综述

大数据下图三角计算的研究进展

金宏桥,董一鸿

(宁波大学信息科学与工程学院,浙江 宁波 315211)

摘　要:图三角数量的计算是计算网络聚集系数和传递性的重要步骤,广泛应用于重要角色识别、垃圾邮件检测、社区发现、生物检测等。在大数据背景下,计算图中三角形算法主要面临时空消耗和计算准确性两大难题。介绍了代表性的大图中计算三角形的算法,主要存在准确计算和近似计算两大类。准确计算算法又分为内存算法、外存算法和分布式算法,时空消耗或 I/O 消耗很大。近似计算算法中,有辅助算法、非流式算法和流式算法之分。最后对计算三角形算法进行了归纳总结。

关键词:准确计算;近似计算;三角形;图

中图分类号:TP391　　**文献标识码**:A

doi: 10.11959/j.issn.1000-0801.2016169

Research progress of triangle counting in big data

JIN Hongqiao, DONG Yihong

Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China

Abstract: Counting triangles in a graph is an important step to calculate the clustering coefficient and the transitivity ratio of the network, which is widely used in important role identification, spam detection, community discovery, biological detection etc. Counting triangles algorithm is mainly faced with two major problems of space-time consumption and accuracy. The representative algorithm of the counting triangles in the big graph was introduced. There existed two kinds of algorithms, which were exact counting algorithm and approximate counting algorithm. Exact counting algorithms were divided into internal memory algorithm, external memory algorithm and distributed algorithm. The space-time consumption or I/O consumption of exact counting algorithm was very large. Approximate counting algorithms were divided into auxiliary algorithm, static algorithm and streaming algorithm. In the end, the counting triangles algorithms were summarized.

Key words: exact counting, approximate counting, triangle, graph

1　引言

随着网络技术和社会网络服务的发展,网络中的数据量和信息量越来越大。在这种大数据的环境下,对数据的分析和挖掘显得尤为重要。近年来,对有大规模数据的网络的分析得到越来越多的关注。计算机学科的数据结构图可以作为很多种网络的模型,如万维网、P2P 网络和社交网络等都可以用含有特定信息的图作为它们的模型。对网络的分析逐渐转化为对保存网络重要信息的图的分析。由于网络中的关系和个体数量非常多,所以作为其模型的图的规模也很大。

社会网络的同质性和传递性产生了对图中三角形的

收稿日期:2016-04-05;**修回日期**:2016-06-12

研究。图中的三角形是复杂网络分析的重要角色，不论是来自社会交互、计算机交流、金融交易、蛋白质还是生态学网络，其中三角形的数量都是巨大的，它在这些领域中有着非常广泛的应用。通过三角形的分布可以区分哪些是垃圾邮件的主人。角色行为识别中，通过使用者参与的三角形的数量可以判断这个使用者的地位。生物信息学中的主题检测需要计算三元组的频率。三角形巨大的数量可以与蛋白质交互网络的拓扑结构和功能性相联系。三角形各点度数之间的关系也可以作为基础图的描述符，在数据库中，三角形也有具体的应用。

目前图中三角形的计算主要分为准确计算(exact counting)和近似计算(approximate counting)两种类型。准确计算可以准确地计算图中三角形的数量，对于大图来说，规模很大，所以计算的时空消耗很大，外存算法的I/O消耗也会很大。研究人员的重点就是在保持准确计算的情况下，减少时空消耗和I/O次数。近年来，分布式框架MapReduce的出现，也使很多研究人员研究此框架下的计算三角形算法。相较而言，近似计算比准确计算的实际应用更广泛，同时空间消耗较少。在保持一定准确度的情况下，研究人员对近似计算三角形数量更感兴趣。对于近似计算，大部分的研究者将重点放在采样上，通过一定的采样方法证实，采样得到的三角形数量与实际数量的差值很小，同时将时间空间的复杂度降低。目前，采样方法很多，各有所长。准确计算三角形算法的部分关键点也可以用在近似计算算法上。如上文所述，随着互联网的快速发展，大规模的图不断涌现，在图流中使用限制的内存来估算三角形的数量，这个问题的研究意义越来越显著，同时难度也越来越大。

2 图三角的基本概念

定义1 (三角形)给定一个图$G=(V,E)$，它包含了一个顶点集合V、一个边的集合E，$|V|$和$|E|$分别表示顶点和边的个数。如果顶点$u、v、w\in V$，边$\{u,v\}、\{v,w\}、\{w,u\}\in E$，那么3个顶点和3条边组成一个三角形，称为$\Delta_{uvw}$。

定义2 (三角形的实际数量、准确数量和估算数量)用T表示图G中三角形的实际数量，用t表示通过准确算法得出的图G中三角形的数量，用T'表示通过估计算法得出的图G中三角形的数量。

定义3 (度和邻域)顶点v的邻域$\Gamma(v)$表示所有与顶点v相邻的点，满足$\Gamma(v)=\{u\in V:(u,v)\in E\}$。顶点$v$的度$d(v)$是顶点$v$连接的所有边的个数或者是它的邻域，满足$d(v)=|\Gamma(v)|$。图$G$最大的度$d_{\max}(G)$是指图中顶点最大的度，即$d_{\max}(G)=\max\{d(v):v\in V\}$。$m、n$分别表示图中边、点的个数，顶点的度和图中的边数满足$\sum_{v\in V}d(v)=2m$。入度和出度是对于有向图来说的，一个顶点的入度等于被所有箭头所指的数量，一个顶点的出度等于被所有箭尾所指的数量。

定义4 ((1+ε)-approximation)返回q的以ε为因子的估计值q'，当q满足$(1-\varepsilon)q\leqslant q'\leqslant(1+\varepsilon)q$，其中，$\varepsilon<0$。

定义5 ((ε, δ)-approximation)返回q的以$\varepsilon、\delta$为因子的估计值q'，当q至少以$1-\delta$的可能性满足，$(1-\varepsilon)q\leqslant q'\leqslant(1+\varepsilon)q$，其中，$\varepsilon<0,\delta<1$。

