计数
19.     U 中的顶点接收来自 V 的消息, 并将其放入 ActiveVector
20.     for u′∈ActiveVector do
21.         根据定理 1, 通过收到的人工激活的顶点的 id 计算 butterfly 数, 并将结果加到 d_⋈^u′ 上
22.         将结果发送到相应的人工激活的顶点
23.     全局栅栏同步
24.     Superstep 4: 计算 butterfly 计数结果
25.     U 中的顶点接收来自 U 的消息, 并将其放入 ActiveVector
26.     for u∈ActiveVector do
27.         将 butterfly 计数结果添加到 d_⋈^u
28. until 每台机器上的 U′不为空
29. return 每个顶点 u∈U 的 d_⋈^u 值
```

- 算法分析

对于每次迭代, 需要 4 个超步来获得包含人工激活顶点的 butterfly 数量. 而这一批中的 m 个顶点在第 1 个超步中被激活, 可在 4 个超步中得到 butterfly 计数结果. 因此, 分布式 butterfly 计数算法可以在 $4\times U'/m$ 超步中收敛. 对于通信成本, 可通过整个过程中传递的消息数量进行分析. 在第 1 个超步中, 人工激活的顶点需要向它们的所有邻居发送消息; 然后, V 中的顶点按照策略 2 发送消息. 因此, 前 2 个超步的消息总数为

$$\sum_{u\in U}\left(N_u+\sum_{v\in N_u}\left(\sum_{u'\in N_v}\mathrm{sgn}(u'.id>u.id)\right)\right),$$

其中, u 是第 1 个超步中人工激活的顶点, u'是 u 的 2 跳邻居. 此外, 在第 3 个超步中, 2 跳邻居将 butterfly 计数的结果发送到相应的人工激活的顶点, 该结果的值是 id 大于 $u.id$ 的 u'的数量.

4 分布式 tip 分解

第 3 节所提出的 DBC 算法执行完成后, 每个顶点 $u\in U$ 已知其 butterfly 度. 类似简单图中顶点的度数, $d^u_{\triangleright\triangleleft}$只能用于粗略分析二部图的结构, 因此需要得到每个顶点的 tip 值来挖掘二部图中更细粒度的结构特征. 本节提出一种基于层次剥离的分布式 tip 分解算法(DTD)来获取 U 中顶点的 tip 值.

策略 1 可解决同侧顶点间无法直接发送消息的问题, 基于此, 可研究分布式 tip 分解算法. 为了在分布式 tip 分解算法中引入并行性, 在每轮迭代过程中, 批量剥离 butterfly 数量最少的顶点, 而剩余未被剥离的顶点更新各自的 tip 值.

定理 2. 给定一个二部图 $G=(U,V,E)$和 U 中最小的 $d_{\bowtie}$值 $d_{\bowtie\min}$, 则 U 中每个顶点的 tip 值都不小于 $d_{\bowtie\min}$.

证明: 给定一个二部图 $G=(U,V,E)$, 且 U 中 $d_{\bowtie}$的最小值为 $d_{\bowtie\min}$. 根据定义 2, G 是一个 $d_{\bowtie\min}$-tip, 并且每个顶点 $u\in U$ 至少存在于 $d_{\bowtie\min}$-tip 中. 结合定义 3, 可推各顶点 tip 值至少为 $d_{\bowtie\min}$. □

顶点被剥离后即可确定其 tip 值, 但是在原始图数据中, 该顶点还存在于其邻接点的邻接点列表中, 而在以顶点为中心的模式下, V中的顶点无法获知 U中剥离顶点情况. 如果后续的迭代中 V的顶点继续向所有邻居转发消息, 将会产生大量冗余消息.

策略 3(消息有效性剪枝策略). 给定一个二部图 $G=(U,V,E)$, V 中的顶点只在 U 中有邻居, 对于一个顶点 $v\in V$接收到被删减邻接点 $u\in U$ 的消息后, 将其置为不被激活状态, 即 0. 当 v 转发消息时, 不需要考虑状态为 0 的邻接点.

