软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(4):1819−1840 [doi: 10.13328/j.cnki.jos.006842]

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

基于顶点组重分配的动态增量图划分算法*

李 贺[1], 刘延娜[1], 杨舒琪[1], 黄健斌[1], 乔少杰[2]

[1](西安电子科技大学 计算机科学与技术学院, 陕西 西安 710126)

[2](成都信息工程大学 软件工程学院, 四川 成都 610103)

通信作者: 李贺, E-mail: heli@xidian.edu.cn

摘 要: 图划分是分布式图计算中的一项基础工作, 其作用是将大规模图进行划分并分配到集群中的不同机器上. 图划分的质量对分布式图计算的性能有很大的影响, 其目标是降低负载平衡和最小化边割. 如今, 现实中的图数据通常呈动态增长态势, 这就需要一种能够处理动态增量图的划分方法, 在图数据动态增长的过程中确保划分的质量不受影响. 目前虽然有一些动态图划分算法被提出, 但它们不能同时专注于实时处理动态变化和获得高质量的划分结果. 提出基于顶点组重分配的动态增量图划分算法 (ED-IDGP) 来解决大规模动态增量图的划分问题. 在 ED-IDGP 算法中, 设计实时处理 4 种不同单元更新类型的动态处理器, 并在每次处理完单元更新后通过在分区发生动态变化的附近执行局部优化器进一步提高图划分的质量. 在 ED-IDGP 的局部优化器中, 利用基于改进标签传播算法的顶点组搜索策略搜索顶点组, 并利用提出的顶点组移动增益公式衡量最有益的顶点组, 将该顶点组移动到目标分区中做优化. 在真实数据集上从不同的角度和度量指标评估了 ED-IDGP 算法的性能和效率.

关键词: 图划分; 局部优化; 动态增量图划分算法

中图法分类号: TP311

中文引用格式: 李贺, 刘延娜, 杨舒琪, 黄健斌, 乔少杰. 基于顶点组重分配的动态增量图划分算法. 软件学报, 2024, 35(4): 1819–1840. http://www.jos.org.cn/1000-9825/6842.htm

英文引用格式: Li H, Liu YN, Yang SQ, Huang JB, Qiao SJ. Dynamic Incremental Graph Partitioning Algorithm Based on Vertex Group Redistribution. Ruan Jian Xue Bao/Journal of Software, 2024, 35(4): 1819–1840 (in Chinese). http://www.jos.org.cn/1000-9825/6842.htm

Dynamic Incremental Graph Partitioning Algorithm Based on Vertex Group Redistribution

LI He[1], LIU Yan-Na[1], YANG Shu-Qi[1], HUANG Jian-Bin[1], QIAO Shao-Jie[2]

[1](School of Computer Science and Technology, Xidian University, Xi'an 710126, China)

[2](School of Software Engineering, Chengdu University of Information Technology, Chengdu 610103, China)

Abstract: Graph partitioning is a basic task for distributed graph computing. It is used to divide a large-scale graph into different parts and allocate them to different machines in a cluster. The quality of graph partitioning has a great impact on the performance of distributed graph computing, and graph partitioning aims to minimize edge cuts and load balance. Nowadays, the graph data usually grow dynamically, which needs a partitioning method to process dynamic incremental graphs, so as to ensure the quality of graph partitioning. Although some dynamic graph partitioning algorithms have been presented recently, they cannot process real-time dynamic changes and obtain high-quality graph partitioning results simultaneously. In this study, a dynamic incremental graph partitioning algorithm based on vertex group redistribution (ED-IDGP) is proposed to solve the problem of large-scale dynamic incremental graph partitioning. In ED-IDGP, a dynamic processor is designed to process four different unit update types in real time, and the graph partitioning quality is further improved by executing a local optimizer near the dynamic change in the partition after each unit update. In the local optimizer of ED-IDGP, a vertex group search strategy based on the improved label propagation algorithm is used to search for the vertex group, and a vertex

* 基金项目: 国家自然科学基金 (61602354, 61876138)
收稿时间: 2022-04-23; 修改时间: 2022-08-29; 采用时间: 2022-11-30; jos 在线出版时间: 2023-07-28
CNKI 网络首发时间: 2023-07-31

group movement gain formula is proposed to measure the most beneficial vertex group and move it to the target partition for optimization. This study evaluates the performance and efficiency of the ED-IDGP algorithm from different perspectives and metrics on real datasets.

Key words: graph partitioning; local optimization; dynamic incremental graph partitioning algorithm

近年来, 随着互联网信息技术的迅猛发展, 各行各业每时每刻都在产生着各种各样的数据. 图作为一种特殊的数据结构不但可以存储数据个体本身, 还可以存储数据个体之间复杂的关联关系. 在现实世界中, 许多应用都可以建模为图, 例如有最短路径查找、关系模式挖掘、Web 网页排序等传统的图应用, 还有如社会网络分析、交通网络分析、基于机器学习和深度学习的知识图挖掘等新兴的图计算应用. 早期的图数据规模较小, 只需要在单个机器中进行存储和处理. 但是现如今, 我们所面临的图数据通常规模庞大、结构复杂, 而且呈动态增长态势. 例如, 微信的日活跃用户从 2015 年的 5.7 亿增长到 2020 年的 10.9 亿. Facebook 的全球日活跃用户从 2011 年的 5 亿增长到 2021 年的 18.8 亿. 万维网的中国网页数量从 2017 年的 2 604 亿增长到 2019 年的 2 978 亿. Google 的知识图谱数据库从 2012 年的 5 亿个数据对象增长到现如今的 570 亿个数据对象. 随着图数据规模的日益增长, 在一台机器上无法存储和处理大规模图数据. 因此, 大规模图数据需要在分布式环境中执行, 为了满足这一需求, 许多分布式图处理系统应运而生, 如 Pregel[1], Giraph[2], GraphLab[3], Powerlyra[4]和 RGraph[5]等. 此外, 还有一些针对图存储和图查询的分布式图数据库系统, 如 Neo4j[6], Horton[7]和 GraphBase[8]等.

在分布式图数据库系统或者分布式图处理系统中, 其中一个关键问题是如何将一个大规模图数据划分到一个集群中的不同机器上, 以为多个客户端提供低延迟的在线查询服务或者为图处理任务提供高性能的并行计算. 因此, 图划分是大规模分布式图处理的首要工作, 对图应用的存储、查询、处理和挖掘起基础支撑作用. 在分布式系统中, 图应用的执行时间取决于一个集群中运行最慢的机器, 这就需要集群中的不同机器的工作负载应尽可能相等. 此外, 图处理任务需要沿着图的拓扑结构进行, 这使得一个机器中的某些顶点需要与其他机器中的邻居顶点通过传递消息进行通信. 例如图查询中需要从某个顶点开始进行遍历. 但是由于网络带宽、价格以及延迟等因素, 集群中不同机器之间进行通信的成本远远要比机器内部的通信成本高, 这就需要不同机器间的通信成本应尽可能低. 而图划分作为图处理的预处理步骤, 它的质量对分布式图处理任务的性能有很大的影响. 因此, 为了提高图计算的性能, 图划分的目标是最小化不同分区间的通信成本, 同时保证各个分区的负载平衡.

图是由顶点和边构成, 划分一个图的基本方式主要有顶点划分 (基于边割的图划分) 和边划分 (基于点割的图划分), 如图 1 所示. 顶点划分是将图中的顶点划分到不同的分区中, 若一条边的两个端点被划分到两个不同的分区, 那么该边被切割. 基于边割的图划分的目标是最小化不同分区之间边被切割的数量, 并且使每个分区中顶点的数量尽可能相等. 而边划分是将图中的边划分到不同的分区中, 若一个顶点的不同邻边被划分到两个或者多个不同的分区中, 那么该顶点被切割. 基于点割的图划分的目标是最小化不同分区之间点被切割的数量, 并且使每个分区中边的数量尽可能相等. 已有的大量工作表明基于边割的图划分有着更好的局部性, 而基于点割的图划分对于具有超度顶点的图会使得分区负载更加平衡[4]. 本文关注的是基于边割的图划分问题.

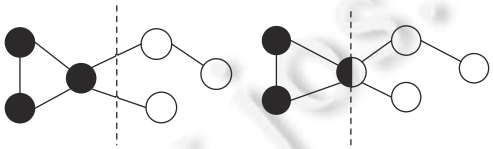


图 1 顶点划分与边划分

图划分问题已经被证明是一个 NP 难问题[9]. 研究者提出了大量的启发式图划分算法, 从全局方法[10−12]到多级方法[13−15]再到局部方法[16,17]. 但这些方法都是用来划分静态图的. 随着图数据的规模不断增长, 真实世界中的图表现出动态性, 即伴随着顶点或边的插入或删除. 如何对动态图进行划分已成为目前图划分领域研究的热点问题. 现有的可以对动态图进行划分的方法主要分为利用已有的静态图划分算法从头开始划分动态图、流式图划分方法[18−22]、动态图重划分方法[23−26]以及增量图划分方法[27−30], 其中增量图划分方法根据处理更新的规模分为单元

素增量图划分方法和批处理增量图划分方法. 当图数据实时发生动态变化时, 对于执行中的分布式图计算任务, 需要一种方法实时地对发生动态变化的图数据做处理以确保任务准确进行, 同时还需要保证图划分的质量以提升分布式图计算的性能. 表 1 给出了不同种类的动态图划分方法的特点, 从中可知利用已有的静态图划分方法从头开始重划分动态图会导致高的划分时间和成本. 大部分流式图划分方法没有局部优化, 一般会导致划分质量的下降, 并且流式图划分方法不能处理顶点或边的删除. 动态图重划分方法在已有初始图的分区上, 假设图发生动态变化已经导致分区发生变化, 通过移动顶点进一步做优化. 该方法没有专门的机制处理动态变化部分以满足正在执行的分布式图计算任务, 因此不适合这种应用场景. 而增量图划分方法能够专注处理动态部分. 但其中的批处理增量图划分方法不是针对实时性提出的, 并且划分质量不高. 基于以上分析, 本文研究的是单元素增量图划分方法, 它能够实时地专注处理动态图数据以确保执行中的分布式图计算任务准确进行, 以及通过局部优化技术以轻量级的性能开销能够获得高质量的划分结果, 这具有重大的理论和现实意义.

表 1 动态图划分方法特点对比

类型	专注处理动态部分	实时性	轻量级	局部优化
从头开始重划分整个图	×	×	×	×
流式图划分方法	√	√	√	×
动态图重划分方法	×	×	√	√
批处理增量图划分方法	√	×	√	×
单元素增量图划分方法	√	√	√	√

本文第 1 节介绍图划分的相关方法和研究现状. 第 2 节介绍问题定义和问题分析. 第 3 节介绍本文提出的基于顶点组重分配的动态增量图划分算法的详细内容, 包括算法动机、算法框架、算法设计以及时空复杂度分析. 第 4 节通过对比实验验证本文提出的算法的有效性. 最后总结全文.

1 相关工作

早期的研究者针对静态图提出了大量的图划分算法, 主要分为基本图划分算法、多级图划分算法、局部图划分算法、分布式图划分算法和基于标签传播的图划分算法等. 基本图划分算法根据不同的算法特征可分为谱划分算法[10]、几何划分算法[11]和分支定界算法[12]等. 基本的图划分算法由于复杂度过高只能划分小规模的图. 随着图规模的持续增长, 基本图划分算法无法满足需求, 于是研究者提出了多级图划分算法. 其中最经典的方法是 Metis[13,14], 使用了多级划分策略, 但在扩展性上有缺陷. 此外, 还有 KL[16], FM[17]等图划分算法被提出进行局部的优化. 为了提高可扩展性, 一些研究者提出了分布式的图划分算法, 如 PT-Scotch[15], JA-BE-JA[31]. 文献 [32] 发现了目前静态图划分中存在的一些问题, 提出了多级标签传播算法 MLP.

以上这些方法都是基于静态图划分提出来的, 真实世界中的图是动态变化的并且是持续增长的. 近年来, 已有大量的研究者提出了动态图划分算法, 它的目标是当图发生动态变化时维持分区的负载平衡, 同时最小化被切割的边数. 动态图划分算法从不同的关注点和特点出发, 主要可以分为流式图划分算法、动态图重划分算法和增量图划分算法. 其中流式图划分算法依次处理顶点流, 可以用来划分动态图. 动态图重划分算法的特点是没有单独的划分策略来处理动态变化部分, 它专注于在动态图的一个快照上通过局部调整不断地优化分区. 而增量图划分算法为动态变化部分提出了处理策略, 该算法的特点是更专注于处理和划分动态变化部分, 通过将动态变化部分的划分与原图的划分融合, 从而获得整个动态图的划分.