定义6 (图流)图中的边是以一串流的形式加入的。

3 图三角的准确计算

图三角的计算有准确计算和近似计算两种类型。准确计算图三角算法目前有内存算法、外存算法、分布式算法3类。内存算法指当内存能容纳整个图，可以在内存中将图中的三角形计算出来的算法；外存算法是指当图的规模很大，不能全部存入内存进行计算时，通过一定的策略将图分为几个部分存入内存进行计算，这种算法会产生一定数量的I/O操作；分布式算法是指用分布式框架来计算图中三角形数量，目前主要采用MapReduce框架。内存算法中除了具有代表性的Node-iterator[1]和Edge-iterator算法，还有Matrix-multiplication[2]算法。基于点边迭代和矩阵相乘也出现了一系列改进的算法，有AYZ[3]、Node-iterator-core[4]、Forward[5]、Forward-hashed[6]、Compact-forward[7]。时隔多年又出现了结合点边迭代的Combined Iterator[8]算法。本文将介绍两个外存算法：Chu和Cheng[9]提出的基于图划分的算法和Hu[10]提出的有效I/O的算法。近些年来，随着分布式框架的应用，出现了基于MapReduce框架的GP[11]、TTP[12]以及CTTP[13]算法。目前出现的准确计算图三角算法都应用在静态图上。以下将一一介绍。

3.1 内存算法

内存算法最初适用于规模比较小的图，小图完全可以存入计算机内存中，并进行计算。内存算法最先找到了如何使用计算机解决三角形计算问题的方法，并且不断地在算法细节中得到突破，以减少运行时空成本。其中，基本迭代算法简单，但成本高，不适合规模大的图。Fast Common Neighbor Iteration算法通过混合线性扫描二分法和分段索

引法，在时间上进行了很大优化，同时增加了空间的开销。随着图规模的增长，内存算法需要更大内存的计算机。

3.1.1　迭代算法

Node-iterator（点迭代）算法检测每一个顶点的每一对邻点之间是否有一条边，如果有，那么就得到一个三角形，反之得不到。为了使每一个三角形只被计算一次，需要安排每个顶点的顺序。Edge-iterator（边迭代）算法迭代所有边，比较每条边两个顶点的领域，对于一条边$\{u,w\}$，仅当 v 同时出现在 $\Gamma(u)$和 $\Gamma(w)$中，3 点$\{u,v,w\}$才组成一个三角形。

Alon、Yuster 和 Zwick 将点迭代和矩阵相乘结合在一起，得到 AYZ 算法。算法将点集分为度数低的顶点集合 $V_{low}=\{v\in V:d(v)\leqslant\beta\}$和度数高的顶点集合 $V_{high}=V/V_{low}$，其中，$\beta=m^{\gamma-1/\gamma+1}$，$\gamma$ 是矩阵乘法指数。低度数的点集采用标准点迭代方法，高度数点集采用快速矩阵乘法。Node-iterator-core 算法是在点迭代的基础上，在每次选择顶点迭代时选择当前度数最小的顶点，当此顶点所在的三角形全部被计算后，将此顶点删除。Forward 算法是边迭代算法的改进。在边迭代算法中，需要将边两个顶点的所有邻接顶点都比较一下，在 Forward 算法中只需要比较边迭代中所有邻接顶点的子集 A。在这个算法中，由于所有的点都是有顺序的，所以可以将图视为有向图，方便理解算法过程。$A(v)$的数据结构的大小是不大于点 v 的入度。Latapy M 提出了一个对 Forward 改进的算法 Compact-forward。Compact-forward 使用迭代器迭代邻接点的子集，迭代方法和边迭代方法相同。邻接点是排序过的，比较语句也在一个可达的确定指数前停止。虽然时间上界和 Forward 算法相同，但是它不需要额外的数组，因此节省了时间和空间。

Oracle Labs 的 Sevenich M 提出了 FCNI（fast common neighbor iteration）算法。该算法的基准算法是结合点迭代、边迭代两种迭代算法的 Combined Iterator 算法，该基准与前面提到的 Forward 算法类似，多了邻点选择时的排序。FCNI 算法指出 Combined Iterator 算法的主要部分是重复求出不同顶点对的共同邻点，所以想要快速计算三角形的个数，就需要快速求出顶点对的共同邻点。面临的问题是：求一个度很高的顶点和其他顶点的共同邻点需要很长的时间。于是他们提出了两个方法来解决这个问题：混合线性扫描二分法和分段索引法。每个顶点的邻点存储在有序的邻接数组中，问题转化为求两个有序数组的共有元素。当两个数组长度差距过大时，采用二分法，算法从选择长数组的中间元素和用二分查找短数组开始。当两个数组度都较小时，采用线性扫描。算法又将度数高的顶点构建了索引，对他们的邻点数组构造了分段索引。这两个方法的应用使该算法的运行时间大大减少，在当前内存算法中运行时间最少。

3.1.2　矩阵相乘算法

在矩阵相乘（matrix-multiplication）算法中，假设 A 是图 G 的邻接矩阵，A^3 对角线上的数字分别代表对应顶点所在三角形个数的两倍。A^3 对角线上的数字之和代表图 G 中三角形个数的 6 倍，因为三角形由 3 个顶点组成，一个三角形会被重复计算 3×2 次。这会导致算法运行时间达到 $O(n^3)$。通过快速矩阵相乘的方法可以使运行时间下降。

3.2　外存算法

当图的规模过大或者计算机的内存不足以装下整个图时，采用外存算法是基本策略。外存算法需要解决的问题是保证计算的三角形个数的准确性，同时减少 I/O 操作。Chu、Cheng 算法和 MGT 算法都是将图的信息分段载入内存进行计算。不同的是，分别用不同的方法保证图中三角形不被拆开。前者划分子图时保留了所有顶点的邻点，后者在有向化图之后载入子图和顶点的邻接表。表 1 是在 I/O 操作和运行时间方面，对 Chu、Cheng 算法 DGP、RGP 和算法 MGT 在内存使用率为 25%时，各算法在不同数据集上的效果比较，其中，使用的机器是 3 GHz CPU 和 8 GB 的内存。MGT 算法在 I/O 操作数和运行时间上都优于 Chu、Cheng 算法。