分布式 tip 分解算法(distributed tip decomposition(DTD))的主要思想是: 根据 $d_{\bowtie\min}$ 值, 批量地剥离顶点. 为了解决分布式存储下无法对顶点进行排序的问题, 以下将构建 butterfly 树 *BFTree* 来分层存储顶点 id. *BFTree* 由多组键值对组成, 其中, "键"表示顶点的 $d_{\bowtie}$值, "值"是一个存储顶点 id 的容器. 伪代码如算法 2 所示.

算法 2. Distributed Tip Decomposition.

输入: 二部图 $G=(U,V,E)$, 每个顶点 $u\in U$ 的 $d^u_{\triangleright\triangleleft}$ 值.

输出: 对于每个顶点 $u\in U$ 的 tip 分解结果.

```
1.  /* 所有机器上的初始化阶段 */
2.  ActiveVector←0
3.  GlobalMinKey←0
4.  根据 d^u_{▷◁} | u∈U 构造一个 butterfly 树 BFTree.
5.  for each u in U do
6.      if BFTree.find( d^u_{▷◁} )=BFTree.end(·) then
7.          BFTree[ d^u_{▷◁} ]←{u.id}
8.      else
9.          BFTree[ d^u_{▷◁} ].push_back(u.id)
10.
11. /* 所有机器上的 tip 分解阶段 */
12. repeat
13.     Superstep 1: 人工激活顶点
14.     GlobalMinKey←Get the minimal key in BFTree
15.     ActiveVector←BFTree[GlobalMinKey]
```

16. *BFTree**BFTree*[*GlobalMinKey*]
17. **for** $u \in ActiveVector$ **do**
18. **for** $v \in N_u$ **do**
19. 向 v 发送消息, 并在下一个超步中激活它
20. $u.peeled \leftarrow 1$
21. 全局栅栏同步
22. Superstep 2: 转发中继消息
23. V 中的顶点接收来自 U 的消息, 并将其放入 *ActiveVector*
24. **for** $v \in ActiveVector$ **do**
25. **for** $u' \in N_v$ **do**
26. **if** $u'.peeled=0$ **then**
27. 将人工激活的顶点的 id 发送到 u'
28. 全局栅栏同步
29. Superstep 3: 更新 2 跳邻居上的 butterfly 数
30. U 中的顶点接收来自 V 的消息, 并将其放入 *ActiveVector*
31. **for** $u' \in ActiveVector$ **do**
32. $preBFcnt \leftarrow d^{u'}_{\triangleright\triangleleft}$
33. 根据定理 1 接收到的人工激活的顶点的 id 减去 $d^{u'}_{\triangleright\triangleleft}$ 的结果计算 butterfly.
34. **if** $d^{u'}_{\triangleright\triangleleft} < GlobalMinKey$ **then**
35. $d^{u'}_{\triangleright\triangleleft} \leftarrow GlobalMinKey$
36. *BFTreeUpdate*($preBFcnt, d^{u'}_{\triangleright\triangleleft}$)
37. **until** 每台机器上的 *BFTree* 不是空的
38. **return** 每个顶点 $u \in U$ 的 $d^{u}_{\triangleright\triangleleft}$ 值

- 如算法 2 所示, 首先, 根据每个顶点的 butterfly 度生成 *BFTree*(第 5 行–第 10 行).
- 在第 1 个超步中, 从 *BFTree* 的键中获得全局最小的 $d_{\bowtie}$值, 即 *GlobalMinKey*(第 14 行). 具有最小 $d_{\bowtie}$ 值的顶点 id 存储在 *BFTree*[*GlobalMinKey*]中, 将其删减并向它们的邻接点发送消息(第 15 行–第 19 行). 此外, 这些人工激活的顶点的状态被设置为被剥离(第 20 行).
- 在第 2 个超步中, V 中的顶点接收消息, 并将这些消息的源 id 转发给尚未被剥离的邻居(第 23 行–第 28 行).
- 在第 3 个超步中, 第 1 个超步中这些人工激活的顶点的 2 跳邻居根据定理 1(第 31 行–第 34 行)更新它们的 $d_{\bowtie}$. 根据定理 2, 仍未剥离的顶点的 tip 值至少为 *GlobalMinKey*. 因此, 如果一个未被剥离的顶点在算法 2 第 34 行之后的 tip 值小于 *GlobalMinKey*, 将其设置为 *GlobalMinKey*(第 35 行、第 36 行).