流式图划分算法的设计之初针对的是常规大图划分, 它将整个图视为一个顶点流, 依次将每个顶点划分到分区中. 由于该算法可以处理新出现的顶点, 所以可以用来划分动态图, 即某个时刻的动态图的划分结果可以被视为流式划分中的一个中间状态. 文献 [18] 提出了一种基于 Hash 的简单但有效的划分方法. 此外, 文献 [18] 也提出了一种基于邻居分布的流式划分算法 LGD. 在 LGD 的基础上, FENNEL[19]将图划分问题描述为流环境中的模块度最大化, 并放宽了分区负载的硬约束. AKIN[20]用 Jaccard 系数来计算顶点之间的相似性, 而不是仅根据邻居数量来

分配顶点. 此外, 文献 [21] 在 LGD 的基础上考虑到集群中网络带宽以及节点计算能力的不同, 提出了一种异构感知的流式图划分算法. 但是上述这些方法均为单元素的流式算法, 即每次划分一个顶点. 在无法得知后续流信息的情况下, 为了平衡分区的负载, 一些顶点会违背贪心策略而被分配到负载较小的分区, 会导致划分质量的下降. 为了解决这些缺陷, 一些基于流的优化划分方法被提出, 如 Loom[22]. 然而, 由于这些方法需要收集一段时间内的动态变化或者操作整个图, 因此它们并不适合用于划分实时持续增长的动态图. 此外, 所有的流式图划分算法都没有考虑图中顶点或边的删除.

针对图结构的动态性, 一些研究提出了动态图重划分算法. 动态图重划分算法的特点是假设图发生动态变化已经导致分区发生了变化, 它更专注于在动态图的一个快照上通过动态调整不断优化分区. 例如文献 [23] 提出了多级图的重划分算法, 在初始划分的基础上, 通过粗化、多级扩散和多级优化 3 个阶段来获取新的划分. 由于多级的划分策略不适合于大规模图, 近期有许多轻量级的重划分算法被提出, 它们只需要少量的关于图的信息. 例如 XDGP[24]在每次迭代中利用标签传播将单个顶点转移到邻居多的分区中. Hermes[25]是应用在图数据库 Neo4j[6]上的重划分算法. 然而, 它有可能会出现转移的震荡, 因此文献 [26] 提出了一种有效的两阶段方法以解决这个问题. 上述的这些方法都假设图划分的环境是同构的且无网络争用的, 也就是各分区之间的单位通信代价都一样. 因此 Planar[33]提出了在异构环境中的图划分. 然而, 上述的这些重划分算法不能够实时地处理图的动态变化, 不满足实时动态图的划分. 此外, 它们都是针对图结构的动态性而提出来的. 但是在分布式图处理系统中, 图计算大多都是迭代进行的, 在每个超步中每个分区都有一部分顶点被激活而进行计算. 为了提升图计算的效率, 一些研究者提出了针对计算动态的重划分算法, 如 CatchW[34], LogGP[35]等.

近年来, 针对图结构的动态性有些研究者提出了增量图划分算法, 这类算法适合于持续增长的动态图. 根据处理动态变化的规模可以将增量图划分算法分为批处理增量图划分算法和单元素增量图划分算法. 顾名思义, 批处理增量图划分算法是针对一批动态变化提出的划分策略. 早期, 文献 [27] 提出了基于线性规划的增量图划分方法, 但是由于计算复杂度过高, 它不能够处理大规模图. 最近, 文献 [28] 为已有的静态图划分算法提出了增量式版本来解决动态图的划分问题, 由于它只是处理动态变化的那部分图数据, 因此划分效率比利用全局算法划分整个图高, 如静态图划分算法 KGGGP[36]的增量式版本 IncKGGGP. 但是这些方法是收集一段时间内的动态变化, 而不是实时处理单个动态变化. 此外, 由于批处理增量图划分算法更专注于处理动态变化部分, 没有使用局部优化技术, 因此其划分质量不高. 而单元素增量划分算法是为每个单独的动态变化提供了划分策略. 例如文献 [37] 为顶点、边的插入或删除分别提出了处理策略, 能够满足实时动态图的划分. 但是对于新的顶点, 它假设新顶点的大部分邻居信息是可用的, 以将该顶点划分到邻居最多的分区中. 而这与一些真实世界中的图的动态性特点不相符. 例如在社交网络中, 某个时刻会有新的用户注册, 但是短时间内缺乏邻居信息. 因此该方法具有一定的局限性. Leopard[29]专门针对只读图计算任务设计的单元素增量图划分算法. IOGP[30]提出了一种用于分布式图数据库的增量在线图划分算法. 然而, IOGP 的设计目的是为图数据库上的联机事务处理操作 OLTP 实现高性能而设计的, 具有一定的局限性.

2 基础知识

本文研究的是动态增量图划分算法来解决实时大规模动态图的划分问题, 下面介绍问题定义和问题分析.

2.1 问题定义

首先给出与图划分问题相关的一些基本概念, 并在此基础上定义本文所研究的动态增量图划分问题.

定义 1. 图 (graph). 给定一个图 $G=(V,E)$, V 是图 G 中的顶点集合, E 是图 G 中的边集合. $|V|$是图 G 中的顶点数量, $|E|$是图 G 中的边数量. 对于图 G 中的任意一个顶点 v, 用 $N(v)$ 表示顶点 v 的邻居顶点的集合.

定义 2. 图划分 (graph partitioning). 基于边割的图划分的划分对象是图 G 中的顶点集合 V. 若图 G 被划分成 k 个分区 $P=\{P_1,P_2,\ldots,P_k\}$, k 是一个远小于$|V|$和$|E|$的正整数, 满足 $V=\bigcup_{i=1}^{k}P_i$ 且当 i 不等于 j 时 $P_i\cap P_j=\varnothing$, 其中 P_i 是 P 的第 i 个分区, 那么称 P 是图 G 的一个顶点划分.

对于图 G 中的一个顶点子集 $S\subseteq V$, 用 $E(S,V\setminus S)$ 表示顶点子集 S 和 V 中的其他顶点之间连接边的集合, 并且

用$|E(S, V\backslash S)|$表示顶点子集 S 和 V 中的其他顶点之间连接边的数量. 在分布式图处理和图计算中, 不同分区的顶点之间需要传递消息, 而由于网络价格、网络带宽以及网络延迟等因素, 不同分区之间的消息传递成本较高, 因此需要最小化不同分区之间边被切割的数量. 此外由于分布式图任务的整体性能由最慢的机器决定, 因此还需要保证每个分区的负载尽可能相等, 即每个分区的顶点数几乎相同. $EC(P)$ 表示在图 G 的一个顶点划分 P 下的边切割的数量, 因此基于边割的图划分的问题定义如公式 (1) 所示.

$$\min EC(P)=\frac{1}{2}\sum_{i=1}^{k}|E(P_i,V\backslash P_i)| \text{ s.t. } |P_i|\leqslant(1+\varepsilon)\frac{|V|}{k} \tag{1}$$

其中, ε 是满足 $0\leqslant\varepsilon\leqslant 1$ 的不平衡系数. 这里, $\varepsilon=0$ 表示每个分区具有相同顶点数量的最严格情况, 而 $\varepsilon=1$ 表示每个分区的顶点数量可能高达平均顶点数量的 2 倍.

定义 3. 负载平衡 (load balance, LB). 给定图 G 包含 k 个分区的一个划分 $P=\{P_1,P_2,\ldots,P_k\}$. $LB(P)$ 表示在图 G 的一个顶点划分 P 下的负载平衡, 如公式 (2) 所示.

$$LB(P)=\sum_{i=1}^{k}\left|W(P_i)-\frac{|V|}{k}\right| \tag{2}$$

其中, $W(P_i)$ 表示分区 P_i 中的顶点数量.

定义 4. 边界顶点 (boundary vertex). 如果分区 P_i 中的一个顶点 v 与 P 中的其他分区中的顶点存在边, 那么该顶点 v 就是分区 P_i 中的一个边界顶点.

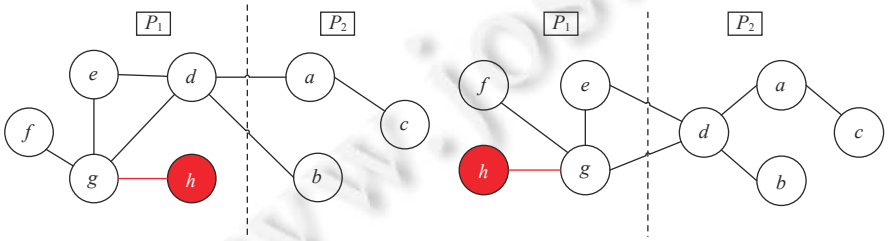
定义 5. 动态图 (dynamic graph). 真实图应用中, 图 $G=(V, E)$ 中存在顶点或边的插入或删除. 在一段时间内, 图 G 转变为图 $G'=(V',E')$, $V'=V\cup V_I-V_D$, $E'=E\cup E_I-E_D$. 其中 V_I 和 E_I 分别表示图中新插入的顶点和边的集合, V_D 和 E_D 分别表示图中被删除的顶点和边的集合. 将动态变化的那部分图数据标记为ΔG, 即 $\Delta G=V_I\cup V_D\cup E_I\cup E_D$.

定义 6. 动态增量图划分 (dynamic incremental graph partitioning). 对于初始图 $G=(V, E)$, 当图发生动态变化变为 $G'=(V',E')$ 时. 动态图划分问题是在原来 G 的初始划分 P 的基础上, 处理动态变化图数据ΔG 以获得一个更新后的划分 P', 并且 P' 需要在维持各个分区中的顶点数量尽可能相等的情况下最小化边切割 $EC(P')$. 此外, 对于动态图划分问题, 除了要保证划分质量外, 还需要最小化 P 和 P' 之间的差异, 减少划分成本和时间开销.

2.2 问题分析

我们分别介绍顶点插入、边插入、顶点删除和边删除这 4 种不同类型的动态变化对图划分质量的影响.

● 顶点插入. 新顶点的插入有可能会导致分区负载不平衡, 从而影响图划分的质量. 图 2 展示了新顶点插入时的一个例子. 从图 2(a) 可看出, 当新顶点 h 插入到 P_1 分区时, 会使得 P_1 分区中的顶点数从 4 变为 5, 而 P_2 分区中的顶点数只有 3, 那么这会导致 P_1 分区和 P_2 分区的负载严重不平衡, 进而影响图划分的质量. 如果将顶点 d 从 P_1 分区移动到 P_2 分区, 那么这两个分区中的顶点数都为 4, 达到负载平衡, 如图 2(b) 所示. 此外, 此次通过移动顶点 d 进行局部结构调整也使得边切割的数量从 3 降为 2, 从而获得了一个局部最优顶点划分.

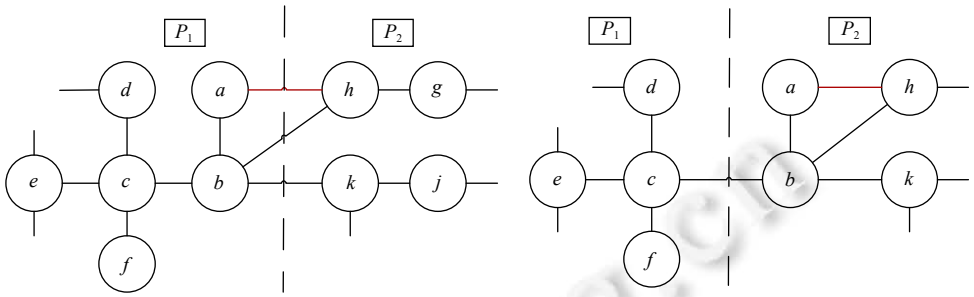


(a) 插入顶点 h 导致分区负载不平衡　　(b) 动态更新后的局部最优顶点划分

图 2　顶点插入对划分质量的影响

● 边插入. 如果插入新边的两个端点处于不同的分区中, 会导致边切割的数量增加 1, 从而导致划分质量的下降. 图 3 展示了新边插入的一个例子. 图 3(a) 是由分区 P_1 和 P_2 组成的一个局部划分, 起初边 (b, h) 和 (b, k) 被切

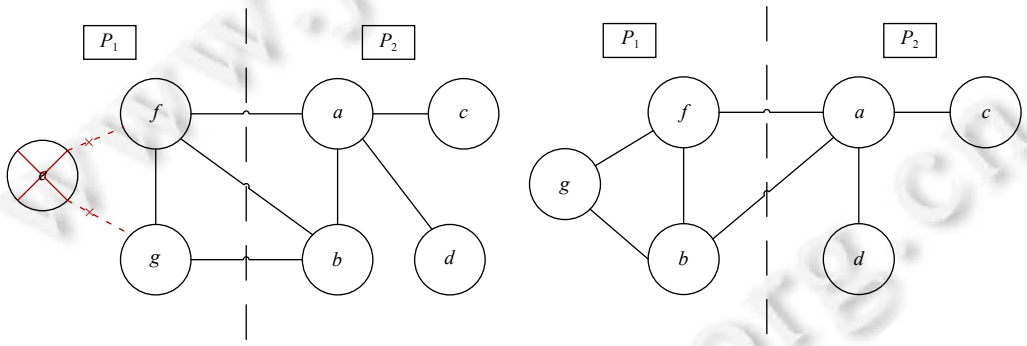
割, 当动态变化插入一条新边 (a, h) 时并且将它分配到该划分中, 使得边 (a, h) 也被切割, 从而边切割的数量从 2 变为 3, 边插入不可避免导致了一个局部次优顶点划分. 而实际上如果在分配完新边之后, 通过调整分区边界区域的局部结构, 将顶点 a 和 b 组成的一个顶点组从分区 P_1 移动到分区 P_2, 那么就会实现一个局部最优顶点划分, 如图 3(b) 所示.