3.2.1　基于图划分的算法

Chu 和 Cheng 提出了基于图划分的外存图三角计算算法，该算法首先将整个图划分成几个子图后，存在外存，子图的规模小，就可以放入内存。然后依次将每个子图调

表 1　DGP、RGP 和 MGT 的比较

数据集	I/O 操作数			运行时间/s		
	DGP	RGP	MGT	DGP	RGP	MGT
LJ	1 053 594	1 257 411	307 893	109.562	156.757	19.053 35
BTC	12 683 280	16 539 188	4 192 628	1 009.11	1 500.01	111.044 9
USRD	831 306	2 141 150	414 240	62.207 3	189.318	6.688 45

入内存,计算当前子图中的三角形数量。为了保证图划分后不会将三角形拆开,每个子图实际上也保留了当前部分中所有顶点的邻点,如图 1、图 2 所示。根据划分图的方法,Chu 和 Cheng 有两种方法:DGP (deterministic graph partition)和 RGP(randomized graph partitioning)。前者采用确定方法划分子图,后者采用随机方法划分子图。划分子图是一个难题,如果某部分子图的邻点很多,会使算法的时空效率变高。

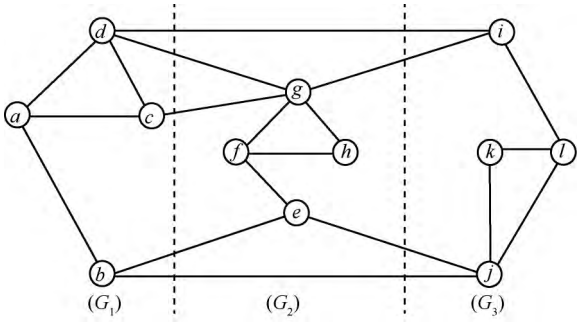


图 1 基于图划分算法的原图

3.2.2 有效 I/O 算法

Hu X C 和 Tao Y F 设计了一种有效 I/O 算法,计算三角形算法 MGT(massive graph triangulation),这是针对静态的图。这个算法与 Chu 的算法有明确的不同。他们将无向图以有向图的形式表现出来,如图 3 所示。有向图的有向边是根据无向图顶点的度和编号设置指向的。例如,当顶点 a 的度小于 b 的度或者 a 的度等于 b 的度但 a 的编号小于 b 的编号,定义 $a \prec b$,此时无向边$\{a,b\}$在有向化后是 a 指向 b。无向三角形 Δ_{uvw} 变为有向三角形Δ^{*}_{uvw},当 $u \prec v \prec w$,其中,顶点 u 被称为(cone vertex 锥顶点),边$\{v,w\}$被称为(pivot edge 中枢边),如图 4 所示。算法的准备工作是将无向图根据规则进行有向化,并以邻接表的形式存储,每一个顶点的邻接表只存它的出—邻居(即它指向的顶点)。MGT 算法是逐步将有向图中的边载入内存,根据需要从外存载入关联的顶点,计算出三角形的数量。具体过程是:

- 将有向图中的一部分边载入内存;
- 得出当前内存中的边所在的顶点;
- 对于每一个顶点 v,从外存中得到它的邻接表(出—邻接表),将其出—邻居与内存中的顶点做交集。将顶点 v 到交集得到的顶点所成的边与当前载入内存中的有向图的边做并集,找出其中以 u 为锥顶点的三角形个数,并释放相应的空间。

MGT 算法可以正确地找出所有的三角形,因为第一步保证了每一条有向图的边都可以在一个独特的迭代中载入内存。同时第二步保证了可以找出以当前载入内存的边为中枢边的三角形。MGT 算法在 I/O 和 CPU 上都非常高效。