需要注意的是, 顶点的 $d_{\bowtie}$值的更新将会影响 *BFTree* 的结构.

函数 *BFTreeUpdate* 旨在当顶点的 $d_{\bowtie}$值改变时更新 *BFTree*(第 34 行). 具体更新方法类似于 *BFTree* 的构建过程. 在前 3 个超步之后, 第 1 个超步中人工激活的顶点已经确认了它们的 tip 值. DTD 算法开始另一次迭代, 直到 *BFTree* 为空(第 12 行–第 38 行).

函数 *BFTreeUpdate*(*preBFcnt*, *curBFcnt*).

输入: 顶点 *u*, *BFTree*, *preBFcnt*, *curBFcnt*.

输出: updated *BFTree*.

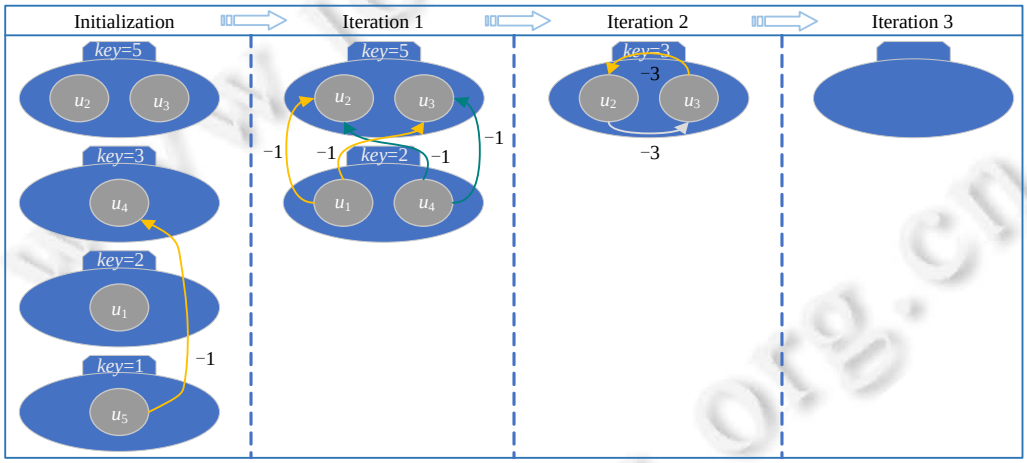
1. 从 *BFTree*[*preBFcnt*]中剥离 *u.id*
2. **if** *BFTree*[*preBFcnt*]为空 **then**

3. 从 *BFTree* 中剥离 *BFTree*[*preBFcnt*]
4. **if** *BFTree*[*curBFcnt*]!=*BFTree.end*(·) **then**
5. *BFTree*[*curBFcnt*].*push_back*(*u.id*)
6. **else**
7. *BFTree*[*curBFcnt*]←{*u.id*}
8. **return** *BFTree*

图 4 展示了图 1(a)中二部图的 tip 分解过程. 如图 4(a)所示, 顶点 u_1–u_5 的 $d_{\bowtie}$值分别为 2, 5, 5, 3 和 1. 相应的 *BFTree* 由这些顶点及其 $d_{\bowtie}$生成, 其结构如图 4(b)所示. 在第 1 次迭代中, u_5 具有最小的 $d_{\bowtie}$值, 并从原始图中被剥离. 因为 u_4 和 u_5 共同参与了一个 butterfly, 使得 $d^{u_4}_{\triangleright\triangleleft}$ 从 3 减少到 2. *BFTree* 的结构也对应更新. 在第 2 次迭代中, u_1 和 u_4 被放入 *ActiveVector* 中, 因为它们都具有当前最小的 $d_{\bowtie}$值 2. 当 u_1 和 u_4 从 U 中剥离时, U 中剩余的顶点为 u_2 和 u_3, 它们将在第 3 次迭代中被剥离. 在这些迭代中, 顶点在被剥离时的 $d_{\bowtie}$值即为 tip 值. 分解结果如图 1(e)所示.

