基于 GAS 模型的 *k*-truss 分解算法*

王 邠，周 刚†，张凤娟
（数学工程与先进计算国家重点实验室，郑州 450001）

摘 要：在大规模网络中发现稠密子图具有极其广泛的应用，如社区发现、垃圾邮件检测等。针对大规模网络数据中快速、有效地发现稠密子图，提出了一种基于 GAS（gather-apply-scatter）编程模型的分布式 *k*-truss 算法——GASTruss。算法采用 GAS 的模式完成数据同步和算法迭代，有效地克服了传统并行算法重复性计算及不能有效处理依赖关系大的数据等问题。实验选择在 GraphLab 平台上进行，结果表明：与串行 *k*-truss 算法以及基于MapReduce的 GPTruss 算法性能相比，GASTruss 算法对数据规模具有良好的拓展性，在保持算法效果的同时能有效降低时间复杂度。

关键词：*k*-truss 分解；稠密子图；分布式算法；GAS 模型
中图分类号：TP301　　**文献标志码**：A　　**文章编号**：1001-3695(2018)06-1691-05
doi:10.3969/j.issn.1001-3695.2018.06.020

GAS based *k*-truss decomposition method

Wang Bin, Zhou Gang†, Zhang Fengjuan
(*State Key Laboratory of Mathematical Engineering & Advanced Computing*, *Zhengzhou* 450001, *China*)

Abstract: Finding dense subgraphs in the large-scale network has the extremely widespread uses, such as community discovery and spam detection, etc. In order to find dense subgraphs rapidly and effectively in the large-scale network, this paper proposed a distributed *k*-truss decomposition method based on GAS: GASTruss. The algorithm adopted the gather-apply-scatter programming model to complete the data synchronization and iteration, and effectively overcame the problem of traditional parallel algorithms in overmuch repetitive calculation and incapacity of handling the data with strong dependencies. It tested experiment on GraphLab platform, the results show that compared with the serial *k*-truss algorithm and the MapReduce-based algorithm GPTruss, this method outperforms in terms of running time while maintaining the calculate accuracy.

Key words: *k*-truss decomposition; cohesive subgraph; distributed algorithm; GAS programming model

0 引言

由于现实中的网络规模大，对全网的性质进行分析时间成本较高，在规模较小、具有代表性的子图上研究全网的连通性、中心性等性质是一种可行有效的研究方法。对稠密子图的研究是这种方法的典型代表。稠密子图在现实生活中有着丰富的应用场景和重要的应用价值。例如，稠密子图常用于在社交网络中发现具有紧密联系的相似性社区；在生物学方面，稠密子图挖掘用来鉴定复杂蛋白质网络的拓扑特征。

稠密子图有多种不同的定义。最基本的稠密子图定义是团[1]和最大团。团是图 G 的一个完全子图，定义如下：给定无向图 $G=(V,E)$，若 $U\subseteq V$，且对任意 $u,v\in U$ 有 $(u,v)\in E$，则称 U 是 G 的团。若团 U 不包含在 G 的更大的团中，称团 U 为图 G 的极大团，图 G 中节点最多的极大团称为图 G 的最大团。团的定义要求子图中任意两点之间存在直接相连的边，这使得图 G 中的团结构只能描述复杂网络中相对较小的一个子结构，它们可以分散在图的各个部分，也可能一部分相互有大量的重叠而另一部分关系很小。如图 1(b)是图 1(a)中的五个最大团，图中可以看出节点集合{5,8,10,19}、{10,15,18,19}构成的团位于图的边缘部分，{4,8,18,21}、{4,8,10,18}两个团有三个重复的节点。文献[2~6]提出一些条件较宽松的稠密子图定义。n-团[2]将团中两个节点之间的距离为 1 的限制放宽到 n；k-plex[3]将 n 个节点的团中每个节点的度为 $(n-1)$ 放宽到 $(n-k)$；准团对子图的密度[4]和节点的度[5,6]的要求进行了进一步的放宽。计算上述稠密子图问题仍然是 NP-难问题[7]。