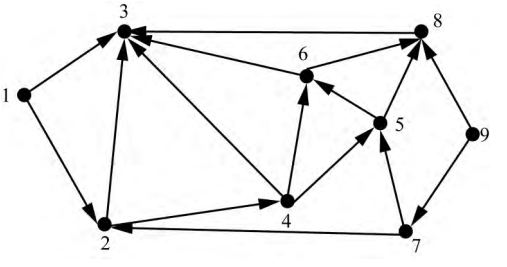


图 3 无向图变为有向图

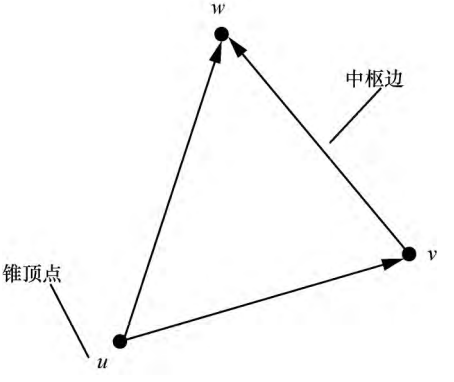


图 4 有向三角形

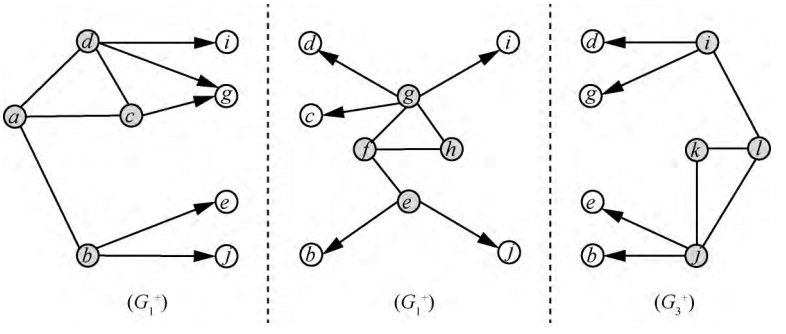


图 2 基于图划分算法划分后的图

3.3 分布式算法

由于分布式算法的普及，一些研究使用 MapReduce 算法计算图三角。目前使用 MapReduce 框架的算法是基于图划分来分析问题的。在某些方面加快了计算算法，但同时也出现了一些问题。接下来介绍 3 种基于 MapReduce 框架的算法。这 3 个算法有一个共同的基础，是图的分割。图的分割算法是先将顶点均分为 p 个部分(partition)，$V=V_1\cup V_2\cup\cdots\cup Vp$，其中，当 $i\neq j$ 时，$V_i\cap V_j=\Phi$。同时定义 $V_{ijk}=V_i\cup V_j\cup V_k$，$E_{ijk}=\{(u,w)\in E:u,w\in V_{ijk}\}$，$G_{ijk}=(V_{ijk},E_{ijk})$。$G_{ijk}$ 叫做 3-partition，G_{ij} 是 2-partition，G_i 是 1-partition，G_{ijk} 和 G_{ij} 的含义如图 5、图 6、图 7 所示。GP 算法是三角形计算在 MapReduce 上的初次应用，GP 算法有很多冗余计算。TTP 算法发现 GP 算法有很多冗余计算，原因是在 map 阶段输出了重复的边。为了避免 GP 算法冗余计算的产生，TTP 算法定义了 3 个类型的三角形。CTTP(colored triangle type partition)算法是针对 GP 算法的“curse of the last reducer”而被提出的，避免了不均衡。表 2 是在时间和每轮 MapReduce 中数据 shuffle 大小上，对 GP、TTP、CTTP 算法的比较。运行平台是 Hadoop，集群由 40 台机器组成，每台机器的内存为 4 GB。由表 2 可见，CTTP 算法在时间和空间上都优于 GP、TTP 算法。

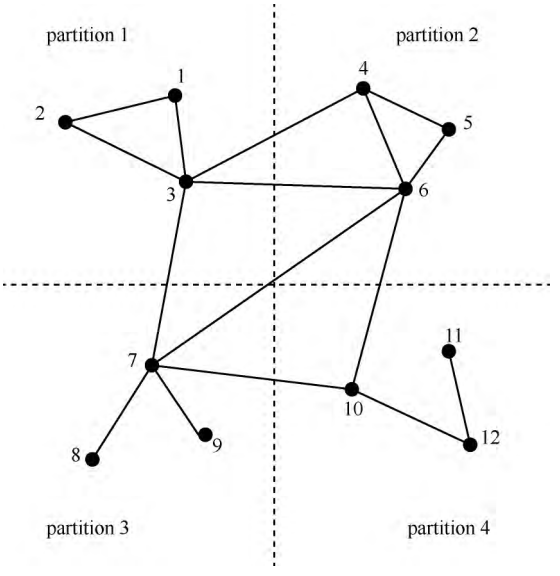


图 5　顶点平均分为 p 个部分(p=4)

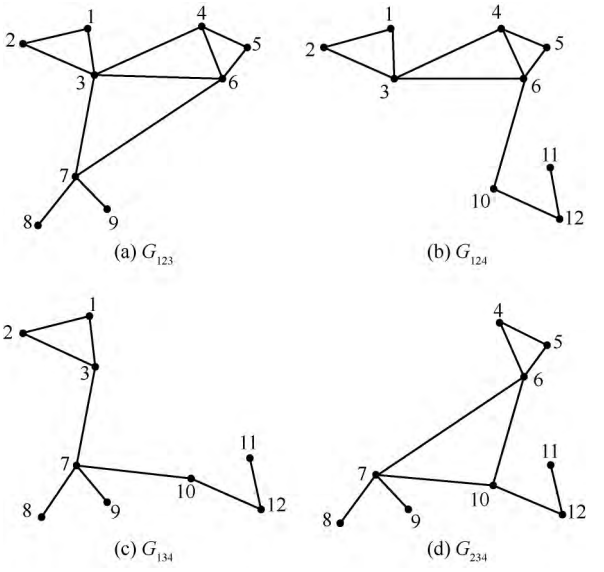


图 6　3-partition G_{ijk}

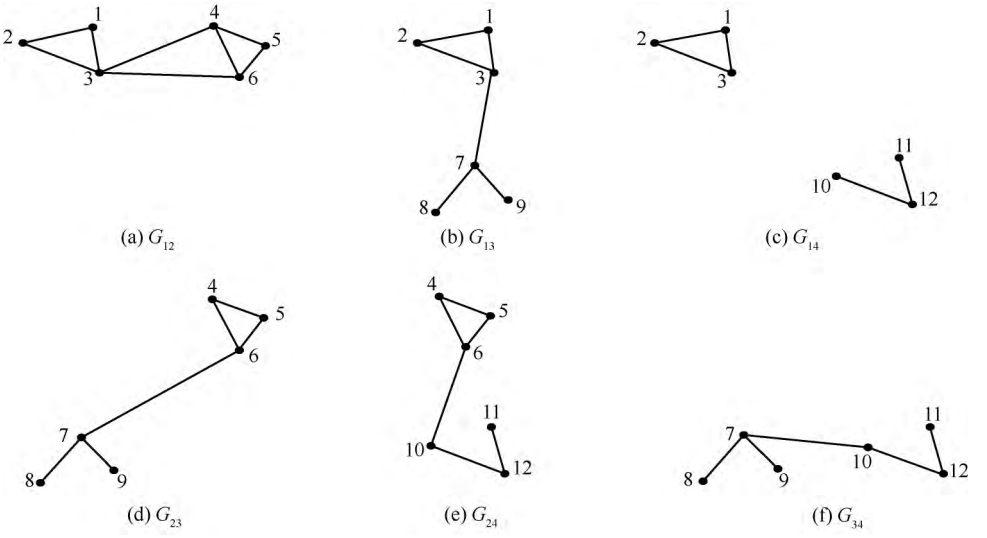


图 7　2-partition G_{ij}

表 2 GP、TTP、CTTP 算法的比较

数据集	时间/min			每轮 MapReduce 中数据 shuffle 的大小/GB		
	GP	TTP	CTTP	GP	TTP	CTTP
LJ	1	1	1	24.4	4.6	4.5
PhoneCall	11	6	6	104	79	69.5
Twitter	302	162	92	2 140	1 300	31.5
SubDomain	1 496	455	218	4 038.4	2 280.6	56.8