Vertex	Initialization	Iteration-1	Iteration-2	Iteration-3
u_1	2	2	_	_
u_2	5	5	3	_
u_3	5	5	3	_
u_4	3	2	_	_
u_5	1	_	_	_

(a) 迭代过程中顶点的 $d_{\bowtie}$值



(b) 迭代过程中的 *BFTree*

图 4 二部图上的 Tip 分解示例图

- 算法分析

给定一个二部图 $G=(U,V,E)$, 算法 2 最多需要 $3\times|U|$个超步来得到 U 中所有顶点的 tip 值. 假设在每次迭代中只有一个顶点具有最小的 $d_{\bowtie}$值, 根据算法 2, 需要消耗 3 个超步来剥离顶点 u, 并更新 u 的 2 跳邻接点的 $d_{\bowtie}$值. 在第 1 个超步中, 激活的顶点 u 向其邻接点 N_u 发送消息. 收到消息的 $v\in N_u$ 在下一个超步中将源 id(即 *u.id*)转发给 $u'\in U$(即 u 的 2 跳邻接点). 之后, 那些 2 跳邻居根据定理 1 更新各自的 tip 值. 综上所述, 每个顶点需要 3 个超步来获得 tip 值, 并更新其 2 跳邻接点的 $d_{\bowtie}$值, 则总的超步消耗为 $3\times|U|$.

5 实验结果与分析

5.1 实验设置

- 实验环境

本文实验代码采用 C++编写, 基于 MPI(消息传递接口)实现不同计算节点间消息通信, 再由 mpicxx 编译为可执行文件. 本文所设计的基于中继的通信模式被嵌入到分布式图计算系统中 并部署于国家超级计算长沙中心(TH-I)上, TH-I 配备 160 Gb/s 高速互联系统, 单个计算节点包含 2 块 Intel(R) Xeon(R) CPU, 拥有 48 GB 主存. TH-I 底层 MPI 为自主实现, 基于 Intel 编译器进行编译, 本文中算法的实验代码契合此环境. 本文采用 10 个计算节点构建分布式环境.

- 实验设置

本文为首次在分布式环境下研究 butterfly 计数和 tip 分解问题. 对于第 3 节所述的 DBC 算法, 基准为设定并行度为 1, 即算法 1 第 6 行中的 m 设置为 1, 对比实验为 m={10,30,50,100,200,500}, 评估指标为总执行时间(T)和总超步数(S). 对于第 5 节所述的 DTD 算法, 基准为不使用消息有效性剪枝策略(策略 3), 对比算法为使用策略 3, 评估指标为总的 tip 分解时间(T)和通信量(C).

- 数据集

实验使用的真实二部图数据集的基本情况见表 2, 数据集均可在 Konnect 官网下载(数据集下载网址: http://konect.cc/). 所有数据集采用 hash 的方式进行分布式存储, 即根据顶点编号的 hash 值确定该顶点所在机器的编号. 基于 hash 的图划分技术是一种普遍应用于分布式图计算系统[29]上的方法, 具有划分效率高、保障存储在各计算节点上的顶点数量均衡、在消息传递过程中快速确定目的顶点所在计算节点的优点.

表 2 真实数据集

数据集	$\|U\|$	$\|V\|$	$\|E\|$
Baidu	901 758	916 634	8 609 972
Dblp	172 072	53 400	293 673
Dblpau	1 953 085	5 624 219	12 282 059
Frwiki	757 621	8 829 774	52 950 008
Twitter	175 214	530 418	1 890 644

5.2 实验结果分析

根据第 5.1 节实验设置所述, 本节就 butterfly 计数和 tip 分解两种算法分别从执行时间、超步数和通信量等指标来对算法的实验结果进行分析.

5.2.1 Butterfly 计数算法性能评估

Butterfly 计数算法(算法 1)旨在计算所有顶点(本实验只考虑 U 侧顶点)所在的 butterfly 数量, 根据策略 1(基于中继的通信模式)可实现分布式环境下二部图顶点间通信. 但是如果在第一超步中同时激活所有顶点, 其通信量过大将导致内存溢出问题, 因此在实验过程中需要手动调整算法并行度(算法 1 第 6 行). 本组实验主要目的在于验证不同并行度下, 分布式 butterfly 计数算法的性能变化趋势.