(a) 插入边 (a, h) 导致局部次优顶点划分　　(b) 动态更新后的局部最优顶点划分

图 3　边插入对划分质量的影响

● 顶点删除. 当删除一个顶点时, 会同时删除该顶点的连接边, 顶点的删除与顶点的插入同样有可能会导致分区负载不平衡. 图 4 展示了一个顶点删除的例子. 当从现有的图中删除顶点 e 时, 如图 4(a) 所示, 会使得 P_1 分区中的顶点数变为 2, 而 P_2 分区中的顶点数为 4, 这导致了分区负载的严重不平衡, 从而出现了一个局部次优顶点划分. 如果将顶点 b 从 P_2 分区移动到 P_1 分区, 那么会使得 P_1 分区和 P_2 分区中的顶点数都为 3, 同时边切割的数量也从 3 降为 2, 从而在删除顶点后通过局部结构的调整达到了一个局部最优顶点划分, 如图 4(b) 所示.



(a) 删除顶点 e 导致分区负载不平衡　　(b) 动态更新后的局部最优顶点划分

图 4　顶点删除对划分质量的影响

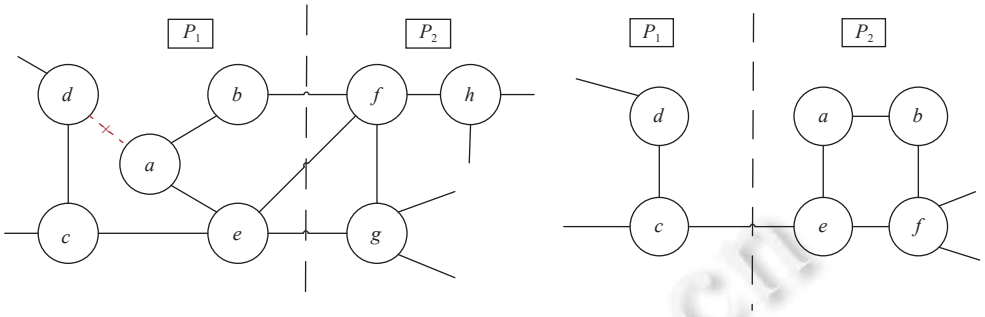
● 边删除. 如果要删除边的两个端点处于同一分区中, 有可能会导致划分质量的下降. 后文图 5 展示了边删除的一个例子. 从图 5(a) 可看出, 当将边 (a, d) 从现有的图中删除之后, 在分区边界区域导致出现了局部次优顶点划分. 图 5(b) 展示了该分区边界区域的局部最优顶点划分, 可以看到将顶点 a、b 和 e 作为一个顶点组整体移动后, 会导致边切割的数量从原来的 3 降为 1, 从而获得了一个更好的划分结果.

3　基于顶点组重分配的动态增量图划分算法

3.1　算法动机

真实世界中的图大多都是不断变化并且持续增长的. 动态图划分方法需要实时处理每次传入的动态变化, 包括插入顶点或边到分区中和从分区中删除顶点或边. 因此, 需要设计动态处理器专门处理图的动态变化, 得到由初始图和动态变化的那部分图数据所构成的一个新的划分. 但是如果只使用动态处理器处理动态变化来解决动态图

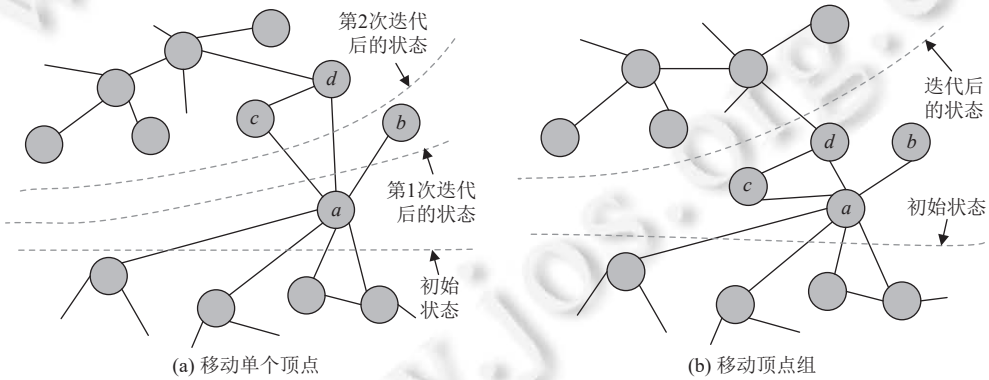
划分问题, 则无法保证图划分的质量, 前面我们已经详细分析了该问题. 因此, 在利用动态处理器处理完图的动态变化后, 在涉及的分区中利用局部优化器进行局部优化, 从而获得高质量的划分结果.



(a) 删除边 (a, d) 导致局部次优顶点划分 (b) 动态更新后的局部最优顶点划分

图 5 边删除对划分质量的影响

对于利用局部优化算法获得高质量的划分结果, 当利用动态处理器处理完动态变化后, 我们可以将此时的划分状态视为当前状态, 将分区边界顶点的移动视为当前状态的邻居解. 在每次迭代中, 我们考虑从相邻解中选择最优解, 这意味着该解使我们的目标函数最小. 这个过程继续, 直到达到一个局部最优. 然而, 直接采用通过移动单个顶点做局部优化可能会陷入局部次优顶点划分中, 即可能不会搜索到局部最优顶点划分, 并且在迭代过程中收敛速度较慢. 以图 6(a) 所示的示例为例. 对于初始状态, 即当前解决方案, 顶点 a 的移动是当前状态的相邻解决方案, 可以看出顶点 a 的移动使边切割的值从 4 降为 3. 由于相邻解决方案比当前解决方案更好, 那么将移动顶点 a. 然后会将第 1 次迭代后的状态作为当前状态, 其顶点 b、c 和 d 的移动分别是当前状态的相邻解决方案, 可以看出顶点 b 的移动使得边切割的值从 3 降为 2, 顶点 c 的移动使边切割的值不变, 顶点 d 的移动使边切割的值从 3 增加到 4. 因此, 对于当前状态, 会从所有相邻解决方案中选择顶点 b 的移动作为新的解决方案, 主要是因为顶点 b 的移动是所有相邻解决方案中的最佳解决方案, 并且它优于当前状态. 在此之后, 会将第 2 次迭代后的状态作为当前状态, 其中相邻的解决方案分别是顶点 c 和顶点 d 的移动, 但这两个解决方案都没有当前状态好. 因此, 通过移动单个顶点的局部优化技术不做任何处理, 停止进一步搜索, 达到局部次优解.



(a) 移动单个顶点 (b) 移动顶点组

图 6 通过移动单个顶点和移动顶点组做局部优化用于改进划分质量的效果比较分析

实际应用中大多数自然图的度分布遵循幂律分布, 即少数顶点有大量邻居顶点, 而大多数顶点有很少邻居顶点[4]. 基于这个特征, 我们观察到一个有效的方法是通过移动一系列高度紧密连接的顶点组成的顶点组, 即顶点组内紧密连接, 而顶点组与外部松散连接. 如图 6(b) 中由顶点 a、b、c 和 d 组成的顶点组, 整体移动该顶点组会使边切割的值从 4 降为 1, 从而达到局部最优解, 这比通过移动单个顶点做优化得到的结果要好. 因此, 本文考虑通过移动顶点组做局部优化来提高图划分的质量.

3.2 算法框架

图划分问题是一个具有有限解集的组合优化问题, 其优化目标是最小化边切割和分区负载平衡. 本文研究的是动态增量图划分算法, 当图发生动态变化时, 利用动态处理器处理完动态更新后, 再利用局部优化算法进一步提高划分质量. 本文提出了基于顶点组重分配的动态增量图划分算法 ED-IDGP 来解决实时并持续增长的大规模动态图的划分问题, ED-IDGP 算法的框架图如图 7 所示. 首先在输入初始图 G 上利用已有的图划分算法得到一个初始划分 $P=\{P_1,P_2,\ldots,P_k\}$. 图动态更新ΔG 是由一系列单元更新组成, 即顶点的插入、边的插入、顶点的删除和边的删除. 图更新ΔG 以动态更新流的方式均匀分配给 k 个动态处理器. 动态处理器用于接收和处理每一个单元更新. 对于顶点的插入或边的插入, 动态处理器会根据当前分区的信息确定其赋值, 并将其发送给目标分区. 对于顶点的删除或边的删除, 动态处理器将查找当前分区的顶点或边. 如果不在当前分区中, 它将被发送到它所在的分区中. 在动态处理器处理完每个单元更新后, 如果分区发生变化, 那么触发局部优化器, 使得分区边界区域从局部次优顶点划分转变为局部最优顶点划分, 从而进一步提高图划分的质量.

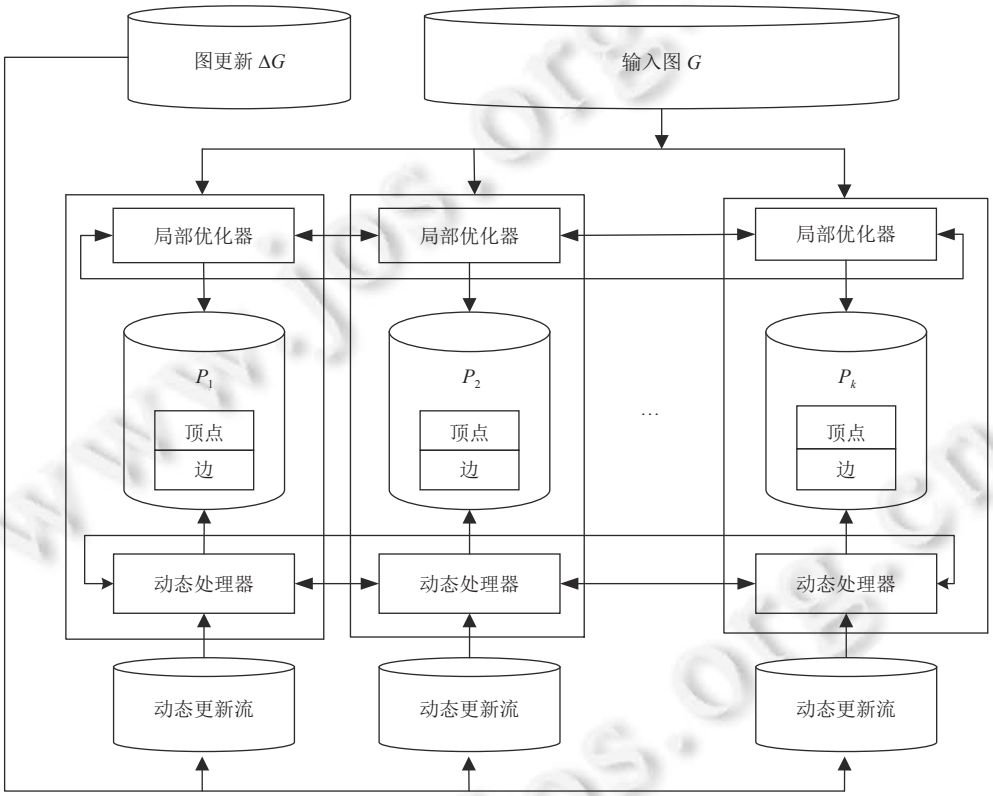


图 7 ED-IDGP 算法框架图

3.2.1 动态处理器

本文分别为顶点插入、边插入、顶点删除和边删除这 4 种不同的图更新类型介绍动态处理策略.

● 顶点插入. 虽然已有的一些工作 (例如, LDG[18], FENNEL[19], Leopard[29], 文献 [37] 等) 为顶点插入提供了划分策略, 但是它们都假设插入新顶点的大部分连接边在那时是已知的, 即一些局部图信息是已知的. 然而, 在许多真实的分布式图处理或者图计算任务中, 图是实时发生变化的, 某个时刻插入一个新顶点并不能获取到该顶点的大部分邻居信息. 例如, 在分布式图数据库中, 顶点及其连接的边通常从多个客户端独立且并发的插入. 这就需要图划分算法能够在拥有有限图知识的情况下, 满足动态图划分的需求, 而现有的方法是不适用或者无效的.

本文考虑顶点插入的两种情况, 其一是某个时刻新顶点插入时, 还没有关于该顶点的边信息, 那么为了考虑负

载平衡, 将新顶点分配到拥有最低负载大小的分区中. 其二是某个时刻新顶点插入时, 有关于该顶点的一条边信息随之插入, 为简单起见, 将该新顶点和这条边分配到其边的另一个端点所在的分区中. 这里指出, 对于该新顶点的其他边, 在下一时刻, 会将其作为新边对待, 后面会具体介绍对应的划分策略.