k-truss 描述了图的一种稠密子图[8]。给定无向图 $G=(V,E)$，图 G 的 *k*-truss 是 G 的一个最大子图 $G_S=(V_S,E_S)$，V_S 和 E_S 表示子图 G_S 的节点集合和边集合，使得 $\forall e\in E_S$，至少包含于子图 G_S 的 $(k-2)$ 个三角形中。图 G 的 truss 分解问题，就是找出 G 的所有非空 *k*-truss，$k\in \mathrm{N}_+$。*k*-truss 从不同的粒度层次反映了网络的核心部分，3-truss 和 4-truss 如图 1(d)所示。由定义可知，图 G 本身构成一个 2-truss，2-truss 到 max-truss 构成一系列子图，越来越细致地度量了网络的核心部分。从图中还可以看出，4-truss 包含了图 G 的所有五个最大团。从这个方面来讲，*k*-truss 更加相似于 *k*-core[9~12]。*k*-core 是一个图的最大子图，子图中每个节点的度至少为 k，图 G 中的 3-core 如图 1(c)所示。如文献[9]中描述的，*k*-core 是一个粗略的稠密子图概念。而且，由定义可知，*k*-truss 中边至少包含于 $(k-2)$ 个

收稿日期：2017-01-18；**修回日期**：2017-03-06　　**基金项目**：数学工程与先进计算国家重点实验室开放基金资助项目(2015A11)
作者简介：王邠(1991-)，男，山东潍坊人，硕士研究生，主要研究方向为大规模图数据处理、机器学习；周刚(1974-)，男(通信作者)，副教授，硕导，博士，主要研究方向为在线社会网络分析、海量信息处理(770117938@qq.com)；张凤娟(1983-)，女，讲师，硕士，主要研究方向为大数据分析、数据挖掘.

三角形中,所以子图中的每个节点至少有($k-1$)个邻居,即 k-truss 是一个($k-1$)-core,但反之不然。

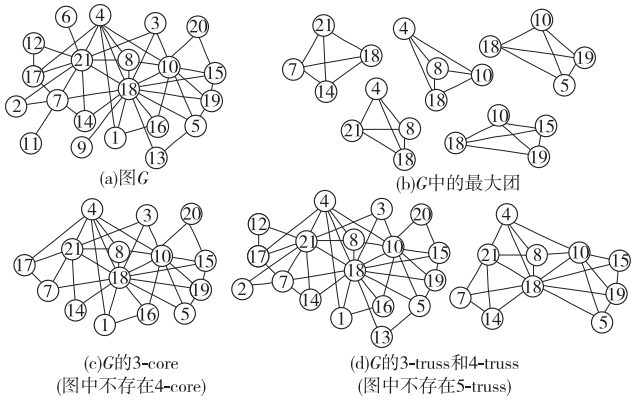


图1 图G中稠密子图例子

从概念上讲,k-truss 的定义也比 k-core 更加严密,因为 k-truss 的定义依据是三角形,而三角形是公认的网络的基础组成模块[10,11,13]。在社交网络中,三角形意味着三个紧密相互连接的节点,也可理解为两个节点共享一个邻居节点。k-truss 通过限制每条边至少包含在($k-2$)个三角形中,更好地表现了社团内部节点之间的紧密程度。从社交网络中可以直观地发现,如果两个人拥有越多的共同好友,则两个人会同时处于越多的三角形之中,也就意味着两个人更可能属于同一个社团组织。相对地,k-core 只限制单个节点的度数,而无法度量节点之间联系的紧密程度。

从计算复杂度上来讲,非层次的稠密子图挖掘算法,主要包括团[1,14]和准团[4,15],时间复杂度很高。层次聚类算法主要包括 k-core、k-truss 算法,只需要多项式时间复杂度。几种稠密子图之间的比较如表 1 所示。

表 1 几种稠密子图的比较

稠密子图种类		特征	优缺点
团		团的内部中任意两个节点之间存在一条边	优点:团内部节点之间连接紧密。缺点:团与团之间可能大部分重叠也可能毫无关联;团可能处于整个网络的边缘位置;团的定义过分严格,网络中团的规模往往较小;计算网络的团是 NP-完全问题
准团	基于密度	子图 S 中至少含有 $\gamma\times|S|(|S|-1)/2$ 条边	优点:准团内部节点之间连接较紧密;放宽了团的定义的限制。缺点:准团中密度分布可能不均衡;计算准团是 NP-难问题;准团之间也存在重叠问题和无关联问题等等
	基于度	k 个节点的子图中每个节点的度大于等于 $\gamma\times(k-1)$	
k-core		k-core 中每个节点的度至少与其他 k 个节点之间存在边	优点:计算 k-core 只需要多项式时间复杂度。缺点:无法度量内部节点之间的紧密程度
k-truss		k-truss 中每条边至少包含在 $k-2$ 个三角形之中	优点:计算 k-truss 只需要多项式时间复杂度;2-truss 到 max-truss 刻画了不同粒度层次的网络核心;内部节点连接紧密