- 执行时间

算法的执行时间是用于评估算法性能最直接、最有效的指标. 根据第 3 节和第 4 节两种算法的执行流程, 实验统计了 10 个计算节点各自在人工激活不同数量的顶点(分别是 1, 10, 30, 50, 100, 200 和 500)时 DBC 算法在不同数据集上的执行时间. 此时, DBC 算法默认已使用策略 1 和策略 2, 统计结果见表 3. 由结果可见, 当顶点数由 1 变化到 50 时, DBC 算法的耗时明显减少; 之后, 随着每个计算节点所激活的顶点数量不断增加, DBC 算法的性能趋于稳定. 由此发现, 尽管算法并行度提高了, 但是引入了单超步通信量和计算量的增加, 使得整体效率提升幅度不再增加.

表 3 DBC 算法在不同数据集上的执行时间

数据集	并行度 m						
	1	10	30	50	100	200	500
Baidu	1 130	684	535	412	411	365	384
Dblp	67	28	21	17	16	16	16
Dblpau	206	28	14	10	8	8	7
Frwiki	1 128	994	911	724	787	791	719
Twitter	638	299	226	179	166	154	160

- 超步数

在分布式系统中, 通信开销为总开销的重要组成部分. 本文所部署系统采用 BSP 计算模型, 因此可用超步数衡量通信开销. 算法 1 的并行度由其第 6 行参数 m 控制, 不同并行度下各数据集执行超步数见表 4.所有数据集随着并行度的增加, 超步消耗都得到有效的降低. 由于本文实验部署在超算中心高性能计算平台上, 底层配备高速通信网络, 超步数优化带来的时间减少不甚明显. 可以预见, 在普通分布式集群上, 超步数的大幅度减少, 可带来更高的性能提升.

表 4 不同并行度下超步数消耗

数据集	并行度 m						
	1	10	30	50	100	200	500
Baidu	361 116	40 148	13 384	7 224	3 612	1 808	724
Dblp	68 836	7 652	2 552	1 380	692	348	140
Dblpau	781 236	86 804	28 936	15 628	7 816	3 908	1 564
Frwiki	303 052	33 676	11 228	6 064	3 032	1 516	608
Twitter	70 420	7 900	2 636	1 412	708	356	144

5.2.2 Tip 分解算法性能评估

Tip 分解算法用来计算所有顶点的 tip 值, 表示该顶点所能存在的拥有最大 k 值的 k-tip 社区, 该项指标可应用于挖掘二部图中的紧密社区. 本组实验主要计算 U 侧顶点 tip 值, 但由于同侧顶点无法直接访问其二跳邻接点, 也无法获取其二跳邻接点状态(是否被剥离), 需要通过异侧顶点(V 侧顶点)进行消息转发. 同时, 本文所设计的消息有效性剪枝策略也由中继顶点执行. 本组实验主要验证所设计的消息剪枝策略的有效性, 并从通信量角度量化分析性能提升的原因.

- 执行时间

DBC 算法使数据集中每个顶点 $u\in U$ 可以得到 butterfly 度后, 本实验重点统计并对比了未使用策略 3 的 DTD 算法和使用了策略 3 的 DTD 算法(表示为 DTD+)在不同数据集上的执行时间. 由表 5 的实验结果可以看出: 消息剪枝策略(策略 3)有效地减少了总时间消耗, 最高提升比例为 51.9%. 其原因在于: 消息量的减少一方面降低了通信量, 另一方面减少了下一超步被激活的顶点数, 进而减少了大量冗余的计算量. 在本算法中, 已经确定 tip 值的顶点无须再接收消息来更新自身 tip 值, 如果不使用策略 3, 则已经确定 tip 值的顶点可能会被多次激活, 从而引入大量冗余计算.

表 5 Tip 分解算法时间消耗

数据集	算法		
	DTD (s)	DTD+ (s)	Rate (%)
Baidu	960	462	51.9
Dblpau	89	77	13.5
Frwiki	3 299	2 980	9.7
Twitter	2 439	2 190	10.2

- 通信量

上述实验分析了策略 3 的有效性来源于减少冗余通信进而减少冗余计算, 本组实验通过具体通信量量化分析策略 3 的有效性. 由表 6 的通信量统计结果可看出: DTD+通信量明显少于 DTD, 最高通信量降低比例约 57.8%. 根据 BSP 计算模式分析, 通信量决定下一轮顶点激活数, 进而决定各计算节点的计算负载, 通信量的

降低可以有效地提升分布式算法性能.