● 边插入. 在增量图划分中, 每一条边插入需要在负载平衡的基础上最小化边切割的数量. 对于新边的插入可分为两种情况. 第 1 种情况是新边的两个端点在同一分区, 因为本文研究的是基于边割的图划分问题, 所以对于这种情况直接将该条边分配到端点所在的分区中. 第 2 种情况是新边的两个端点在不同的分区中, 如果直接将该条边分配到现有的划分中, 那么会导致边切割的数量增加 1. 对于这种情况, 给定一条新边 (v_i, v_j), 端点 v_i 所在的分区标记为 P_s, 端点 v_j 所在的分区标记为 P_t, 本文提出了一个分数度量 $f(v_i, v_j)$ 来确定这条边的分配, 如公式 (3) 所示.

$$f(v_i, v_j) = \arg\max\left(N_{v_i}^t - N_{v_i}^s, N_{v_j}^s - N_{v_j}^t\right) \tag{3}$$

其中, $N_{v_i}^t$ 表示端点 v_i 在分区 P_t 中的邻居的数量, $N_{v_i}^s$ 表示端点 v_i 在它所在分区中的邻居的数量, $N_{v_j}^s$ 表示端点 v_j 在分区 P_s 中的邻居的数量, $N_{v_j}^t$ 表示端点 v_j 在它所在分区中的邻居的数量. $f(v_i, v_j)$ 指出了将边 (v_i, v_j) 分配到分区中, 并调整局部分区结构. 直觉上, 将边 (v_i, v_j) 和端点 v_i 分配到端点 v_j 所在的分区 P_t, 或者将边 (v_i, v_j) 和端点 v_j 分配到端点所在的分区 P_s, 这取决于哪种分配会使得边切割的数量更低.

● 顶点删除. 将一个顶点从现有的划分中删除, 那么同时会将该顶点的连接边一同删除. 顶点的删除有可能会导致分区负载不平衡, 即会出现分区欠负载的情况. 因此, 当一个顶点删除时, 选择从高负载的分区中移动顶点到欠负载的分区中, 并且要保证不能增加边切割的数量.

● 边删除. 对于删除一条边存在两种情况. 第 1 种情况是边的两个端点在不同分区中, 如果将该条边删除, 那么会导致边切割的数量减 1. 对于这种情况, 不会做额外的动作. 第 2 种情况是边的两个端点在同一分区中, 由于本文研究的是基于边割的图划分问题, 所以删除该条边不会导致划分质量的下降.

本节分别提出了不同图更新类型的动态处理策略. 但从问题分析中可知, 图的动态变化会影响划分质量, 在分区边界区域会导致出现局部次优顶点划分. 因此在处理完每一个动态变化后, 在涉及动态变化的分区中, 通过基于顶点组重分配的局部优化技术进一步提高图划分的质量, 使得达到局部最优顶点划分, 该局部优化方法接下来进行介绍.

3.2.2 局部优化器

当图发生动态变化时, 对于涉及单元更新的分区, 在单元更新的附近有可能会出现局部次优顶点划分. 基于该特点, 在 ED-IDGP 算法中, 本文提出一种基于改进标签传播算法的顶点组搜索策略搜索候选顶点组, 并且提出了顶点组移动增益公式衡量最有益的顶点组 VG. 由于在基于改进标签传播算法的顶点组搜索策略的描述中需要用到顶点组移动增益公式, 所以我们首先介绍本文提出的同时考虑边切割增益和负载平衡增益的顶点组移动增益公式, 然后介绍基于改进标签传播算法的顶点组搜索策略.

定义 7. 负载平衡增益 (load balance gain). 对于一个给定的顶点组 VG, 将顶点组 VG 从分区 P_s 移动到分区 P_t 的负载平衡增益定义为公式 (4).

$$lb: gain^{s\to t}(VG) = LB(P) - LB(P') \tag{4}$$

其中, $LB(P)$ 表示顶点组 VG 移动之前整个划分的负载平衡, $LB(P')$ 表示顶点组 VG 移动之后整个划分的负载平衡. 由于负载平衡的变化只有顶点组 VG 所在的分区 P_s 和目标分区 P_t 决定, 所以顶点组 VG 的负载平衡增益也可以用公式 (5) 表示.

$$lb: gain^{s\to t}(VG) = \left|W(P_s) - \frac{|V|}{k}\right| + \left|W(P_t) - \frac{|V|}{k}\right| - \left|W(P_s - VG) - \frac{|V|}{k}\right| + \left|W(P_t + VG) - \frac{|V|}{k}\right| \tag{5}$$

其中, $W(P_s)$ 和 $W(P_t)$ 分别表示 VG 移动之前分区 P_s 和 P_t 中的顶点数, $W(P_s) - W(VG)$ 和 $W(P_t) + W(VG)$ 分别表示 VG 移动之后分区 P_s 和 P_t 中的顶点数.

定义 8. 边切割增益 (edge cut gain). 对于一个给定的顶点组 VG, 将顶点组 VG 从分区 P_s 移动到分区 P_t 的边切割增益定义为公式 (6).

$$ec:gain^{s\to t}(VG)=EC(P)-EC(P') \tag{6}$$

其中, $EC(P)$ 表示顶点组 VG 移动之前整个划分的边切割, $EC(P')$ 表示顶点组 VG 移动之后整个划分的边切割. 类似地, 边切割的变化只有顶点组 VG 所在的分区 P_s 和目标分区 P_t 决定. 因此, 我们用公式 (7) 表示顶点组 VG 的边切割增益.

$$ec:gain^{s\to t}(VG)=N_t(VG)-N_s(VG) \tag{7}$$

其中, $N_t(VG)$ 和 $N_s(VG)$ 分别表示顶点组 VG 在分区 P_t 和 P_s 的其他部分中的邻居的数量.

定义 9. 顶点组移动增益 (vertex group migration gain). 为了综合考虑边切割和负载平衡这两个优化目标, 在负载平衡增益和边切割增益的基础上, 公式 (8) 给出了将顶点组 VG 从它所在的分区 P_s 移动到目标分区 P_t 的移动增益.

$$vgm:gain^{s\to t}(VG)=\gamma\times\frac{ec:gain^{s\to t}(VG)}{|E|}+(1-\gamma)\times\frac{lb:gain^{s\to t}(VG)}{|V|} \tag{8}$$

其中, γ 同样指定了边切割和负载平衡之间的相对重要性, 且 γ 是介于 0–1 之间的值. 如果 γ 等于 0, 那么意味着不考虑负载平衡. 如果 γ 等于 1, 那么意味着不考虑边切割. γ 越大, 划分越平衡. γ 越小, 边切割越低.

基于改进标签传播算法的顶点组搜索策略考虑在每个分区中, 当出现动态变化后, 在发生单元更新的附近区域搜索顶点组 VG. 前面我们提出了动态处理器, 分别针对 4 种不同的动态变化类型提出了不同的处理策略. 但是图划分问题有两个目标, 一是最小化不同分区之间边被切割的数量, 即 min $EC(P)$; 二是每个分区中的顶点的数量要尽可能相等, 即 min $LB(P)$. 所以当考虑从一个分区移动顶点组 VG 到其他分区中, 要想降低边切割的数量, 必须在该顶点组 VG 中包含分区的边界顶点. 因此, 本文采用的基于改进标签传播算法的顶点组搜索策略必须从单元更新附近的边界顶点开始逐步搜索顶点组 VG. 接下来我们对顶点插入、边插入、顶点删除以及边删除这 4 种不同的情况分别进行讨论分析.

首先, 对于顶点插入, 考虑顶点插入的两种情况. 第 1 种情况是新插入的顶点作为孤立顶点分配到拥有最低负载大小的分区中, 主要是为了使得每个分区的负载大小平衡, 在这种情况下局部优化器不做考虑. 第 2 种情况是新插入的顶点与其相关的一条边信息随之插入, 正如图 2 所分析的, 该种情况下在图动态变化的附近会出现局部次优顶点划分, 因此需要局部优化器做优化, 使得从局部次优顶点划分转变为局部最优顶点划分. 但是很明显新插入的顶点并不是一个边界顶点, 所以需要沿着新插入的这个顶点开始, 从它的邻居中搜索出邻居跳数相对较小的一些边界顶点. 这里指出, 对于顶点插入这种单元更新, 在做局部优化时, 只涉及一个分区.

其次, 对于边插入, 同样也考虑边插入的两种情况. 第 1 种情况是新插入的边的两个端点都在同一分区中, 该种情况下只涉及一个分区. 如果新插入边的两个端点中有边界顶点, 那么基于改进标签传播算法的顶点组搜索策略从这些边界顶点开始进行搜索. 但是如果新插入边的两个端点都不是边界顶点, 那么同样我们需要在这两个端点的邻居中搜索出邻居跳数相对较小的某些边界顶点. 第 2 种情况是新插入的边的两个端点在两个不同的分区中, 该种情况下会涉及两个分区, 即需要在边插入这种单元更新类型涉及的两个分区中执行局部优化器做优化. 正如图 3 所示, 新边涉及的两个端点都是边界顶点, 所以从这些边界顶点开始利用局部优化器做优化.

再次, 对于顶点删除, 正如图 4 所示, 顶点的删除有可能会导致分区出现欠负载的情况. 对于这种情况, 与前面分析的其他情况不同, 我们不能在该顶点删除涉及的分区中移出顶点组 VG, 这样只会更加使得分区负载不平衡. 因此, 为了让每个分区中的顶点数尽可能相等, 我们考虑从所有分区中找出拥有最高负载的分区, 从高负载分区的边界顶点开始, 搜索顶点组 VG 并将其移动到欠负载的分区中.

最后, 对于边删除, 考虑边删除的两种情况. 第 1 种情况是要删除边的两个端点在两个不同的分区中. 这种情况下需要分别在这两个分区中执行局部优化器, 需要从这些端点的邻居中找出邻居跳数相对较小的一些边界顶点, 从而从这些边界顶点开始搜索可用于移动的候选顶点组 VG. 第 2 种情况是要删除边的两个端点在同一分区中, 正如图 5 所示, 此种情况只涉及一个分区, 同样需要从这两个端点开始搜索出其邻居跳数相对较小的那些边界顶点.

基于改进标签传播算法的顶点组搜索策略如算法 1 所示, 算法的输入是当前的单元更新, 该单元更新涉及的分区 P_s, 以及顶点组移动增益公式 (8) 中参数 γ. 算法的输出是在单元更新附近搜索出的最有益的顶点组 VG. 这里指出, 在搜索顶点组 VG 时要用到标签传播技术[38,39]的原因. 一方面标签传播算法对于发现紧密联系的一组顶点是有效的并且拥有较低的复杂度, 紧密联系的一组顶点与本文定义的顶点组 VG 类似. 另一方面标签传播算法不只是能用于整个图上, 最重要的是它能在局部图上并行搜索紧密联系的顶点组, 使得算法效率大大提高.

算法 1. 基于改进标签传播算法的顶点组搜索策略.

输入: partition P_s, unit update, γ;
输出: vertex group (VG).

1. some boundary vertices BVS←GetBVS (P_s, unit update)
2. **foreach** boundary vertex $v \in BVS$ **do**
3. assign a new label to v according to formula (9)
4. **end**
5. **repeat**
6. the vertex which has label performs LPA
7. the vertex updates the label according to formula (10)
8. **foreach** the candidate vertex group with the same label **do**
9. **if** $W(P_s) - W(VG) < (1-\varepsilon)\ |V|/k$ **then**
10. **continue**
11. **end if**
12. update *migration_gain_array* for VG
13. select $P_t \leftarrow \arg\max(migration\_gain\_array[t-1])$
14. **if** $vgm: gain^{s\to t}(VG) > 0$ **then**
15. preserve VG
16. **end if**
17. **end**
18. **until** $W(P_s) - W(VG) < (1-\varepsilon)\ |V|/k$ for $\forall VG$
19. rollback to find VG with maximum migration gain
20. **return** VG

在算法 1 中, 首先由分区 P_s 和单元更新会得到一些在动态变化附近的边界顶点, 这些边界顶点用集合 BVS 表示 (行 1). 这里, 标签赋值和更新从 BVS 中的边界顶点开始. 初始时, 在 BVS 中的边界顶点 v 的标签为其外部邻居的最大数目所在的分区 ID, 这可以由公式 (9) 来计算 (行 2–4).

$$l(v) = \arg\max \sum_{u \in N_v^j} w(u),\ 1 \leqslant j \leqslant k \text{ and } j \neq i \tag{9}$$

其中, $l(v)$ 表示的是边界顶点 v 的新标签, N_v^j 表示的是顶点 v 在分区 P_j 中的邻居集合. 之后, 这些边界顶点将传播标签给它们所在的分区 P_s 中的邻居顶点 (行 6). 由于每个顶点可能收到多个来自边界顶点的标签, 收到标签传播信息的新的顶点将使用公式 (10) 提出的主投票规则赋值标签.

$$l(v) = \arg\max(l(v_1), \ldots, l(v_k)),\ v_k \in UN_v^s \tag{10}$$

其中, UN_v^s 表示的是顶点 v 在其所在分区 P_s 中的已更新标签的邻居集合, $l(v)$ 表示的是其已更新邻居所拥有的最大数量的标签. 这里指出, 当收到标签传播信息的顶点已经被赋值了一个标签, 那么同样利用公式 (10) 提出的主投票规则更新其标签 (行 7). 顶点组 VG 通过不断加入与其相连并且有着相同标签的新的顶点进行扩展. 对于每一

个具有相同标签的候选顶点组 VG, 如果该候选顶点组 VG 的移动会导致分区欠负载, 那么什么都不做, 跳过该候选顶点组 VG (行 9–11). 否则, 将根据公式 (8) 计算该候选顶点组 VG 的移动增益 $vgm:gain^{s\to t}(VG)$. 同样, 设计一个拥有 k 个元素的数组, 用 *migration_gain_array* 表示, 该数组中的每一项代表候选顶点组 VG 的移动增益, 移动增益利用公式 (7) 提出的综合考虑边切割和负载平衡这两个优化目标的方式来进行计算. 假设候选顶点组 VG 当前所在分区为 P_i, 则 *migration_gain_array* 数组中第 i 个元素对应该候选顶点组 VG 所在的分区, 我们设置其为 0, *migration_gain_array* 中第 j 个元素代表的是将该候选顶点组 VG 移动到分区 P_j 的移动增益值, 这里 $1\leqslant j\leqslant k$ 并且 $j\neq i$. 根据公式 (8) 更新每一个候选顶点组 VG 的 *migration_gain_array* 数组, 并且挑选出在该 VG 和目标分区之间拥有最大移动增益的目标分区 P_t (行 12–13). 在这个过程当中, 拥有正的移动增益值的候选顶点组 VG 被保留 (行 14–16). 搜索过程继续, 直到对于所有的候选顶点组 VG 而言, 移动它将违反其所在分区 P_s 的负载约束 (行 18). 最后, 该过程将回退到拥有最大移动增益的候选顶点组 VG, 并且将其移动到目标分区中 (行 19).