将 k-truss 子图用于网络分析的方法最早由 Cohen[4] 提出,其算法过程中需要随机访问节点或者边,因此需要将整个网络载入到内存中。然而,对于现实的复杂网络,将整个网络载入到内存中不再现实。2009 年,Cohen[16] 又提出了一种基于 MapReduce的 truss 分解并行算法——MPTruss,不需要将整个网络载入到单个机器中。为了计算 k-truss,该算法反复调用 map-reduce 过程来列举三角形,找到那些至少包含在($k-2$)个三角形中的边。然而利用 MapReduce 计算三角形的过程需要迭代执行多次主程序,造成磁盘利用率低、过多的中间结果以及大量不必要的 I/O 操作。2014 年,Chen 等人[17] 基于 BSP 模型对 truss 分解的并行算法进行了改进,提出一种基于局部性理论的 k-truss 算法——GPTruss,从而降低了通信开销和算法迭代次数,在运行效率方面优于 MPTruss 算法,是目前最高效的并行 k-truss 算法。本文提出了一种基于 GAS 模型的 truss 分解方法——GASTruss。GAS 编程模型的核心思想是以节点为中心,支持稀疏的计算依赖、异步迭代计算,在处理数据依赖性强、流计算、执行过程多次迭代等问题方面表现良好。算法在分布式图计算框架 GraphLab 进行了实现,该框架对分布式计算进行了高层次抽象,有助于高效快速实现迭代的分布式算法,GraphLab 的内存共享机制使得算法能够处理大规模的复杂网络并且可以随着集群规模扩展。使用现实中的网络数据集对 GASTruss 算法的性能进行了测试,并同 GPTruss 算法进行了比对分析,结果表明 GASTruss 对数据规模具有良好的扩展性,在保持算法子图挖掘效果的同时能有效降低时间复杂度。

1 问题描述

1.1 k-truss 相关概念

本文中只考虑简单图。给定一个图 G,用 V_G 和 E_G 表示图 G 的节点集合和边集合,$n=|V_G|$ 表示 G 中节点的个数,$m=|E_G|$ 表示 G 中边的个数。用 $nb(v)$ 表示节点 v 的邻居集合,即 $nb(v)=\{u|(u,v)\in E_G\}$。定义节点 v 的度为 $|nb(v)|$,用 $\deg(v)$ 表示。图 G 中的三角形是图中一个长度为 3 的环。假设 $u,v,w\in V_G$ 是环上的三个节点,用 Δ_{uvw} 来表示这个三角形。另外,用 Δ_G 来表示 G 的所有三角形的集合。基于三角形的定义,定义边的支持度[4] 如下:

定义 1 支持度。图 G 中,边 $e=(u,v)\in E_G$,节点 $w\in V_G$,那么定义 e 的支持度 $\sup(e,G)=|\{\Delta_{uvw}\mid \Delta_{uvw}\in\Delta_G\}|$,简写为 $\sup(e)$。

边 e 的支持度是图 G 中包含 e 的三角形的数量。下面给出 k-truss[4,7] 的定义。

定义 2 k-truss。图 G 中,定义 k-truss,用 T_k 表示,为满足下列不等式的最大子图,即 $\forall e\in E_{T_k},\sup(e,T_k)\geqslant(k-2)$,其中 $k\geqslant 2$。

根据定义,图 G 本身就是一个 2-truss。定义边 e 的 truss 数为 $\max\{k|e\in E_{T_k}\}$,用 $\varphi(e)$ 表示。根据定义,若 $\varphi(e)=k$,有 $e\in E_{T_k}$ 但 $e\notin E_{T_{k+1}}$。用 $k_{\max}$ 来表示 G 中所有边的 truss 数的最大值。根据 truss 数的定义有了 k-class 的概念。