表 6 Tip 分解算法通信量分析

数据集	算法		
	DTD (条数)	DTD+ (条数)	Rate (%)
Baidu	285 815 206	126 092 382	55.9
Dblpau	4 903 833	2 069 477	57.8
Frwiki	653 528 468	534 658 219	18.2
Twitter	1 297 868 458	1 151 330 554	11.3

6 总结与展望

本文针对现有 butterfly 计数算法和 tip 分解算法在单机上存在存储瓶颈问题以及在分布式环境下存在大量通信开销的问题, 提出了一种新的基于中继的通信模式, 并在此基础上提出一种更高效的分布式 butterfly 计数算法, 并通过引入基于顶点优先级的消息剪枝策略, 显著减少冗余通信和额外的开销. 随后提出了一种分布式 tip 分解算法, 通过基于有效性的消息剪枝策略, 提高了算法效率. 实验结果表明, 本文所提算法能够有效地解决大规模二部图上的 tip 分解问题. 动态变化已成为大规模二部图数据的重要特性, 为此, 在未来的工作中将进一步研究大规模动态二部图的分布式 tip 值增量式维护算法.

References:

[1] Dhillon IS. Co-clustering documents and words using bipartite spectral graph partitioning. In: Proc. of the 7th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2001. 269−274.

[2] Wang K, Lin X, Qin L, *et al*. Efficient bitruss decomposition for large-scale bipartite graphs. In: Proc. of the 2020 IEEE 36th Int'l Conf. on Data Engineering (ICDE). IEEE, 2020. 661−672.

[3] Gibson D, Kumar R, Tomkins A. Discovering large dense subgraphs in massive graphs. In: Proc. of the 31st Int'l Conf. on Very Large Data Bases. 2005. 721−732.

[4] Seidman SB. Network structure and minimum degree. Social Networks, 1983, 5(3): 269−287.

[5] Malliaros FD, Giatsidis C, Papadopoulos AN, *et al*. The core decomposition of networks: Theory, algorithms and applications. The VLDB Journal, 2020, 29(1): 61−92.

[6] Cohen J. Trusses: Cohesive subgraphs for social network analysis. National Security Agency Technical Report, 2008.

[7] Newman MEJ. Scientific collaboration networks. I. Network construction and fundamental results. Physical Review E, 2001, 64(1): No.016131. [doi: 10.1103/PhysRevE.64.016131]

[8] Sarıyüce AE, Pinar A. Peeling bipartite networks for dense subgraph discovery. In: Proc. of the 11th ACM Int'l Conf. on Web Search and Data Mining. 2018. 504−512.

[9] Shi J, Shun J. Parallel algorithms for butterfly computations. In: Proc. of the Symp. on Algorithmic Principles of Computer Systems. Society for Industrial and Applied Mathematics, 2020. 16−30.

[10] Wang K, Lin X, Qin L, *et al*. Vertex priority-based butterfly counting for large-scale bipartite networks. Proc. of the VLDB Endowment, 2019, 12(10): 1139−1152.

[11] Sanei-Mehri SV, Sariyuce AE, Tirthapura S. Butterfly counting in bipartite networks. In: Proc. of the 24th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining. 2018. 2150−2159.

[12] Lakhotia K, Kannan R, Prasanna V, *et al*. RECEIPT: REfine CoarsE-grained independent tasks for parallel tip decomposition of bipartite graphs. arXiv preprint arXiv:2010.08695, 2020.

[13] Malewicz G, Austern MH, Bik AJC, *et al*. Pregel: A system for large-scale graph processing. In: Proc. of the 2010 ACM SIGMOD Int'l Conf. on Management of Data. New York: Association for Computing Machinery, 2010. 135−146.

[14] Sakr S, Orakzai FM, Abdelaziz I, *et al*. Large-scale Graph Processing using Apache Giraph. Springer, 2016.

[15] Gonzalez JE, Xin RS, Dave A, *et al*. GraphX: Graph processing in a distributed dataflow framework. In: Proc. of the 11th USENIX Symp. on Operating Systems Design and Implementation. 2014. 599−613.