3.3 算法设计

在局部优化器中, 我们讨论和分析了顶点插入、边插入、顶点删除和边删除这 4 种不同的单元更新类型会涉及的分区发生变化的个数, 以及如何在每种单元更新附近找到一些边界顶点, 并从这些边界顶点开始利用基于改进标签传播的顶点组搜索策略搜索出最有益的顶点组 VG. 从中可以得知, 每个单元更新会涉及一个或两个分区. 当单元更新涉及两个分区时, 同时执行局部优化器有可能出现移动干扰, 移动干扰会导致算法效率降低和划分质量下降. 因此, 为了避免无效的移动和最大限度地提高划分质量, 在本文提出的 ED-IDGP 算法中, 采用一种顺序的方式解决移动干扰的问题. 对于一个单元更新会涉及两个分区发生变化的这种情况, 我们不是同时独立地在这两个分区中运行局部优化器, 而是一个一个地依次执行局部优化器. 这里指出, 由于只有两个分区有可能会出现移动干扰的问题, 所以采用顺序的方式对算法效率的影响不是很大, 同时也保证了图划分的质量不受影响.

本节提出了 ED-IDGP 算法 (如算法 2 所示) 解决动态增量图划分问题, 在拥有高效率的情况下最大限度获得高质量的划分结果. ED-IDGP 算法的输入参数是初始图 G 的一个划分 P, 动态更新 ΔG, 在顶点组移动 ED-IDGP 算法的详细过程如下: 对于在动态更新 ΔG 中的每一个单元更新, 利用提出的动态处理器进行处理 (行 2). 如果该单元更新是顶点删除, 在除该单元更新所涉及的分区的所有其他分区中, 选择拥有最高负载大小的分区考虑做移动 (行 3–6). 否则, 对于其他单元更新, 获得其涉及动态变化的分区 P_s (行 7–9). 将分区 P_s 和参数 γ 作为算法 1 的输入, 利用基于改进标签传播算法的顶点组搜索策略获得顶点组 VG (行 10). 从该顶点组 VG 的移动增益数组 *migration_gain_array* 中选择拥有最大元素值所代表的目标分区 P_t (行 11). 如果将该顶点组 VG 从分区 P_s 移动到分区 P_t 的移动增益值 $vgm:gain^{s\to t}(VG)$ 大于 0, 那么我们就将该顶点组 VG 移动到目标分区 P_t 中 (行 12–14).

算法 2. 基于顶点组重分配的动态增量图划分算法.

输入: partitioning P of G, dynamic update ΔG, γ;
输入: updated partitioning P' of $G+\Delta G$.

```
1. for each unit update in ΔG
2.    process it according to dynamic processor
3.    if the unit update is vertex deletion then
4.       foreach part P_i ∈ P do
5.          find P_s with maximum overload ratio
6.       end
7.    else
8.       P_s ← the unit update
9.    end if
10.   VG←基于改进标签传播算法的顶点组搜索策略 (P_s, γ)
11.   select P_t ← arg max (migration_gain_array[t − 1])
```

12. **if** $vgm: gain^{s\to t}(VG) > 0$ **then**
13. migrate VG to P_t
14. **end if**
15. **end for**
16. **return** P'

3.4 算法时空复杂度分析

ED-IDGP 算法首先需要利用动态处理器处理单元更新, 同样由于其简单但有效性, 这部分的时间复杂度可忽略不计. 对于 ED-IDGP 算法, 它的时间复杂度主要由基于改进标签传播算法的顶点组搜索策略的时间复杂度决定. 因为在基于改进标签传播算法的顶点组搜索策略中, 它不仅完成了顶点组搜索的过程, 而且也完成了找出拥有最大移动增益的顶点组 VG. 在顶点组搜索过程中, ED-IDGP 算法从单元更新附近的边界顶点开始, 利用标签传播的原理从而获得一系列的候选顶点组 VG. 因此, 改进标签传播至多需要 $O(|P_s|)$ 的时间复杂度. 事实上, 由于负载平衡的约束, 顶点组搜索的时间复杂度远远要小于 $O(|P_s|)$. 因此, 对于每一个单元更新, ED-IDGP 算法的时间复杂度为 $O(|P_s|)$. 对于所有的动态变化ΔG, 在 ED-IDGP 算法中, 我们使用了一种分布式流的方式将ΔG 分给 k 个动态处理器分别做处理, 所以其时间复杂度为 $O((|\Delta G|/k)\times|P_s|)$.

ED-IDGP 算法的空间复杂度主要由两方面决定, 一是在顶点组搜索过程中需要保留一些候选顶点组, 二是数组 *migration_gain_array*. ED-IDGP 算法需要额外的内存空间记录顶点组 VG 移动到其他分区 P_j 的移动增益值, 这里 $1 \leqslant j \leqslant k$ 并且 $j \neq i$. 数组 *migration_gain_array* 被用来存储这部分的数据, 其所需要的内存空间大小为 $O(k)$. 此外, 我们需要在单元更新附近搜索顶点组 VG 的过程中保留那些移动增益大于 0 的候选顶点组, 这也需要一个额外的内存空间存储, 我们用 $O(|PVG|)$ 标记这部分的内存空间, 它是一个很小的值. 因此, ED-IDGP 算法的空间复杂度为 $O(k+|PVG|)$.

Streaming 算法[18]采用一个评分函数来处理每一个单元更新, 每个分数的计算是一个常数时间的操作. 因此, 对于所有的动态变化中, 如果我们同样采用分布式流的方式将ΔG 分给 k 个动态处理器分别做处理, Streaming 算法的时间复杂度为 $O((|\Delta G|/k)\times k)$. Streaming 算法的空间复杂度只包括 k 个分区的评分. 因此, Streaming 算法的空间复杂度为 $O(k)$. 可以看到, Streaming 算法的时空复杂度都相对较小, 因为其只是把动态变化部分加入图中, 并没有对更新后的图进行进一步的划分以保证图的划分质量, 所以 Streaming 算法的划分质量相对较差.

4 实验分析

4.1 实验环境与数据集

本文主要使用真实图对 IDGP 算法和 ED-IDGP 算法的效果和性能进行评估和分析, 表 2 总结了实验用到的 6 个真实图[40,41]. 在这些真实图中, RoadNet-PA 是一个道路网络, DBLP 是一个 Web 网络, 其他是社交网络. RoadNet-PA 服从均匀分布, 其他服从幂律分布. 为了模拟动态环境, 我们将原始数据集随机分为两部分, 一部分作为初始图 G, 将被划分为 k 个分区, 而另外一部分作为动态变化ΔG. 对于所有的图, 我们生成由大小$|\Delta G|$控制的随机单元更新. 例如, 将图分为 70% 和 30%, 将 70% 那部分作为初始图进行划分, 将 30% 那部分作为动态变化部分. 由于初始图和动态变化的选择总体上是随机的, 因此会遵循一些动态变化规律以确保图的属性不会发生显著的变化. 例如, 动态更新导致直径更小并且平均度更高, 仅有很少的高度顶点会在动态更新过程中出现, 大量的高度顶点已经在初始图 G 中存在. 实验选择 Streaming 算法[18]获得初始图 G 的一个划分 P. 我们在一台内存大小为 32 GB, 并且处理器的配置为 Intel(R) Core(TM) i7-4790@3.60 GHz 的机器上执行所有的实验.

实验中, 我们选择单元素增量图划分方法 CCP[37]、批处理增量图划分方法 IncKGGGP[28]、流式图划分方法 Streaming[18]作为比较方法, 与本文提出的 IDGP 算法和 ED-IDGP 算法做对比.

(1) 单元素增量图划分算法 CCP 为顶点或边的插入或删除分别提出了划分策略, 满足实时图的划分. 但是 CCP 算法在处理动态变化时假设新顶点的大部分邻居信息是可用的, 具有一定的局限性. 此外, CCP 算法在做优化时仅考虑移动单个顶点.

(2) 批处理增量图划分算法 IncKGGGP 是已有的静态图划分算法 KGGGP[36]的增量式版本, KGGGP 是一种贪婪图生长方法. IncKGGGP 算法只处理动态变化部分, 没有做局部优化.

(3) 流式图划分算法 Streaming 提出了很多启发式用于将每一个新顶点分配到分区中, 在本实验中选用划分效果最好的 LDG 启发式[18], 该算法是流式图划分算法中最具有代表性的其中一个.

表 2 数据集的统计信息

图	$\|V\|$ (M)	$\|E\|$	平均度	最大度
DBLP	0.32	1.05M	6.62	3.43k
RoadNet-PA	1.09	1.54M	2.83	9
Youtube	1.13	2.99M	5.27	28.7k
LiveJournal	4.00	34.7M	17.4	14.8k
Orkut	3.07	117M	76.3	33.3k
Twitter	41.6	1.20G	57.8	3.00M

为了客观地比较我们提出的 IDGP 和 ED-IDGP 与最先进的方法的效果和性能, 我们使用以下度量指标. 图划分问题的目标之一是最小化不同分区之间边被切割的数量. 为了比较边切割的数量, 我们使用公式 (11) 定义的标准化边切割大小 *NEC* 来衡量.

$$NEC = \frac{EC(P)}{|E|} \tag{11}$$

此外, 我们也使用公式 (12) 定义的 *ECI* 来衡量与初始图 G 的划分 P 相比边切割的改进值.

$$ECI = \frac{EC(P) - EC(P')}{EC(P)} \tag{12}$$

图划分问题的另一个目标是使每个分区中的顶点数尽可能相等, 但分区不是绝对平衡的. 为了比较分区负载平衡, 我们使用公式 (13) 定义的 *LBI* 来衡量与初始图 G 的划分 P 相比负载平衡的改进值.

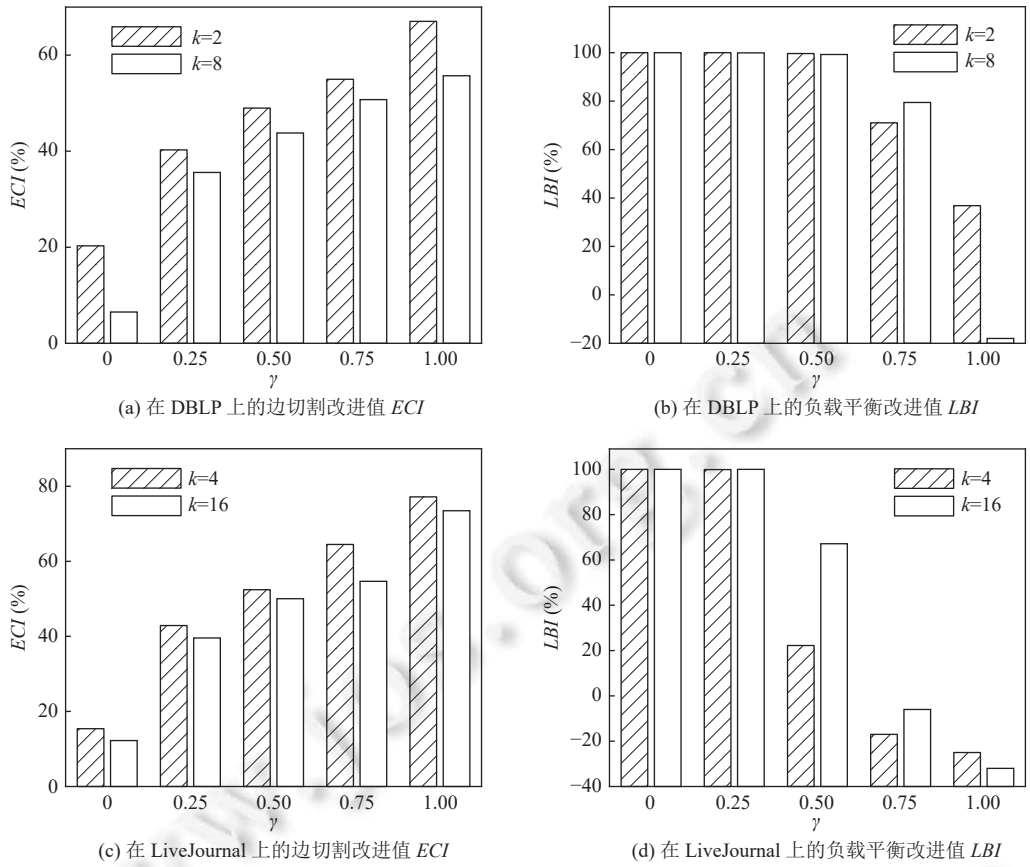
$$LBI = \frac{LB(P) - LB(P')}{LB(P)} \tag{13}$$