定义 3 k-class。图 G 中,定义 k-class 为 $\{e\mid e\in E_G,\varphi(e)=k\}$,用 Φ_k 表示。

1.2 truss 分解

本文中,truss 分解问题的定义如下:给定一个图 G,计算 G 所有的 k-truss,其中 $2\leqslant k\leqslant k_{\max}$。$k$-truss 用边的集合来表示可以表达为 $E_{T_k}=\cup_{j\geqslant k}\Phi_k$,也就是说,$k$-truss 是所有 truss 数至少为 k 的所有边的集合。图 2 中,不同形状的线条表示不同的 k-class,其中 $2\leqslant k\leqslant 5$。每条边的权值表示每条边的 truss 数,图 2 中 $k_{\max}=5$。由图中所示,图 G 中,2-class 只有一条边(9,12),即 $\Phi_2=(9,12)$。那么同理,$\Phi_3=\{(4,7),(4,11),(4,12),(5,6),(5,7),(6,7),(7,8),(7,11),(7,12)\}$,$\Phi_4=\{(6,8),(6,9),(6,10),(8,9),(8,10),(9,10)\}$,$\Phi_5=\{(1,2),(1,3),(1,4),(1,5),(2,3),(2,4),(2,5),(3,4),(3,5),(4,5)\}$。通过 k-class 可以得到 k-truss 分解结果如下:2-truss,T_2 即 G 本身;T_3 是由边的集合 $\{\Phi_3\cup\Phi_4\cup\Phi_5\}$ 组成的子图;T_4

是由 $\{\Phi_4 \cup \Phi_5\}$ 组成的子图;T_5 是由 Φ_5 组成的子图。

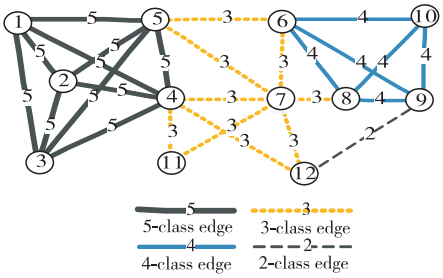


图2　图G中的k-class

2　相关研究

2.1　k-truss 串行算法——Cohen 算法

k-truss 串行算法的基本思路是:从 $k=3$ 开始,首先计算图 G 中每条边的支持度,循环删除支持度小于$(k-2)$的边,得到 k-truss。算法 1 举例了文献[4]中提出的 truss 分解算法。算法伪代码如下所示。

算法 1　k-truss 串行算法

输入:网络 $G=(V_G,E_G)$。
输出:所有的 k-truss,$3 \leqslant k \leqslant k_{\max}$。
//初始化,计算每条边的支持度
1　$k \leftarrow 3$;
2　for each $e=(u,v) \in E_G$ do
3　　$sup(e)=|nb(u) \cap nb(v)|$;
　　//删除支持度不足的边,并递减受影响的边的支持度
4　　while ($\exists e=(u,v)$ such that $sup(e)<(k-2)$)
5　　$W \leftarrow (nb(u) \cap nb(v))$;
6　　for each $e'=(u,w)$ or $e'=(u,w)$ where $w \in W$, do
7　　　$sup(e') \leftarrow sup(e')-1$;
8　　remove e from G;
9　　output G as the k-truss;
10　if (not all edges in G are removed)
11　　$k \leftarrow (k+1)$;
12　go to step 4;

算法 1 在初始化阶段计算了图 G 中每条边的支持度(步骤 2~3)。对于每条边 $e=(u,v)$,$nb(u)$ 和 $nb(v)$ 的交集返回的是与定点 u 以及节点 v 构成三角形的节点集合,所以交集的势就是边 e 的支持度。

初始化之后,从 $k=3$ 开始,每次循环中算法移除支持度小于$(k-2)$的边,根据定义这些边不可能存在于 k-truss 中(步骤 4~8)。然而,将边 $e=(u,v)$ 移除会破坏形如 $\{\Delta_{uvw} \mid w \in W=nb(u) \cap nb(v)\}$ 的所有其他包含边 e 的三角形的结构,所以需要递减三角形中另外两条边(u,w)和(v,w)的支持度。这个过程迭代重复执行,直到图 G 所有剩余边的支持度均不小于$(k-2)$,这样就得到一个 k-truss。如果此时图 G 中仍有边没有被移除,重复执行上述步骤 4~8 计算下一个 k-truss。

算法 1 复杂度如文献[7,17]所描述,需要 $O(m+n)$ 的存储空间来维护整个图以及每条边的支持度。初始化过程(步骤 2、3)可以使用内存三角计数算法[18,19]进行加速优化。然而,步骤 5 中处理每条边仍然需要 $O(\deg(u)+\deg(v))$ 的时间,所以时间复杂度为 $O(\sum_{v \in V_G}(\deg(v))^2)$。

2.2　基于 MapReduce 的 k-truss 算法——MPTruss

根据定义,基于 MapReduce 的 k-truss 算法是基于三角枚举算法的,通过枚举三角形计算每条边的支持度,支持度大于等于$(k-2)$的边的集合构成一个 k-truss。基于 MapReduce 的 k-truss 算法描述如下。