3.3.1 GP 算法

Suri 和 Vassilvitskii 使用 MapReduce 框架提出了 GP(graph partition)算法。算法的第一步是将图中的顶点划分为 p 个部分,$G_i=(V_i,E_i)$,其中,$0<i\leqslant p$,所以每个部分含有几乎相同数目的顶点。GP 算法使用内存算法计算每一个 3-partition G_{ijk} 中三角形的数目,其中,$0<i<j<k\leqslant p$。最后根据三角形的顶点是否被分到同一个部分,将结果整合起来,得出最终的结果。如果三角形的 3 个顶点都出现在同一个部分中,那么对于 3-partition G_{ijk},此三角形便被计算了 3 次。

3.3.2 TTP 算法

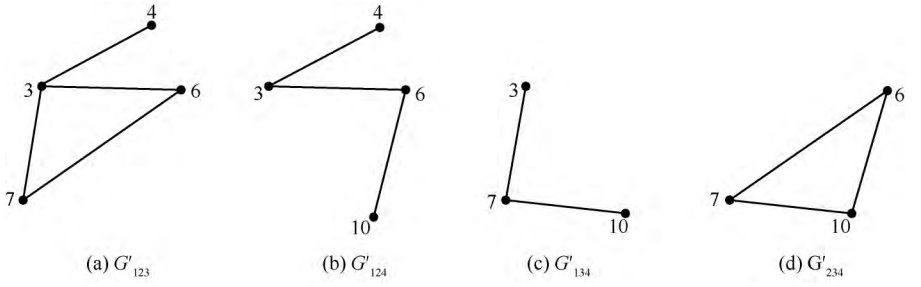
Park 和 Chung 针对 GP 算法的不足提出了 TTP(triangle type partition)算法。Park 和 Chung 认为 GP 算法有很多冗余计算,比如上面提到一个三角形可能被计算多次。他们发现三角形被计算多次的原因是,在 map 阶段输出了重复的边。为了避免这个情况的产生,TTP 算法定义了 3 种类型的三角形。第 1 类三角形是三角形的 3 个点在同一个部分中;第 2 类三角形任意两个顶点在同一个部分中,另一个在其他部分中;第 3 类三角形是指 3 个顶点都在不同的部分中。GP 算法就是重复计算了第 1 类和第 2 类三角形。TTP 算法为了避免过多的重复计算,定义了 inner-edge:边的两个顶点在一个部分中,相反的即是 outer-edge。图 8 表示不含 inner-edge 的 3′-partition。TTP 算法在 2-partition 中计算第 1 类、第 2 类三角形,在 3′-partition 中计算第 3 类三角形,因此减少了冗余计算,减少了时间复杂度。

3.3.3 CTTP 算法

Park 和 Silvestri 针对 GP 算法的 "curse of the last reducer",提出了 CTTP(colored triangle type partition)算法,是第一个保证了每个 reducer 的最大输入的算法。该算法是在 MapReduce 的计算模型 MR(m,M)中提出来的,其中,m 是每个 mapper 或者 reducer 需要的空间,M 是整个计算中需要的空间。本算法进行 R 次 MapReduce 过程。CTTP 算法从 4-wise 独立族函数中随机选择一个颜色函数 $h(\cdot)$,进行顶点划分。CTTP 算法在 TTP 算法基础上,将问题分解为 $K=\binom{p}{3}+\binom{p}{2}=p(p^2-1)/6$ 子问题。子问题分为两类:一类是(i,j,k)子问题,用来计算第 3 类三角形;另一类是(i,j)子问题,用来计算第 1 类和第 2 类三角形。CTTP 通过均匀地将 K 个子问题分配给 $R=pE/M$ round 的方法,解决一个子问题只需要一个 reducer 的问题。如果 R 不是 2 或者 3,那么每一个 round 解决 K/R 个子问题。这个算法保证了每一个 mapper 发出同样数量的数据对。因此这个算法避免了被一些慢的 mapper 延迟了计算。

4 近似计算图三角

由于准确计算图三角的时空复杂度很大,同时很多应用只需要近似得到图三角的数量,所以近年来近似算法得到了很多关注。本文将近似计算图三角算法分为辅助算法、非流式算法和流式算法 3 个类别进行介绍。



(a) G'_{123} (b) G'_{124} (c) G'_{134} (d) G'_{234}

图 8 3′-partition G'_{ijk}

4.1 辅助算法

辅助算法是指这类算法经常被其他计算图三角算法引用，常常作为其他算法的一部分。

4.1.1 DOULIN 算法

Tsourakakis C E，Kang U，Miller G L 和 Faloutsos C 发明了 DOULIN[14]算法。DOULIN 算法不是处于其他计算三角形算法的对立面，而是处于所有算法的友好面。不论图能装进内存还是装不进，它都非常的实用。DOULIN 对每一个边都投掷一枚硬币，此边被保留的可能性是 p，被删除的可能性是 $1-p$。在最后剩下的图中找到的三角形的个数乘以 $1/p^3$ 就是对原图三角形个数的估计。

4.1.2 Colorful Function 算法

Colorful Funtion（颜色函数）[15]算法为图中每个顶点分配一种颜色，总的颜色数是 $N=1/p$，其中，p 是一个小于 1 的参数。当一个边的两个顶点被分配同一种颜色时，这个边称作是单色的。然后从所有的单色边中采样，计算采样到的单色边组成的三角形的个数，最后将计算的个数除以 p^2，得出近似估计的三角形个数。这个算法的关键点是关联采样的边。

4.2 静态算法

静态算法是基于静态图的，适用于离线计算。目前近似计算的静态算法只在单机上。在此方向上的研究较少，突破也很少，以下介绍的两种方法是有特点的算法。基于度的顶点划分的算法实现了时间空间更低的复杂度。随机矩阵迹算法采用了蒙特卡洛模拟方法。这两种方法均与其他方法有明显的区别。