由于算法需要移动顶点来提高图划分的质量, 移动顶点需要一定的迁移成本. 为了比较迁移效率, 我们使用公式 (14) 定义的顶点迁移率 *VMR* 来表示不同分区之间顶点移动的个数与总顶点数的比率, 这衡量了由于动态变化 ΔG 而重分配 G 的成本.

$$VMR = \frac{migrated\ vertices}{|V|} \tag{14}$$

4.2 算法中的参数分析

本节分析 ED-IDGP 算法中在顶点组移动增益 $vgm : gain^{s\rightarrow t}(VG)$ 中所使用的参数 γ 对不同分区之间边切割的数量和分区负载平衡的影响. 我们选用数据集 DBLP 和数据集 LiveJournal 做评估和分析. 在每次实验中, 我们固定 30% 的动态变化, 主要涉及顶点或边的插入. 为了展示多样化的实验结果, 在数据集 DBLP 上我们设置 k 为 2 和 k 为 8 这两种情况, 在数据集 LiveJournal 上我们设置 k 为 4 和 k 为 16 这两种情况. 图 8 分别给出了在数据集 DBLP 和数据集 LiveJournal 上的边切割改进值 *ECI* 和负载平衡改进值 *LBI*. 最小化不同分区之间边被切割的数量和使得每个分区中的顶点数量尽可能相等是冲突的, 当参数 γ 的值为 0 时, 这意味着本文提出的 ED-IDGP 算法只关注最小化分区负载不平衡, 而完全忽略了最小化边切割. 因此, 在这种情况下, 分区的负载几乎是相等的. 从图 8(b) 和图 8(d) 可看出, 所有的结果都表明负载平衡改进值 *LBI* 是最大的. 同时, 从图 8(a) 和图 8(c) 可看出, 相应的边切割改进值 *ECI* 是最小的. 随着参数 γ 不断增大, 边切割所占的权重将不断增加, 即边切割改进值 *ECI* 将不断增大. 同时, 分区负载平衡所占的权重将不断减少, 即负载平衡改进值 *LBI* 将不断减小. 当参数 γ 的值为 1 时, 这意味 ED-IDGP 算法只关注边切割, 而完全忽略了分区负载平衡. 为了实现更低的边切割, 拥有更大移动增益的顶点组被允许从欠负载的分区移动到超负载的分区中. 然而, 这也许会导致严重的分区负载不平衡. 同样, 为了在边切割和分区负载平衡之间达到一个自适应的平衡, 在接下来的所有实验中, 我们将 γ 值设置为中间值 0.5.



(a) 在 DBLP 上的边切割改进值 *ECI*

(b) 在 DBLP 上的负载平衡改进值 *LBI*

(c) 在 LiveJournal 上的边切割改进值 *ECI*

(d) 在 LiveJournal 上的负载平衡改进值 *LBI*

图 8 ED-IDGP 算法中的参数 γ 对划分质量的影响

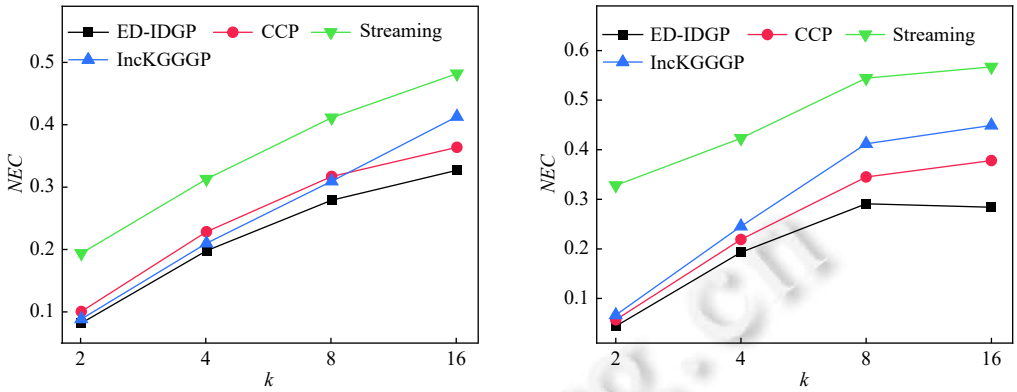
4.3 算法效果与性能的多维度评估

4.3.1 划分质量评估

在本实验中, 我们固定 40% 的动态变化, 其中 30% 是顶点或边的插入, 10% 是顶点或边的删除. 动态图划分问题的目标是最小化边切割 $EC(P')$ 和最小化负载平衡 $LB(P')$. 对于最小化边切割 $EC(P')$ 的目标, 我们在数据集 Youtube 和数据集 Twitter 上评估了标准化边切割 *NEC*, 如图 9 所示. 在图 9(a) 和图 9(b) 中, 横坐标表示的是分区的数量 k, 纵坐标表示的是标准化边切割 *NEC*. 从图 9 中可知: (1) 一般来说, 与现有的 CCP、IncKGGGP 和 Streaming 算法相比, 本文提出的动态增量图划分算法 ED-IDGP 具有更低的或者可比较的标准化边切割 *NEC*. (2) Streaming 算法有着最高的标准化边切割 *NEC*, 表现出了最差的划分质量, 这主要是因为 Streaming 算法提出了惩罚机制, 保证了图划分结果的负载平衡, 从而忽略了减少边切割. (3) 在数据集 Twitter 上, 当分区数 k 为 16 时, 本文提出的 ED-IDGP 算法的标准化边切割 *NEC* 分别比 CCP、IncKGGGP 和 Streaming 低 24.87%、36.75% 和 49.91%. (4) 在所有的数据集和分区数 k 上, ED-IDGP 算法的标准化边切割 *NEC* 平均比 CCP、IncKGGGP 和 Streaming 低 16.15%、20.54% 和 49.53%.

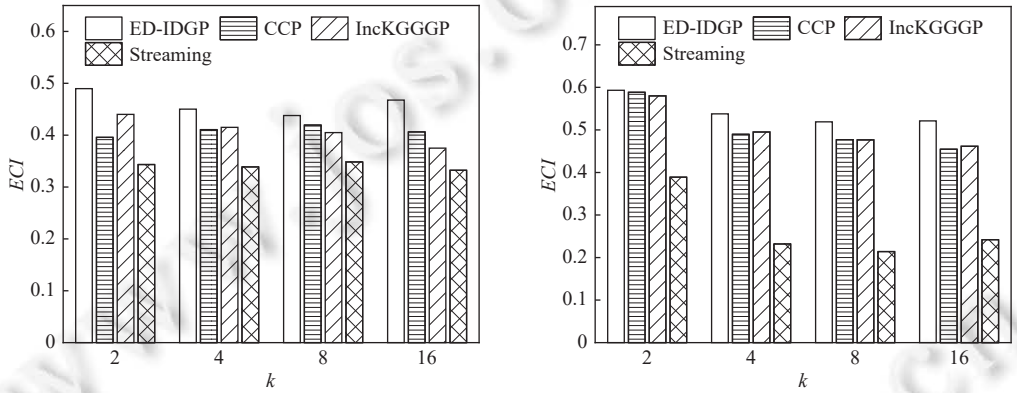
同时, 对于最小化边切割 $EC(P')$ 的目标, 我们在数据集 DBLP 和数据集 LiveJournal 上也评估了边切割改进值 *ECI*, 如图 10 所示. 在图 10(a) 和图 10(b) 中, 横坐标表示的是分区的数量 k, 纵坐标表示的是边切割改进值 *ECI*. 从图 10 中可知: (1) 与现有的 CCP、IncKGGGP 和 Streaming 算法相比, 本文提出的 ED-IDGP 算法具有更高的边切割改进值 *ECI*. (2) 类似地, Streaming 算法在所有数据集和分区数 k 上拥有最低的边切割改进值 *ECI*. (3) 当分区数 k 为 4 时, 在数据集 DBLP 上, ED-IDGP 算法的边切割改进值 *ECI* 分别比 CCP、IncKGGGP 和 Streaming 高

9.59%、8.48% 和 32.89%. (4) 在所有的数据集和分区数 k 上, ED-IDGP 算法的边切割改进值 *ECI* 平均比 CCP、IncKGGGP 和 Streaming 高 10.86%、10.75% 和 72.99%.



(a) 在 Youtube 上的标准化边切割 *NEC* (b) 在 Twitter 上的标准化边切割 *NEC*

图 9 不同的分区数量 k 所对应的标准化边切割 *NEC*



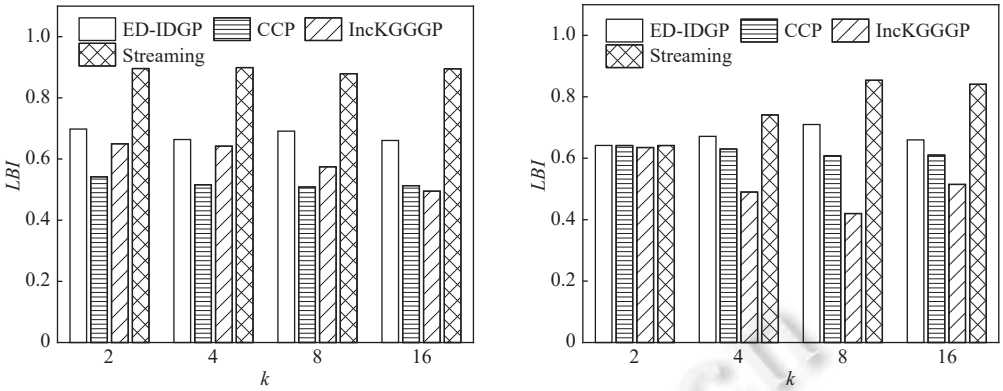
(a) 在 DBLP 上的边切割改进值 *ECI* (b) 在 LiveJournal 上的边切割改进值 *ECI*

图 10 不同的分区数量 k 所对应的边切割改进值 *ECI*

此外, 对于最小化负载平衡 $LB(P')$ 的目标, 我们在数据集 DBLP 和数据集 LiveJournal 上评估了负载平衡改进值 *LBI*, 如后文图 11 所示. 在图 11(a) 和图 11(b) 中, 横坐标表示的是分区的数量 k, 纵坐标表示的是负载平衡改进值 *LBI*. 从图 11 中可知: (1) Streaming 算法拥有最高的负载平衡改进值 *LBI*, 但是它的边切割改进值 *ECI* 却是最低的 (如图 10 所示), 这主要是由 Streaming 算法特性决定的. (2) 除了 Streaming 算法, 与其他算法相比, 本文提出的 ED-IDGP 算法具有更高的或者可比较的负载平衡改进值 *LBI*, 这主要是因为我们提出的方法同时考虑了边切割和负载平衡, 从而实现了边切割和负载平衡之间的自适应平衡. (3) 在数据集 LiveJournal 上, 当分区数 k 为 8 时, 本文提出的 ED-IDGP 算法的负载平衡改进值 *LBI* 分别比 CCP、IncKGGGP 高 16.68%、73.84%. (4) 在所有的数据集和分区数 k 上, ED-IDGP 算法的负载平衡改进值 *LBI* 平均比 CCP、IncKGGGP 高 19.2%、26.46%.

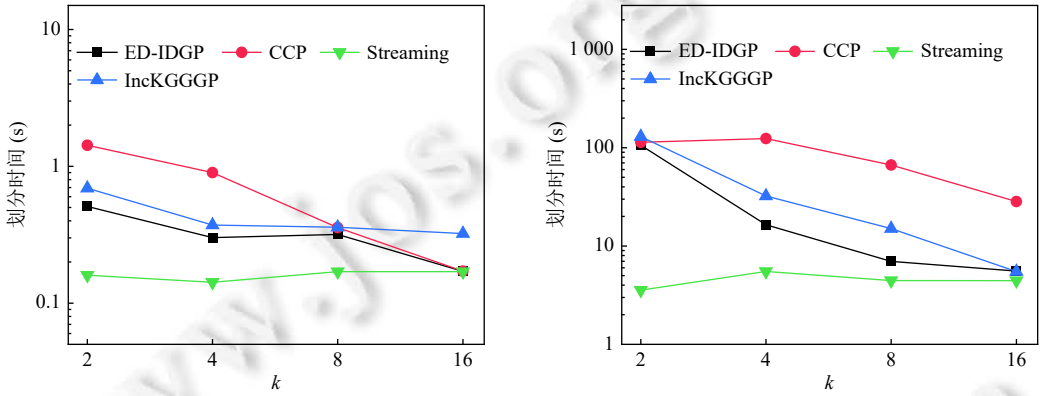
4.3.2 算法效率分析

我们使用了与第 4.3.1 节相同的实验环境. 我们在数据集 DBLP 和数据集 LiveJournal 上评估了划分动态图数据 ΔG 的时间 (如图 12 所示), 并且在数据集 Youtube 和数据集 Twitter 上评估了顶点的迁移效率 (如图 13 所示). 对于划分时间, 从图 12 中可知: (1) 总体来说, Streaming 算法用于划分动态图数据 ΔG 的时间最少, 这主要是因为 Streaming 算法是一种流式图划分算法, 其初衷是以牺牲一定的划分质量为代价, 从而大大减少划分图数据的时间. (2) 总体来说, 除了 Streaming 算法, 与其他算法相比, 本文提出的 ED-IDGP 算法拥有更低的或者可比较的划分时间.