算法 2　基于 MapReduce 的 truss 分解算法

输入:网络 $G=(V_G,E_G)$,$k=2$。
输出:k-truss。
//枚举三角形
map 1:
for each $e=(u,v) \in E_G$ {
　key = min{u.id, v.id};
　emit(map1_record, key, e.value());}
reduce 1:
if(map1_record[i].key = map1_record[j].key && $i \neq j$) {
　key = asstring('(i,j)');
　value = map1_record[i].value() + map1_record[j].value();
　emit(triad, key, value);}
map 2:
for each $e=(u,v) \in E_G$ {
　key = asstring('(u,v)');
　emit(map2_record, key, e.value());}
reduce 2:
if triad[i].key = map2_record[j].key {
　value = triad[i].value() + map2_record[j].value();
　emit(triangle, null, value);
　triangle_num ++;}
　//计算边的支持度,输出支持度大于等于 $k-2$ 的边
map 3:
for ($i=0$; $i \leqslant$ triangle_num; i ++) {
　if $e=(u,v)$ contained in triangle[i]
　e.truss = e.truss + 1;}
reduce 3:
for each $e=(u,v) \in E_G$ {
　if e.truss $\geqslant k-2$;
　output(e);}
if(output is not null) {
　　$k=k+1$;
　　go to step map 1;}

基于 MapReduce 的 truss 分解算法分为两个过程:a)枚举三角形;b)计算边的 truss 数,去掉支持度不足的边。算法 2 中,步骤 1~4 表示枚举三角形,主要分为两个 map-reduce 阶段:步骤 1、2 是第一个阶段枚举形如 $\{(A,B),(B,C)\}$ 的一对边构成的三元组(triads),本文称之为开放三角形;步骤 3、4 是第二个阶段辨别是否存在一条边(A,C)与该开放三角形构成一个闭合的回路,从而构成三角形。

在算法描述中,步骤 5、6 构成了另一个 MapReduce 过程,它计算每条边的支持度,删除支持度小于$(k-2)$的边,即得到一个 k-truss。若图 G 不为空,重复上述过程。

MapReduce 方法存在的问题在于,当每次移除支持度不足的边之后,会造成其他相关联边的支持度降低,从而每次迭代过程需要重新进行三角形枚举的步骤,而三角形枚举需要进行两个 map-reduce 阶段从而导致算法有许多冗余的工作。

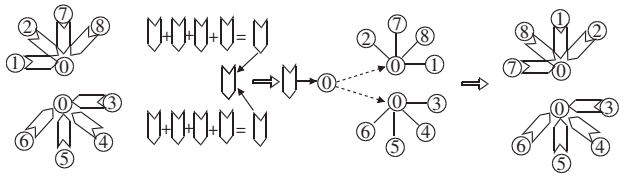
2.3　GAS 计算模型及 GraphLab 框架

如今,海量数据盛行,很多复杂网络算法数据依赖性强;流处理复杂,整个执行过程需要多次迭代才能完成计算。针对海量数据的复杂网络分析算法对并行计算的性能、开发复杂度提出了新的挑战。

为了应对海量数据以及大图数据分析时计算依赖高、迭代计算等特点,以节点为中心的编程模型首先由 Pregel[12] 系统提出。这种编程模型将图节点看做计算的中心,开发人员可以自定义节点更新函数用来获取并改变节点及与其关联的边的权值,从而改变整个图的状态,如此反复迭代,直到达到预定的标准。

GAS 编程模型可以看做是对以节点为中心的图计算编程模型的一种细粒度的改造,它通过进一步细分更新函数来增加计算的并行性。GAS 模型明确地将更新函数划分为三个连续

的处理阶段,即 gather、apply、scatter。它将完整的计算流程细分为可以并发执行的子处理阶段,进一步增强了系统的并发处理性能。GAS 编程模型如图 3 所示。



(a)gather (b)apply (c)scatter

图 3 GAS 模型

Gather 阶段,当前工作节点收集邻接节点和相连的边(可能是所有边,也有可能是入边或者出边)的数据,根据用户定义的累加函数求和。Apply 阶段,利用汇总的结果和上一步的节点数据进行进一步计算后更新节点数据。最后,在 scatter 阶段,当前工作节点更新完成之后,更新边上的数据,并通知对其有依赖的邻接节点更新状态。2010 年 CMU 的 Select 实验室提出了 GraphLab 框架[20]。2012 年 GraphLab 优化其并行模型[21],使其对处理图的并行性能得到显著提升。它是一个分布式异步内存共享系统,节点程序可以直接访问该节点、相邻边和相邻节点的信息,并通过阻止相邻的程序同时运行来保证结果的正确性,其核心思想是以节点为中心,支持稀疏的计算依赖、异步迭代计算等,有效解决了 MapReduce 不适应需要频繁数据交换的迭代并行算法的问题。