4.2.1 基于度的顶点划分算法

Kolountzakis M N [16]等人研究发现，基于度对顶点进行划分，能得出在计算三角形时更小的运行时间上界。因为每个三角形都对应一个三元组，于是他们构造了一个三元组集合 U，这个三元组集合包括了所有的三角形。在这个集合里，均匀地选出一些三元组，标记为 1 到 s。当第 i 个三元组被采样时，如果它是一个三角形，那么 X_i 为 1；如果不是，则赋值为 0。由于是均匀地选取，并且一共得出 t' 个三角形，那么 $E(X_i)=t'/|U|$。因为每个 X_i 都是独立的，所以通过切诺夫界可以得到：

$$Pr\left[\left|\frac{1}{s}\sum_{i=1}^{s}X_i-\frac{t'}{|U|}\right|>\varepsilon\frac{t'}{|U|}\right]\leqslant 2e^{-e^2t's/(2|U|)} \quad (1)$$

由于 $|U|\leqslant n^3$，所以运行时间是 $O(n^3\lg n/(te^2))$。根据度划分顶点得出了 $|U|$ 更小的上界。理由是：对于一个包含 u 的三元组 (u,v,w)，如果 $\{u,v\}$、$\{u,w\}\in E$，这些三元组中含有 u 的个数最多是 $d(u)^2$。如果 $\{v,w\}\in E$，那么三元组中含有 u 的个数最多是 m。当 $d(u)^2>m$ 时，后者的界限更紧。所以，当顶点的度小于 $m^{1/2}$ 时，那么它属于低度顶点，三元组中含有低度顶点的三角形的个数最多是 $m^{3/2}$。当所有顶点度之和是 $2m$ 时，三元组中含有高度顶点的三角形的个数最多是 $2m^{3/2}$。结合起来，$|U|$ 的上界是 $3m^{3/2}$，所以得到 $|U|\leqslant O(m^{3/2})$，时间上界是 $O\left(m+\frac{m^{3/2}\lg n}{te^2}\right)$。

4.2.2 随机矩阵迹算法

Avron H[17]采用随机矩阵迹的方法去估计大图中三角形的个数。该方法依据的是准确计算三角形算法里的矩阵相乘算法，采用 Monte-Carlo 模拟来估算三角形个数。每一个样本需要 $O(E)$ 的时间，需要 $O(e^{-2}\lg(1/\delta)\rho(G)^2)$ 个样本才能保证 (ε,δ)-approximation，其中，$\rho(G)$ 是对图 G 稀疏的一种测量。这个算法很高效，只需要 $O(V)$ 的空间和 $O(\lg^2|V|)$ 个样本就能达到一个很好的估算。一个维数高的矩阵的立方计算量很大，所以这个算法生成一个随机向量 $x=(x_k)$，其中，$x_i\sim N(0,1)$（正态分布）。将 y 赋值为 Ax，$T_i=(y^TAy)/6$，循环 $M=|\gamma\ln^2 n|$ 次，最后三角形的个数估计为 $\frac{1}{M}\sum_{i=1}^{M}T_i$。

4.3 流式算法

流式算法基于动态图流，与静态算法相比，适用于在线计算。图流有不同形式，主要分为任意流（arbitrary streams）和事件流（incidence streams）两类。在任意流中，边在流中是不重复的，且是以任意顺序出现的；在事件流中，边是按照每个顶点的邻边出现的，例如，首先顶点 v_1 的所有邻点出现，接着 v_2 的所有邻点出现。$v_1,v_2,\cdots,v_n$ 的顺序是由输入方确定的。根据算法通过流的次数，可以将算法又分为 one-pass 算法和 multiple-passes 算法。

下面介绍一个 multiple-passes 采样三角形的算法，这里的流是任意流，算法是 Buriol L S[18]提出的 3-passes 算法：将流中所有边的数目计算出来，为 $|E|$；从流中均匀选择一条边 $e=\{a,b\}$，也均匀选择出一个顶点 v，这个顶点属于 $V\backslash\{a,b\}$；如果 $\{a,v\}$ 和 $\{b,v\}$ 都属于 E，计数 $\beta=1$，否则计数 $\beta=0$。最后返回 β 值。这个算法中有一定数目的估计器（estimator），每个估计器得到一个 β 值，求出期望 $E[\beta]$ 后，三角形的个

数 T' 就估算为 $E[\beta]\cdot|E|\cdot(|V|-2)/3$。

multiple-passes 算法可以合成为 one-pass 算法，Buriol L S 将 3-passes 算法合成的 1-pass 算法是：随机取一个顶点 v，并在流中采样一条边 $\{a,b\}$，如果能在接下来的流中检测到边 $\{a,v\}$ 和 $\{b,v\}$，则三角形计数，否则不计数。multiple-passes 算法的消耗多于 one-pass 算法[30]。

下面介绍 3 种 one-pass 算法：基于邻居采样(Neighborhood sampling)算法、基于 2-path 和图稀疏的算法和 TRIEST 算法，都与采样有关。但由于采样方法各不相同，3 种算法的时空消耗显而易见。基于 2-path 和图稀疏的算法需要存储多个稀疏图，所以空间消耗很大，TRIEST 算法对每条边的到来都进行一次邻点交集计算，所以时间消耗很大。3 种算法的准确率也有差异。表 3 是 3 个算法准确度的比较。对于几种不同的数据集，并没有完全优势的算法。可见对于含有不同数据意义的应用，需要采用不同的算法。

表 3　3 种流式算法的准确度比较

数据集	A. Pavany	Laurent Bulteau	TRIEST
DBLP	0.19	2.3	5.3
YouTube	4.42	1.7	7.8

4.3.1　基于邻居采样算法

Pavany A 和 Tangwongsan K[19]等人设计了一个时间空间效率都挺高的图流算法。这个算法是基于邻居采样的 one-pass 算法：首先从流中随机采样一条边，然后采样和该边有共同顶点的边。$N(e)$ 表示在流中与边 e 相邻，但是在 e 边后面到来的边。其中，$c=c(e)=|N(e)|$。数据以块的形式到达，块的大小是 w。对于每一个边的到来，设置 r 个估计器，以 $m/(w+m)$ 的概率保留这条边。然后采样边的邻边，以 $c^+(e)/(c^-(e)+c^+(e))$ 的概率从 $N(e)\cap B$ 采样一条边。最后判断这两条边能否与后来的边组成一个三角形。这个算法的时间空间复杂度都是 $O(r+w)$。