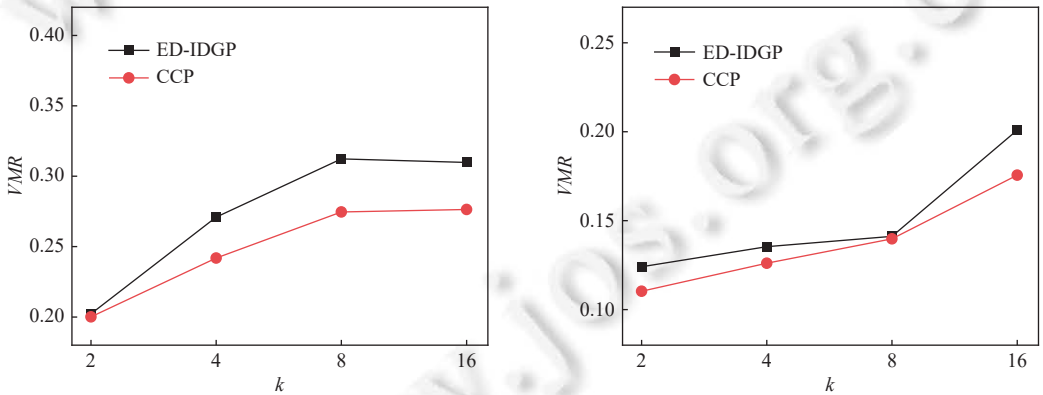
(a) 在 DBLP 上的负载平衡改进值 *LBI* (b) 在 LiveJournal 上的负载平衡改进值 *LBI*

图 11 不同的分区数量 *k* 所对应的负载平衡改进值 *LBI*



(a) 在 DBLP 上的划分时间 (b) 在 LiveJournal 上的划分时间

图 12 动态图划分时间效率评估



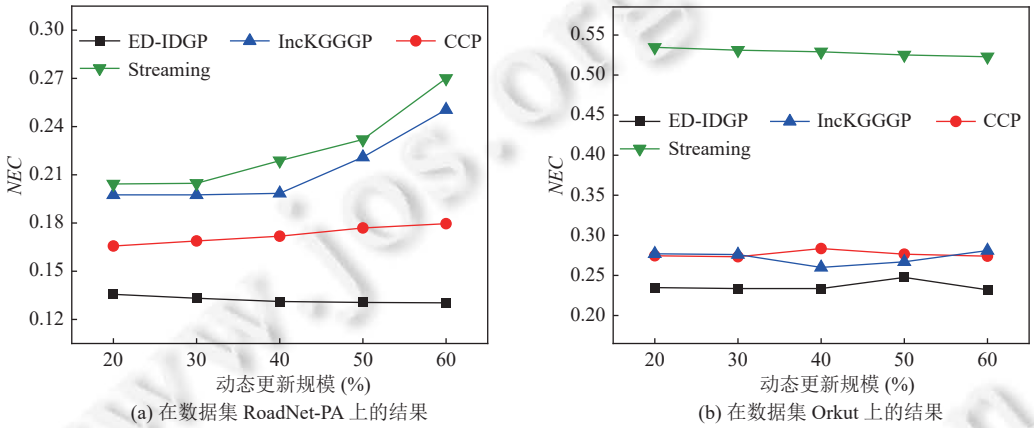
(a) 在 Youtube 上的顶点迁移率 *VMR* (b) 在 Twitter 上的顶点迁移率 *VMR*

图 13 动态图划分迁移效率评估

对于迁移效率, 从图 13 中可知: 除了在数据集 Twitter 上当分区数 *k* 为 4 时, CCP 算法有着更低或者可比较的顶点迁移率 *VMR*, 这主要是因为 CCP 算法是一种简单的增量图划分算法, 它在每次处理完单元更新后, 只在较少情况下通过移动单个顶点做局部优化, 导致划分质量不高. 而本文提出的 ED-IDGP 算法在利用动态处理器处理完每个单元更新后, 通过移动顶点组 *VG* 做局部优化, 以获得高质量的划分结果.

4.3.3 对不同动态变化规模的鲁棒性

在本实验中, 当动态更新ΔG的大小不同时, 我们讨论和分析本文提出的 ED-IDGP 算法的鲁棒性. 我们选择在数据集 RoadNet-PA 和数据集 Orkut 上做评估, 并且设置分区的数量 k 为 8. 我们设置$|\Delta G|$从 20% 到 60%, 其中固定 10% 的顶点或边的删除, 其他是顶点或边的插入. 在图 14 中展示了在数据集 RoadNet-PA 和数据集 Orkut 上, 不同的动态变化规模$|\Delta G|$所对应的标准化边切割 *NEC*. 从图 14(a) 和图 14(b) 中可知: (1) 在数据集 RoadNet-PA 和数据集 Orkut 上, Streaming 算法对于所有的动态更新都有着最高的标准化边切割 *NEC*, 这与前面的实验结果一致. (2) 在数据集 RoadNet-PA 上, 即使当动态更新规模$|\Delta G|$达到 60%, 本文提出的 ED-IDGP 算法有着最低的标准化边切割 *NEC*. 类似地, 在数据集 Orkut 上, 与其他算法相比, ED-IDGP 算法始终拥有最低的标准化边切割 *NEC*. (3) 在数据集 RoadNet-PA 上, 当$|\Delta G|$为 30% 时, ED-IDGP 算法的标准化边切割 *NEC* 分别比 CCP、IncKGGGP 和 Streaming 低 21.12%、32.57% 和 34.95%. 当$|\Delta G|$为 50% 时, ED-IDGP 算法的标准化边切割 *NEC* 分别比 CCP、IncKGGGP 和 Streaming 低 26.18%、40.92% 和 43.72%. (4) 对于所有的数据集和不同的动态更新规模$|\Delta G|$, ED-IDGP 算法的标准化边切割 *NEC* 平均比 CCP、IncKGGGP 和 Streaming 低 18.9%、25.19% 和 48.06%.



(a) 在数据集 RoadNet-PA 上的结果　　(b) 在数据集 Orkut 上的结果

图 14　不同的动态变化规模所对应的标准化边切割 *NEC*

4.3.4 分区数量的可收缩性分析

在本实验中, 我们讨论和分析当分区的数量 k 不同时, 本文提出的 IDGP 算法和 ED-IDGP 算法的可收缩性. 我们主要从划分质量和算法效率这两个方面对分区数量的可收缩性进行分析. 首先, 对于划分质量, 从第 4.3.1 节的实验结果可知: (1) 如图 9 所示, 对于所有的算法, 随着分区的数量 k 增大, 标准化边切割 *NEC* 也会随着分区的数量 k 的增大而增大. 这是合理的, 因为当分区的数量 k 增大时, 不同分区之间的通信成本会增加, 即边切割的数量会增大. 当分区数 k 不同时, 与 CCP、IncKGGGP 和 Streaming 算法相比, ED-IDGP 算法的标准化边切割 *NEC* 更低或者是可比较的. (2) 如图 11(b) 所示, 在数据集 LiveJournal 上, 当分区数 k 为 4 时, 本文提出的 ED-IDGP 算法的负载平衡改进值 *LBI* 分别比 CCP 和 IncKGGGP 高 6.51% 和 36.98%. 当分区数 k 为 16 时, 本文提出的 ED-IDGP 算法的负载平衡改进值 *LBI* 分别比 CCP 和 IncKGGGP 高 8.13% 和 30.89%. 然后, 对于算法效率, 从第 4.3.2 节的实验结果可知, 如图 12 所示, 当分区的数量 k 从 2 增大到 16 时, 本文提出的 ED-IDGP 算法的划分时间总体都呈下降趋势.

4.4 算法在分布式图计算任务中的效果分析

为了评估本文提出的 ED-IDGP 算法与其他比较方法相比在分布式图计算任务中的性能, 我们选用在图划分领域中常见的用于评估图划分算法性能的两个图计算任务网页排名计算 (PageRank) 和单源最短路径 (SSSP). 我们首先将 ED-IDGP、CCP、IncKGGGP 和 Streaming 算法得到的图发生动态变化后 $G+\Delta G$ 的划分结果 P' 作为输入导入到分布式图处理系统 Giraph 中, 然后在 Giraph 平台上运行分布式图计算任务 PageRank 和 SSSP. 在本次实

验中, 我们固定 40% 的动态变化, 其中 30% 是顶点或者边的插入, 10% 是顶点或者边的删除, 并且设置分区的数量 k 为 8. 我们分别评估了运行 PageRank 和 SSSP 的执行时间 (s) 和通信成本 (GB). 为了保持一致性, 我们将 PageRank 的迭代次数设置为 100, 并且选用拥有最大度的顶点作为 SSSP 的源顶点.

表 3 给出了 PageRank 和 SSSP 在所有数据集上的执行时间 (s) 和通信成本 (GB). 从表 3 中可知: (1) 本文提出的 ED-IDGP 算法在所有数据集上的划分结果使得 PageRank 和 SSSP 的执行时间 (s) 和通信成本 (GB) 最低, 分布式图计算任务 PageRank 和 SSSP 获得了最好的性能. (2) Streaming 算法得到的划分结果总体上获得了最差的性能, 这主要是因为 Streaming 算法用于划分图的时间开销低, 但它所获得的图划分质量较差, 所以将 PageRank 和 SSSP 运行在 Streaming 算法的划分结果上得到了最差的性能. (3) 对于分布式图计算任务 PageRank, 在数据集 RoadNet-PA 上, ED-IDGP 算法的执行时间分别比 CCP、IncKGGGP 和 Streaming 低 13.03%、33.25% 和 50.38%. ED-IDGP 算法的通信成本分别比 CCP、IncKGGGP 和 Streaming 低 11.85%、34.25% 和 40.8%. (4) 对于分布式图计算任务 SSSP, 在数据集 Orkut 上, ED-IDGP 算法的执行时间分别比 CCP、IncKGGGP 和 Streaming 低 7.41%、11.11% 和 29.82%. ED-IDGP 算法的通信成本分别比 CCP、IncKGGGP 和 Streaming 低 3.02%、4.89% 和 52.55%. (5) 对于分布式图计算任务 PageRank, 对于所有的数据集, ED-IDGP 算法的执行时间平均比 CCP、IncKGGGP 和 Streaming 低 13.06%、17.42% 和 36.09%. ED-IDGP 算法的通信成本平均比 CCP、IncKGGGP 和 Streaming 低 13.58%、17.59% 和 47.23%. (6) 对于分布式图计算任务 SSSP, 对于所有的数据集, ED-IDGP 算法的执行时间平均比 CCP、IncKGGGP 和 Streaming 低 11.25%、17.51% 和 37.42%. ED-IDGP 算法的通信成本平均比 CCP、IncKGGGP 和 Streaming 低 17.49%、25.39% 和 55.27%.