3 基于 GAS 的 *k*-truss 算法

根据 k-truss 的定义,一个 truss 中的边要求至少存在于 $k-2$ 个三角形中,在计算过程中需要以边为对象进行操作。首先,传统的 MapReduce 模型处理边时,只能将边作为一条记录,它与图中的节点以及其他对象只能用不同的键值进行区分,这一方面增加了空间复杂度,另一方面暴露了 MapReduce 在处理图数据时的缺陷。其次,在每次迭代计算过程中,MapReduce需要重新枚举三角形从而得到边的 truss 数,每次计算需要访问全局所有记录,这导致计算过程中存在大量冗余工作。

GAS 模型是一种对以节点为中心的图计算编程模型的一种细粒度改造,它将数据抽象成 graph 结构,并在 scatter 过程中可以直接对边数据进行更新并通知对其有依赖的邻接顶点更新状态;另一方面,GraphLab 提供 GDT(global data table),使节点和边可以通过访问 GDT 访问全局参数,然后读写邻域内的节点或边。这些特性可以使 k-truss 在计算边的支持度时有较高的效率。

基于以上,选择 GraphLab 框架来实现 GASTruss 算法。其基于 GAS 编程模型 truss 分解算法的基本思路是:gather 阶段,工作节点 v 收集邻居节点的 id,mirror 将收集的结果发送给 master 进行汇总,所有机器保证完成通信并保持同步;apply 阶段,工作节点将 gather 阶段汇总的邻居节点信息储存为节点数据,master 将更新后的节点数据发送给所有的 mirror,mirror 接收新的节点数据进行数据更新;scatter 阶段,对于图 G 中的每一条边 e,计算其两个顶点的邻居节点的交集大小作为自身的支持度,删除 G 中支持度小于 $(k-2)$ 的边,完成一次完整的迭代步骤。每次迭代后 G 中剩余的边的集合即构成一个 k-truss。若 G 中边还没有被完全删除,则 $k\leftarrow(k+1)$,进行下一轮迭代,计算 $(k+1)$-truss;否则,迭代结束。

算法 3 GASTruss 算法

输入:网络 $G=(V_G,E_G)$。
输出:所有的 k-truss,$3\leqslant k\leqslant k_{max}$。
//gather 阶段,获取节点的邻居信息
1 $k\leftarrow 3$;
2 for each $v\in V_G$ do
3 for each $u\in nb(v)$ do
4 gather. $v=u$. id;
5 同步以保证所有机器均完成通信;
//apply 阶段,更新节点数据
6 for each $v\in V_G$ do
7 v. data( ). vid_set. assign(neighborhood. vid_vec);
8 同步以保证所有机器均完成通信;
//scatter 阶段,计算节点支持度,删除支持度不足节点
9 for each $e\in E_G$ do
10 e. data( ) + count_set_intersect(targetlist. vid_set, srclist. vid_set);
11 if(e. data( ) < $(k-2)$)
12 remove e from G;
13 同步以保证所有机器均完成本轮迭代;
14 output G as the k-truss;
15 if(not all edges in G are removed)
16 $k\leftarrow(k+1)$;
17 go to step 2;

4 实验评估

4.1 实验环境

实验均使用三台处理器服务器,服务器配置为:英特尔酷睿 i3-3240 双核,主频为 3.40 GHz,内存为 4 GB,操作系统为 Ubuntu14.04,Graphlab2.2,OpenMPI1.6,编译环境为 GCC 4.8,编程语言为 C++。

4.2 测试数据集

实验选用 Zachary 数据集对算法结果正确性进行验证,并选用斯坦福大学网络分析平台 SNAP[22] 提供的 com-DBLP、com-Youtube 和 RoadNet-TX 三个数据集对算法效率进行验证。其中 com-DBLP、com-Youtube 是两个稠密的图数据集,图中每个节点都拥有较大的度,图拥有较多的三角形个数;RoadNet-TX 是拥有节点个数最多的数据集,拥有超过 137 万个节点和 192 万条边,但是因为图中节点的平均度数小于 3,可以认为它是一个稀疏的图数据集。表 2 列举了三个数据集的统计信息,表的每一行分别表示图中节点和边的数目($|V_G|$ 和 $|E_G|$)、三角形个数、图中边 truss 数的最大值(k_{max})以及节点支持度的最大值(sup_{max})。