[16] Yan D, Cheng J, Lu Y, *et al*. Effective techniques for message reduction and load balancing in distributed graph computation. In: Proc. of the 24th Int'l Conf. on World Wide Web. Florence, 2015. 1307−1317.

[17] Yu G, Gu Y, Bao YB, Wang ZG. Large scale graph data processing on cloud computing environments. Chinese Journal of Computers, 2011, 34(10): 1753−1767 (in Chinese with English abstract).

[18] Lu L, Hua B. A platform-and-workload aware online graph partitioning algorithm. Chinese Journal of Computers, 2020, 43(7): 1230−1245 (in Chinese with English abstract).

[19] Zhang CB, Li Y, Jia T. Survey of state-of-the-art fault tolerance for distributed graph processing jobs. Ruan Jian Xue Bao/Journal of Software, 2021, 32(7): 2078−2102 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6269.htm [doi: 10.13328/j.cnki.jos.006269]

[20] Engelhardt N, So HKH. Gravf: A vertex-centric distributed graph processing framework on FPGAS. In: Proc. of the 2016 26th Int'l Conf. on Field Programmable Logic and Applications (FPL). IEEE, 2016. 1−4.

[21] Weng T, Zhou X, Li K, *et al*. Efficient distributed approaches to core maintenance on large dynamic graphs. IEEE Trans. on Parallel and Distributed Systems, 2021, 33(1): 129−143.

[22] Luo W, Zhou X, Yang J, *et al*. Efficient approaches to top-*r* influential community search. IEEE Internet of Things Journal, 2020, 8(16): 12650−12657.

[23] Ma C, Cheng R, Lakshmanan LVS, *et al*. LINC: A motif counting algorithm for uncertain graphs. Proc. of the VLDB Endowment, 2019, 13(2): 155−168.

[24] Fang Y, Cheng R, Luo S, *et al*. Effective community search for large attributed graphs. Proc. of the VLDB Endowment, 2016, 9(12): 1233−1244.

[25] Ma C, Fang Y, Cheng R, *et al*. Efficient algorithms for densest subgraph discovery on large directed graphs. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. 2020. 1051−1066.

[26] Chen R, Shi JX, Chen HB, *et al*. Bipartite-oriented distributed graph partitioning for big learning. Journal of Computer Science and Technology, 2015, 30(1): 20−29.

[27] Liu B, Yuan L, Lin X, *et al*. Efficient (α,β)-core computation in bipartite graphs. The VLDB Journal, 2020, 29(5): 1075−1099.

[28] Chiba N, Nishizeki T. Arboricity and subgraph listing algorithms. SIAM Journal on Computing, 1985, 14(1): 210−223.

[29] Yan D, Cheng J, Özsu MT, *et al*. A general-purpose query-centric framework for querying big graphs. Proc. of the VLDB Endowment, 2016, 9(7): 564−575.

附中文参考文献:

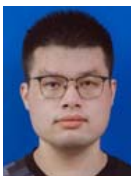
[17] 于戈, 谷峪, 鲍玉斌, 王志刚. 云计算环境下的大规模图数据处理技术. 计算机学报, 2011, 34(10): 1753−1767.

[18] 陆李, 华蓓. 平台和负载特征感知的在线图分割算法. 计算机学报, 2020, 43(7): 1230−1245.

[19] 张程博, 李影, 贾统. 面向分布式图计算作业的容错技术研究综述. 软件学报, 2021, 32(7): 2078−2102. http://www.jos.org.cn/1000-9825/6269.htm [doi: 10.13328/j.cnki.jos.006269]

周旭(1983－), 女, 博士, 副教授, 主要研究领域为并行计算, 图数据管理, 图计算.

李博仁(1998－), 男, 硕士生, 主要研究领域为图计算, 社区搜索.



翁同峰(1992－), 男, 博士生, CCF 学生会员, 主要研究领域为分布式图计算.

张吉(1977－), 男, 教授, 主要研究领域为数据挖掘, 图数据管理.

杨志邦(1984－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为并行计算, 图数据管理.

李肯立(1971－), 男, 博士, 教授, CCF 会士, 主要研究领域为并行分布式处理, 超级计算与云计算, 面向大数据和人工智能的高效能计算.