4.3.2　基于 2-path 和图稀疏的算法

Bulteau L 和 Froese V [20] 等人设计了一个基于 2-path 采样和图稀疏的方法，来估计动态增删图流中三角形的数量，采用的是 one-pass。相对基础图流采样对删除边后采样的子图是否仍存在的未知，图稀疏能处理边的删除。该算法最大的挑战是显示了稀疏图中的 2-path 采样几乎等同于原图的 2-path 采样。随机选择了大量的 2-path，与用其中能组成三角形的 2-path 数量去估计传递系数。对于图流中每一个增删边的到来会更新 SME (second moment estimator)，图流完全通过将返回整体 2-path 值。与此同时，用颜色散列(coloring hash)函数族去稀疏图流，得到数个稀疏图。对于每一个稀疏图，随机采样一定数目的 2-path 判断是否组成三角形，并计数。最后用稀疏图中采样的 2-path 中能组成三角形的数量与采样的 2-path 数量的比值，与整体 2-path 数量比较，去估计整个图流中三角形的数量。该算法得到了图流中计算三角形复杂度的下界。

4.3.3　TRIEST 算法

由于基于采样的流算法，事先都要确定边采样的概率 p，所以会造成一些问题。如在存储空间有限的情况下，需要知道流的规模；采样留下的边的规模会增长，如果 p 较大，会溢出存储空间，如果 p 较小，得到的结果是次优的；即使设定了特定的 p，使在流结束时恰好装满存储空间，得到的结果也不是最好的。TRIEST 算法针对这些问题提出了解决方法。

TRIEST [21]算法是在 one-pass 下的，相对于只有增加边的流算法 MSCOT[22]，它还有删除边的流，时刻计算三角形的数量，并且存储空间是固定的。该算法采用水库采样(reservoir sampling)和随机配对(random pairing)两种采样方案使得存储空间尽可能多地被利用。由于是时刻计算三角形的数量，图采用了时间标记 G^t，每到来一条边，时间增长一个单位。该算法默认对一条边的删除一定是在对这条边增加之后。如果设定的内存大小是 M，对于当前流入的插入边 $e=\{a,b\}$，如果之前图的大小 G^{t-1} 不大于 M，当前边被保留。如果之前的图的大小已经达到过 M，这时以 M/t 的概率决定直接舍弃当前边，或者选择删除之前图中的一条边，且插入当前边，这是标准水库采样的应用。若插入了当前边，便计算当前边是否组成三角形。对于当前流入的边是删除边，计算当前边是否在之前流入的图中组成三角形，如果组成，便减去相应的数量。为保证存储空间在增加边和删除边的时候都能充分利用设定的内存，TRIEST 基于随机配对做了补偿策略，删除边需要被未来的插入边补偿。设置了两个计数器 d_{in} 和 d_{out}。如果之前流入图的大小 G^{t-1} 已经达到内存大小 M，当前边是一条删除边，如果此删除边仍在 G^{t-1} 中，d_{in} 加一；如果此删除边之前被替换出去，不在 G^{t-1} 中，d_{out} 加一。之后到来的增加边会根据 d_{in} 和 d_{out} 的值被保留或不被保留。如果 d_{in}、d_{out} 之和等于零，增加边采用标准的水库采样方案。如果不等于零，便以 $d_{in}/(d_{in}+d_{out})$ 的概率保留增加边，如果保留，$d_{in}-1$，否则 $d_{out}-1$。TRIEST 算法在完全动态的图流中做到了无偏、低方差、高质量的

估计。并且在有数十亿条边的大图中有更小的平均估计误差。

5　性能比较

综上所述，在计算图三角的算法中，准确计算图三角与近似计算图三角算法有区别也有联系。准确计算中，点边迭代算法和矩阵相乘算法是最基础的算法。但是对于数据量越来越大的图来说，在普通计算机上运行这些算法的时间空间复杂度非常高，并不适用。基于它们的改进算法 FCNI 提供了快速实现求共同邻点的方法，也使在大规模图中计算三角形的运行时间大大提高，但是实验机器成本很高。外存计算算法很好地解决了图规模过大，普通机器内存不够用的问题。Chu 和 Cheng 提出的基于图划分的算法，一定程度上避免了内存小的问题，但是对于不均匀的图来说，划分是一个难题。Hu 提出的有效 I/O 的算法，巧妙避免了重复计算，时空复杂度为目前最好，且可以移植到不同平台上进行计算。GP、TTP 和 GTTP 采用分布式框架为图三角算法提供了新的平台，提供了更多的内存，又可以并行，使时空复杂度减少。TTP、GTTP 分别针对 GP 的重复计算和冗余等待时间做了改进，在 MapReduce 框架下取得了好的效果。近似计算中，DOULION 算法和颜色函数对采样很有帮助，被很多其他算法运用。静态图的计算图三角算法中，基于度分割和随机矩阵迹是有特点的方法，但是空间消耗都较大、效率不高。部分算法是基于动态图流的，multiple-passes 算法时空消耗大，更多算法采用的是 one-pass。近似算法大部分采用采样方法，各有优缺点。Pavany A 提出的邻居采样算法的空间效率很高，但是参数过多。Laurent 提出的基于 2-path 和图稀疏的算法效果一般，但首次达到了每边的固定处理时间。TRIEST 算法是目前准确率最高的，且可以在固定内存中时刻计算图流中三角形数量。近似计算应用范围很广，并且较准确计算来说需要的时空复杂度明显降低，但准确度是一个关键问题。大部分近似算法是在准确计算的基础上进行改进的，如矩阵迹算法是在矩阵相乘算法上加入了蒙特卡洛方法。上文几种采样算法是在点边迭代算法的基础上加入了采样策略。

本文将近几年的计算图三角算法按准确计算和近似计算进行了比较，见表 4、表 5。

6　结束语

面对社会网络的快速发展，图的规模越来越大，关于图的问题也越来越多样化。选择合适图存储模型和计算模型对图的计算很重要。根据不同的问题需要选择不同的解决方案，面对新的应用也要研究出新的方法。近年来出现的高效的分布式系统为图三角算法提供了新的平台。基于分布式的三角形计算算法的可行性越来越大，这将会成为三角形计算的下一个研究重点。

参考文献：

[1] THOMAS S. Algorithmic aspects of triangle-based network analysis[J]. Phd in Computer Science, 2007:26-29.