表 2 实验数据集

数据集	$\|V_G\|$	$\|E_G\|$	三角个数	k_{max}	sup_{max}
com-Youtube	1 134 890	2 987 624	3 056 386	19	4 034
com-DBLP	317 080	1 049 866	2 224 385	114	312
RoadNet-TX	1 379 917	1 921 660	82 869	4	3

4.3 实验结果分析比较

首先使用 Zachary 数据集对 GASTruss 算法的正确性进行验证,算法运行结果如图 4 所示;其次,将 GASTruss 算法的性能与目前性能最好的基于 MapReduce 的方法 GPTruss[17] 进行比较,并针对每个计算节点设置不同进程数对 GASTruss 算法进行了效率对比,实验结果如图 5 所示。

图 4 中显示了 GASTruss 算法在 Zachary 数据集上的运行结果。图 4(a)表示 Zachary 网络,(b) ~ (d)分别表示网络中的 3-truss、4-truss 以及 5-truss(图中 $k_{max}=5$)。从图中可以看出,GASTruss 算法的结果可以遍历图中所有节点并得到网络

中的全部 truss 结构。

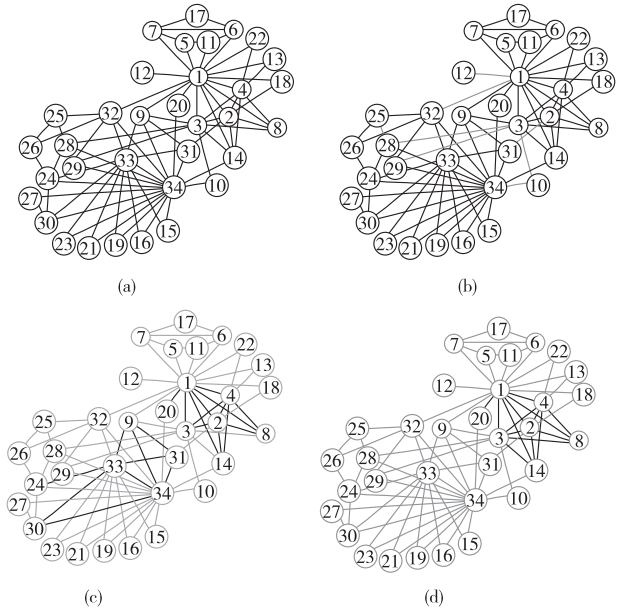


图4　Zachary数据集GASTruss运行结果

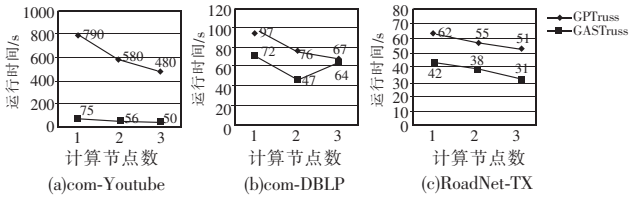


图 5　在三个数据集上 GPTruss 和 GASTruss 运行时间

从图 5 中可以看出,GASTruss 效率明显高于 GPTruss。图 5(a)中,图中节点的支持度最高达到 4 304,此时 GASTruss 算法加速比可以达到 9.6 以上,说明处理稠密以及依赖关系较大的数据时,GASTruss 的效率要远高于 GPTruss。从图 5(b)和(c)可以看出,在处理相对稀疏的数据时,GASTruss 算法与 GPTruss 的效率相差不大,但是 GASTruss 的加速比仍达到 1.04 ~ 1.65。另外从图 5 中可以看出,GASTruss 算法的运行效率受图 G 的规模影响不大,证明算法对于数据规模具有良好的拓展性。

图 5 中下方的三条折线表示 GASTruss 的运行时间。从图中可以看出,随着计算节点的增多,算法效率总体有所提升。当计算节点从 2 个增加到 3 个后,图 5(a)中算法的运行时间减少不明显,图 5(b)甚至有所增加,这是由于 com-Youtube 和 com-DBLP 两个数据集数据比较稠密、节点支持度较大的特点,算法运行过程数据更新的频率较大,此时算法运行过程中计算节点之间的通信频次明显增加,通信时间占比重较大,计算节点的增多反而会导致运行时间增加。而图 5(c)中随着计算节点的增多算法运行时间明显减少,这是因为 RoadNet-TX 数据集数据比较稀疏,图中边最大 truss 数以及节点最大支持度较小,在计算过程中计算节点之间的通信频次较少,计算时间占比重较大,所以计算节点的增多会降低算法的运行时间。这证明 GASTruss 算法在处理计算密集型数据时,优势更加明显。