表 3 图计算任务 PageRank 和 SSSP 的执行时间和通信成本

图计算任务	数据集	度量指标	图发生动态变化后$G+\Delta G$的划分 P'			
			Streaming	IncKGGGP	CCP	ED-IDGP
PageRank	DBLP	执行时间 (s)	34.2	24.0	22.1	19.1
		通信成本 (GB)	0.262	0.160	0.152	0.139
	RoadNet-PA	执行时间 (s)	65.9	49.0	37.6	32.7
		通信成本 (GB)	0.201	0.181	0.135	0.119
	Youtube	执行时间 (s)	78.6	55.5	55.4	46.7
		通信成本 (GB)	0.802	0.562	0.561	0.421
	LiveJournal	执行时间 (s)	843.1	754.8	721.9	683.9
		通信成本 (GB)	8.789	5.078	5.099	4.767
	Orkut	执行时间 (s)	2 102.2	1 910.1	1 812.1	1 775.8
		通信成本 (GB)	3.678	1.915	1.876	1.868
	Twitter	执行时间 (s)	82 891.5	53 956.5	61 678.9	43 919.5
		通信成本 (GB)	288.971	179.621	191.234	135.441
SSSP	DBLP	执行时间 (s)	4.3	3.1	2.7	2.6
		通信成本 (GB)	0.009 3	0.005 6	0.003 9	0.003 4
	RoadNet-PA	执行时间 (s)	11.2	9.2	8.1	6.3
		通信成本(GB)	0.007 2	0.006 2	0.004 3	0.001 8
	Youtube	执行时间 (s)	14.9	12.6	12.5	10.1
		通信成本 (GB)	0.026 9	0.018 1	0.017 9	0.013 9
	LiveJournal	执行时间 (s)	112.8	89.8	85.2	81.6
		通信成本 (GB)	0.283 1	0.160 9	0.154 3	0.149 8
	Orkut	执行时间 (s)	243.8	192.5	184.8	171.1
		通信成本(GB)	1.321 9	0.659 5	0.646 7	0.627 2
	Twitter	执行时间 (s)	4 567.9	2 688.1	2 477.9	2 212.3
		通信成本 (GB)	9.100	5.383 9	5.288 6	4.987 7

5 总 结

本文提出了基于顶点组重分配的动态增量图划分算法 ED-IDGP 来解决现实中持续增长的动态图划分问题, 设计了处理实时动态图中顶点插入、边插入、顶点删除和边删除这 4 种不同的动态变化类型的动态处理器, 并在每次处理完单元更新后通过在分区发生动态变化的附近执行提出的局部优化器进一步提高图划分的质量. 本文在真实数据集上将 ED-IDGP 算法与现有的先进动态图划分算法从不同的维度和不同的度量指标进行了充分的对比. 实验结果表明, ED-IDGP 算法的标准化边切割、边切割改进值和负载平衡改进值分别平均比现有的动态图划分算法低 27.81%、26.34% 和 24.04%. 本文还将各个算法获得的动态图划分结果导入到分布式图处理系统 Giraph 中, 通过运行分布式图计算任务评估了 ED-IDGP 算法的性能. ED-IDGP 算法的执行时间和通信成本分别平均比现有的动态图划分算法低 20.96% 和 27.44%.

References:

[1] Malewicz G, Austern MH, Bik AJC, Dehnert JC, Horn I, Leiser N, Czajkowski G. Pregel: A system for large-scale graph processing. In: Proc. of the 2010 ACM SIGMOD Int'l Conf. on Management of Data. Indianapolis: ACM, 2010. 135–146. [doi: 10.1145/1807167.1807184]

[2] Giraph. 2021. https://giraph.apache.org/

[3] Low Y, Bickson D, Gonzalez J, Guestrin C, Kyrola A, Hellerstein JM. Distributed GraphLab: A framework for machine learning and data mining in the cloud. Proc. of the VLDB Endowment, 2012, 5(8): 716–727. [doi: 10.14778/2212351.2212354]

[4] Chen R, Shi JX, Chen YZ, Zang BY, Guan HB, Chen HB. PowerLyra: Differentiated graph computation and partitioning on skewed graphs. ACM Trans. on Parallel Computing, 2018, 5(3): 13. [doi: 10.1145/3298989]

[5] Cui PJ, Yuan Y, Li CH, Zhang C, Wang GR. RGraph: Effective distributed graph data processing system based on RDMA. Ruan Jian Xue Bao/Journal of Software, 2022, 33(3): 1018–1042 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6449.htm [doi: 10.13328/j.cnki.jos.006449]

[6] Neo4j. 2022. https://neo4j.com/

[7] Sarwat M, Elnikety S, He YX, Kliot G. Horton: Online query execution engine for large distributed graphs. In: Proc. of the 28th IEEE Int'l Conf. on Data Engineering. Arlington: IEEE, 2012. 1289–1292. [doi: 10.1109/ICDE.2012.129]

[8] Kang U, Tong HH, Sun JM, Lin CY, Faloutsos C. Gbase: A scalable and general graph management system. In: Proc. of the 17th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. San Diego: Association for Computing Machinery, 2011. 1091–1099. [doi: 10.1145/2020408.2020580]

[9] Andreev K, Racke H. Balanced graph partitioning. Theory of Computing Systems, 2006, 39(6): 929–939. [doi: 10.1007/s00224-006-1350-7]

[10] Hendrickson B, Leland R. An improved spectral graph partitioning algorithm for mapping parallel computations. SIAM Journal on Scientific Computing, 1995, 16(2): 452–469. [doi: 10.1137/0916028]

[11] Gilbert JR, Miller GL, Teng SH. Geometric mesh partitioning: Implementation and experiments. SIAM Journal on Scientific Computing, 1998, 19(6): 2091–2110. [doi: 10.1137/S1064827594275339]

[12] Hager WW, Phan DT, Zhang HC. An exact algorithm for graph partitioning. Mathematical Programming, 2013, 137(1–2): 531–556. [doi: 10.1007/s10107-011-0503-x]

[13] Hendrickson B, Leland R. A multi-level algorithm for partitioning graphs. In: Proc. of the 1995 ACM/IEEE Conf. on Supercomputing. San Diego: IEEE, 1995. 28.

[14] Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing, 1998, 20(1): 359–392. [doi: 10.1137/S1064827595287997]

[15] Chevalier C, Pellegrini F. PT-Scotch: A tool for efficient parallel graph ordering. Parallel Computing, 2008, 34(6–8): 318–331. [doi: 10.1016/j.parco.2007.12.001]

[16] Kernighan BW, Lin S. An efficient heuristic procedure for partitioning graphs. The Bell System Technical Journal, 1970, 49(2): 291–307. [doi: 10.1002/j.1538-7305.1970.tb01770.x]

[17] Fiduccia CM, Mattheyses RM. A linear-time heuristic for improving network partitions. In: Proc. of the 19th Design Automation Conf. Las Vegas: IEEE, 1982. 175–181. [doi: 10.1109/DAC.1982.1585498]

[18] Stanton I, Kliot G. Streaming graph partitioning for large distributed graphs. In: Proc. of the 18th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Beijing: ACM, 2012. 1222–1230. [doi: 10.1145/2339530.2339722]

[19] Tsourakakis C, Gkantsidis C, Radunovic B, Vojnovic M. FENNEL: Streaming graph partitioning for massive scale graphs. In: Proc. of the 7th ACM Int'l Conf. on Web Search and Data Mining. New York: ACM, 2014. 333–342. [doi: 10.1145/2556195.2556213]

[20] Zhang W, Chen Y, Dai D. AKIN: A streaming graph partitioning algorithm for distributed graph storage systems. In: Proc. of the 18th IEEE/ACM Int'l Symp. on Cluster, Cloud and Grid Computing. Washington: IEEE, 2018. 183–192. [doi: 10.1109/CCGRID.2018.00033]

[21] LI Q, LI HX, Zhong J, Ying CT, LI Q. Research on graph partitioning in heterogeneous computing environment. Chinese Journal of Computers, 2021, 44(8): 1751–1766 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2021.01751]

[22] Firth H, Missier P, Aiston J. Loom: Query-aware partitioning of online graphs. In: Proc. of the 21st Int'l Conf. on Extending Database Technology. Vienna: EDBT, 2018. 337–348.

[23] Schloegel K, Karypis G, Kumar V. Dynamic repartitioning of adaptively refined meshes. In: Proc. of the 1998 ACM/IEEE Conf. on Supercomputing. Orlando: IEEE, 1998. 29. [doi: 10.1109/SC.1998.10025]

[24] Vaquero LM, Cuadrado F, Logothetis D, Martella C. Adaptive partitioning for large-scale dynamic graphs. In: Proc. of the 34th Int'l Conf. on Distributed Computing Systems. Madrid: IEEE, 2014. 144–153. [doi: 10.1109/ICDCS.2014.23]

[25] Nicoara D, Kamali S, Daudjee K, Chen L. Hermes: Dynamic partitioning for distributed social network graph databases. In: Proc. of the 18th Int'l Conf. on Extending Database Technology. Brussels: EDBT, 2015. 25–36.

[26] Li H, Yuan H, Huang JB, Cui JT, Yoo J. Dynamic graph repartitioning: From single vertex to vertex group. In: Proc. of the 25th Int'l Conf. on Database Systems for Advanced Applications. Jeju: Springer, 2020. 482–497. [doi: 10.1007/978-3-030-59416-9_29]

[27] Ou CW, Ranka S. Parallel incremental graph partitioning using linear programming. In: Proc. of the 1994 ACM/IEEE Conf. on Supercomputing. Washington: IEEE, 1994. 458–467. [doi: 10.1109/SUPERC.1994.344309]

[28] Fan WF, Liu MY, Tian C, Xu RQ, Zhou JR. Incrementalization of graph partitioning algorithms. Proc. of the VLDB Endowment, 2020, 13(8): 1261–1274. [doi: 10.14778/3389133.3389142]

[29] Huang JW, Abadi DJ. Leopard: Lightweight edge-oriented partitioning and replication for dynamic graphs. Proc. of the VLDB Endowment, 2016, 9(7): 540–551. [doi: 10.14778/2904483.2904486]

[30] Dai D, Zhang W, Chen Y. IOGP: An incremental online graph partitioning algorithm for distributed graph databases. In: Proc. of the 26th Int'l Symp. on High-performance Parallel and Distributed Computing. Washington: ACM, 2017. 219–230. [doi: 10.1145/3078597.3078606]

[31] Rahimian F, Payberah AH, Girdzijauskas S, Jelasity M, Haridi S. JA-BE-JA: A distributed algorithm for balanced graph partitioning. In: Proc. of the 7th IEEE Int'l Conf. on Self-adaptive and Self-organizing Systems. Philadelphia: IEEE, 2013. 51–60. [doi: 10.1109/SASO.2013.13]

[32] Wang L, Xiao YH, Shao B, Wang HX. How to partition a billion-node graph. In: Proc. of the 30th IEEE Int'l Conf. on Data Engineering. Chicago: IEEE, 2014. 568–579. [doi: 10.1109/ICDE.2014.6816682]

[33] Zheng AG, Labrinidis A, Chrysanthis PK. Planar: Parallel lightweight architecture-aware adaptive graph repartitioning. In: Proc. of the 32nd IEEE Int'l Conf. on Data Engineering. Helsinki: IEEE, 2016. 121–132. [doi: 10.1109/ICDE.2016.7498234]

[34] Shang ZC, Yu JX. Catch the wind: Graph workload balancing on cloud. In: Proc. of the 29th IEEE Int'l Conf. on Data Engineering. Brisbane: IEEE, 2013. 553–564. [doi: 10.1109/ICDE.2013.6544855]

[35] Xu N, Chen L, Cui B. LogGP: A log-based dynamic graph partitioning method. Proc. of the VLDB Endowment, 2014, 7(14): 1917–1928. [doi: 10.14778/2733085.2733097]

[36] Predari M, Esnard A. A k-way greedy graph partitioning with initial fixed vertices for parallel applications. In: Proc. of the 24th Euromicro Int'l Conf. on Parallel, Distributed, and Network-based Processing. Heraklion: IEEE, 2016. 280–287. [doi: 10.1109/PDP.2016.109]

[37] Abdolrashidi A, Ramaswamy L. Continual and cost-effective partitioning of dynamic graphs for optimizing big graph processing systems. In: Proc. of the 2016 IEEE Int'l Congress on Big Data. San Francisco: IEEE, 2016. 18–25. [doi: 10.1109/BigDataCongress.2016.12]

[38] Pang S, Chen CJ, Wei T. A realtime community detection algorithm: Incremental label propagation. In: Proc. of the 1st Int'l Conf. on Future Information Networks. Beijing: IEEE, 2009. 313–317. [doi: 10.1109/ICFIN.2009.5339592]

[39] Raghavan NU, Albert R, Kumara S. Near linear time algorithm to detect community structures in large-scale networks. Physical Review E, 2007, 76(3): 036106. [doi: 10.1103/PhysRevE.76.036106]

[40] Stanford large network dataset collection. 2014. http://snap.stanford.edu/data/index.html

[41] Rossi RA, Ahmed NK. The network data repository with interactive graph analytics and visualization. In: Proc. of the 29th AAAI Conf. on Artificial Intelligence. Austin: AAAI Press, 2015. 4292–4293. [doi: 10.1609/aaai.v29i1.9277]

附中文参考文献:

[5] 崔鹏杰, 袁野, 李岑浩, 张灿, 王国仁. RGraph: 基于RDMA的高效分布式图数据处理系统. 软件学报, 2022, 33(3): 1018–1042. http://www.jos.org.cn/1000-9825/6449.htm [doi: 10.13328/j.cnki.jos.006449]

[21] 李琪, 李虎雄, 钟将, 英昌甜, 李青. 异构计算环境中图划分算法的研究. 计算机学报, 2021, 44(8): 1751–1766. [doi: 10.11897/SP.J.1016.2021.01751]

李贺(1983－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为图数据管理, 图数据挖掘, 深度学习.

黄健斌(1975－), 男, 博士, 教授, CCF 专业会员, 主要研究领域为数据挖掘, 智慧交通, 人工智能.

刘延娜(1996－), 女, 硕士, 主要研究领域为大图分割, 分布式计算, 图数据挖掘.

乔少杰(1981－), 男, 博士, 教授, CCF 杰出会员, 主要研究领域为数据库, 人工智能, 数据挖掘.

杨舒琪(2001－), 女, 硕士生, 主要研究领域为大图分割, 分布式计算, 深度学习.