[2] COPPERSMITH D, WINOGRAD S. Matrix multiplication via

表 4　准确计算三角形算法比较

类别	算法	优点	缺点	适用范围
内存算法	FCNI	时间效率高	空间消耗大	内存大，静态图
外存算法	Chu and Cheng	简单易实现	时空消耗大	静态图
	MGT	I/O 效率高	冗余	静态图
分布式算法	GP	简单易实现	冗余	分布式，静态图
	TTP	时间效率高	没有考虑负载均衡	分布式，静态图
	GTTP	考虑负载均衡	时间复杂度高	分布式，静态图

表 5　近似计算三角形算法比较

类别	算法	优点	缺点	适用范围
静态算法	Kolountzakis N	时间复杂度低	空间消耗大	静态图
	Avron H	时间复杂度低	空间消耗大	静态图
流式算法	Pavany A	空间消耗小	只考虑边插入的情况	支持仅有插入边的图流
	Bulteau L	考虑边的插入/删除	空间消耗较大	支持有插入或删除边的图流
	TRIEST	考虑边的插入/删除，可设定存储大小	时间消耗较大	支持有插入或删除边的图流

arithmetic progressions [J]. Journal of Symbolic Computation, 1990, 9(3): 251-280.

[3] ALON N, YUSTER R, ZWICK U. Finding and counting given length cycles[J]. Algorithmica, 1997, 17(3): 209-223.

[4] THOMAS S, WAGNER D. Finding, counting and listing all triangles in large graphs, an experimental study [C]//The 4th International Workshop, May 10-13, 2005, Santorini Island, Greece. New York: Springer, 2005.

[5] CHIBA N, NISHIZEKI T. Arboricity and subgraph listing algorithms[J]. Siam Journal on Computing, 1985, 14(1): 210-223.

[6] KUMAR R, RAGHAVAN P, RAJAGOPALAN S, et al. The web as a graph: measurements, models, and methods [C]//The 5th Annual International Conference, July 26-28, 1999, Tokyo, Japan. New York: ACM Press, 2000: 1-17.

[7] LATAP M. Theory and practice of triangle problems in very large (sparse (power-law)) graphs[EB/OL]. [2006-09-20]. http://arxiv.org/pdf/cs/0609116. pdf.

[8] SEVENICH M, HONG S, WELC A, et al. Fast in-memory triangle listing for large real-world graphs[C]//The 8th Workshop on Social Network Mining and Analysis SNAKDD, August 24-27, 2008, Las Vegas, NV, USA. New York: ACM Press, 2014.

[9] CHU S, CHENG J. Triangle listing in massive networks and its applications [C]//The 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 21-24, 2011, San Diego, CA, USA. New York: ACM Press, 2011: 672-680.

[10] HU X C , TAO Y F . I/O efficient algorithms on triangle listing and counting [J]. ACM Transactions on Database System, 2014, 39(4): 1-30.

[11] SURI S, VASSILVITSKII S. Counting triangles and the curse of the last reducer [C]//The 20th International Conference on World Wide Web, March 28-April 1, 2011, Hyderabad, India. New York: ACM Press, 2011: 607-614.

[12] PARK H M, CHUNG C W. An efficient MapReduce algorithm for counting triangles in avery large graph [C]//ACM Conference of Information and Knowledge Management, October 27-November 1, 2013, San Francisco, CA, USA. New York: ACM Press, 2013: 539-548.

[13] PARK H M, SILVESTRI F, KANG U, et al. MapReduce triangle enumeration with guarantees[C]//ACM Conference of Information and Knowledge Management, November 3-7, 2014, Shanghai, China. New York: ACM Press, 2014.

[14] TSOURAKAKIS C E, KANG U, MILLER G L, et al. DOULIN: counting triangles in massive graphs with acoin [C]//The 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, June 28-July 1, 2009, Paris, France. New York: ACM Press, 2009: 837-846.

[15] PAGH R, TSOURAKAKIS C E. Colorful triangle counting and Mapreduce implementation [J]. Information Processing Letters, 2011, 112(7): 277-281.

[16] KOLOUNTZAKIS M N, MILLER G L, PENG R, et al. Efficient triangle counting in large graphs via degree-based vertex partitioning[J]. Internet Mathematics, 2010, 8(1-2): 15-24.

[17] AVRON H. Counting triangles in large graphs using randomized matrix trace estimation [J]. In Large-Scale Data Mining: Theory and Applications (KDD Workshop), 2010.

[18] BURIOL L S, FRAHLING G, LEONARDI S, et al. Counting triangles in data streams [C]//The 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA. New York: ACM Press, 2006: 253-262.

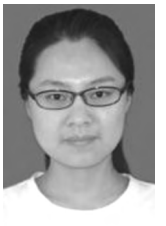
[19] PAVANY A, TANGWONGSAN K, TIRTHAPURAZ S, et al. Counting and sampling triangles from a graph stream [J]. Proceedings of the Vldb Endowment, 2013, 6(14): 1870-1881.

[20] BULTEAU L, FROESE V, KUTZKOV K, et al. Triangle counting in dynamic graph streams [EB/OL]. [2015-07-14]. http://itu.dk/people/konk/papers/dtc_full.pdf.

[21] STEFANI L D, EPASTO A, RIONDATO M, et al. TRIEST: counting local and global triangles in fully-dynamic streams with fixed memory size[EB/OL]. [2016-02-24]. http://arxiv.org/abs/1602.07424.

[22] LIM Y, KANG U. MASCOT: memory-efficient and accurate sampling for counting local triangles in graph streams [C]//The 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD, August 10-13, 2015, Hilton, Sydney. New York: ACM Press, 2015: 685-694.

[作者简介]



金宏桥(1993–),**女**,**宁波大学信息科学与工程学院硕士生**,**主要研究方向为大数据、数据挖掘**。

董一鸿(1969–),**男**,**博士**,**宁波大学教授**,**主要研究方向为大数据、数据挖掘和人工智能**。