5　结束语

本文基于 GraphLab 平台设计和实现了基于 GAS 模型的分布式 truss 分解算法 GASTruss。实验结果表明,该算法在运行效率方面比现有性能最好的基于 MapReduce 的分布式算法 GPTruss 有了明显的提升,同时算法对数据规模具有良好的拓展性,可以支持更大规模的图数据处理。下一步研究过程中可以进一步探讨 truss 结构在社区发现算法中的应用。

参考文献:

[1] Luce R D, Perry A D. A method of matrix analysis of group structure [J]. Psychometrika, 1949, 14(2): 95-116.

[2] Luce R D. Connectivity and generalized cliques in sociometric group structure[J]. Psychometrika, 1950, 15(2): 169-190.

[3] Seidman S B, Foster B L. A graph-theoretic generalization of the clique concept[J]. Journal of Mathematical Sociology, 1978, 6(1): 139-154.

[4] Cohen J. Trusses: cohesive subgraphs for social network analysis[R]. USA: National Security Agency, 2008: 16.

[5] Matsuda H, Ishihara T, Hashimoto A. Classifying molecular sequences using a linkage graph with their pairwise similarities[J]. Theoretical Computer Science, 1999, 210(2): 305-325.

[6] Pei Jian, Jiang Daxin, Zhang Aidong. On mining cross-graph quasi-cliques[C]//Proc of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining. New York: ACM Press, 2005: 228-238.

[7] Wang Jia, Cheng J. Truss decomposition in massive networks[J]. Proceedings of the VLDB Endowment, 2012, 5(9): 812-823.

[8] Hanneman R A, Riddle M. Introduction to social network methods [R]. Riverside, CA: University of California at Riverside, 2005.

[9] Seidman S B. Network structure and minimum degree [J]. Social Networks, 1983, 5(3): 269-287.

[10] Sariyuce A E, Pinar A. Fast hierarchy construction for dense subgraphs [J]. Proceedings of the VLDB Endowment, 2016, 10(3): 97-108.

[11] Rossi M E G, Vazirgiannis M. Exploring network centralities in spreading processes[C]//Proc of International Symposium on Web Algorithms. 2016.

[12] Malewicz G, Austern M H, Bik A J C, *et al*. Pregel: a system for large-scale graph processing [C]//Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2010: 135-146.

[13] Batagelj V, Zaversnik M. Short cycle connectivity [J]. Discrete Mathematics, 2007, 307(3-5): 310-318.

[14] Bron C, Kerbosch J. Algorithm 457: finding all cliques of an undirected graph[J]. Communications of the ACM, 1973, 16(9): 575-577.

[15] Abello J, Resende M G C, Sudarsky S. Massive quasi-clique detection [C]//Proc of the 5th Latin American Symposium on Theoretical Informatics. London: Springer-Verlag, 2002: 598-612.

[16] Cohen J. Graph twiddling in a MapReduce world[J]. Computing in Science & Engineering, 2009, 11(4): 29-41.

[17] Chen Peiling, Chou Chungkuang, Chen Mingsyan. Distributed algorithms for k-truss decomposition [C]//Proc of IEEE International Conference on Big Data. Piscataway, NJ: IEEE Press, 2014: 471-480.

[18] Latapy M. Main-memory triangle computations for very large (sparse (power-law)) graphs[J]. Theoretical Computer Science, 2008, 407(1): 458-473.

[19] Schank T. Algorithmic aspects of triangle-based network analysis[D]. Karlsruhe: University Karlsruhe, 2007.

[20] Low Y, Gonzalez J E, Kyrola A, *et al*. GraphLab: a new framework for parallel machine learning[EB/OL]. (2014-08-09). https://arxiv.org/abs/1408.2041.

[21] Low Y, Bickson D, Gonzalez J, *et al*. Distributed GraphLab: a framework for machine learning and data mining in the cloud[J]. Proceedings of the VLDB Endowment, 2012, 5(8): 716-727.

[22] Jure L, Krevl A. SNAP datasets: large network dataset collection[EB/OL]. http://snap.stanford.edu.

[23] Huang Xin, Cheng Hong, Yu J X. Attributed community analysis: global and ego-centric views[C]//Bulletin of the IEEE Computer Society Technical Committee on Data Engineering. Washington DC: IEEE Computer Society, 2016.