计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190900014

基于 FLINK 的滑动窗口内三角形计数算法研究

王 旭[1] 杨晓春[2]
1 北京理工大学计算机学院 北京 100081
2 东北大学计算机科学与工程学院 沈阳 110819
(yoo.wangxu@gmail.com)

摘 要 三角形计数旨在计算图中全局三角形和局部三角形的数量,是图数据挖掘中的一类重要工作。三角形的数量被广泛应用于角色识别、推荐系统、社区发现、垃圾邮件和欺诈检测等领域。在以流形式给出的图中,边具有时间性,同时现实生活中的图存在着大量的重复边。为充分利用图中的时间信息以挖掘网络知识,研究在多图流上计算滑动窗口内全局和局部三角形数量的问题,使用窗口机制同时研究多个窗口以利用隐含的时间关系获取更多信息。文中提出基于 FLINK 窗口操作的三角形计数算法和基于滑动窗口的三角形增量计数算法,以现有的边采样工作为基础,使用边集存储窗口历史数据实现一遍流计算,从而准确地计算面向多图流的滑动窗口内全局和局部三角形数量。基于 FLINK 窗口操作的三角形计数算法使用 FLINK 提供的窗口机制,基于滑动窗口的三角形增量计数算法,通过计算窗口滑入和滑出数据来实现窗口计数,避免了相邻两个窗口间重合边的大量重复计算,无缝地处理多个时间窗口,对于滑入和滑出数据中的重复数据,使用去重机制来进一步减小计算量。理论证明两种算法可以实现滑动窗口内三角形准确计数,并通过实验分析了窗口大小、滑动距离、数据分布和数据流速等因素对窗口处理时间的影响。与 TRIEST 算法相比,当窗口较小时,基于 FLINK 窗口操作的三角形计数算法和基于滑动窗口的三角形增量计数算法速度更快;当窗口较大时,保证了计算结果的准确性。

关键词:三角形计数;滑动窗口;FLINK;图流挖掘;准确流算法
中图法分类号 TP301.6

Study of Triangle Counting Algorithm with Sliding Windows Based on FLINK

WANG Xu[1] and YANG Xiao-chun[2]
1 School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China
2 School of Computer and Engineering, Northeastern University, Shenyang 110819, China

Abstract Triangle Counting, calculating global and local triangle counts, is an important work in data mining, whose number is widely used in important role identification, recommendation systems, community discovery, spam and fraud detection etc. In the graphs presented as a stream of edges, edges are temporal, and there are a large of duplicate edges in real-world graphs. For full use of the time information in the graph and mining the network knowledge, this work studied the problem of estimating global and local triangle counts on a multigraph stream with sliding windows, that simultaneously studied multiple windows by window mechanism to obtain more information in implicit time relationships. A triangle counting algorithm based on FLINK window operation and a triangle increment counting algorithm based on sliding window are proposed. Like the existing edge sampling work, the edge set is used to store window history data for accurately calculating global and local triangle counts on a multigraph stream with sliding windows in one-pass. The triangle counting algorithm based on FLINK window operator uses the window mechanism provided by FLINK. While the triangle increment counting algorithm based on sliding window realizes window counting by calculating slide-in and slide-out data through the window, reducing a large number of repeated calculations of coincident edges between adjacent windows, seamlessly processing multiple time windows, and for duplicate data in slide-in and slide-out, uses a deduplication mechanism to further reduce the calculation. The theory has been proven that both the algorithms can accurately count the triangles in the sliding window, and the effects of window size, sliding distance, data distribution and data flow rate on window processing time were analyzed through experiments. Compared with the TRIEST algorithm, when the window is small, the triangle counting algorithm based on FLINK window operation and the triangle increment counting algorithm based on sliding window have faster speed. When the window is large, the accuracy of the calculation result is guaranteed.

Keywords Triangle counting, Sliding window, FLINK, Graph stream mining, Accurate stream algorithm

到稿日期:2019-09-02 返修日期:2019-11-04 本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金(61572122,61532021)
This work was supported by the National Natural Science Foundation of China (61572122,61532021).
通信作者:杨晓春(yangxc@cse.neu.edu.cn)

1 引言

随着网络技术的快速发展，网络数据规模呈海量增长的趋势，其中蕴含了大量的可用信息。大数据时代，能否实现海量数据的有效挖掘与应用直接影响着劳动生产率的提高和消费者的福利满意度。目前，多种网络使用图作为它们的模型，如万维网、社交网络等，因此图数据挖掘成为了获取网络内蕴含信息的一种重要手段。

三角形计数是图数据挖掘中的一类重要工作，主要用于计算图中所有三角形(全局三角形)和各点组成的三角形(局部三角形)的数量。三角形是图中最小的非平凡的团，其数量被广泛应用于重要角色识别[1]、推荐系统[2]、社区发现[3]、垃圾邮件和欺诈检测[4]等领域。社交网络分析中的许多概念，如社会平衡、聚类系数和传递率等都是基于三角形的数量[5]。

很多大型图具有时间性，被表示为一系列边组成的边流，如网络中的分组传输、电话呼叫历史和金融交易等。在这些图中存在着交换信息、打电话、交易等交互行为，这些交互行为被模型化为图中的边，并具有一个自然时间戳[6]。

对于以流形式给出的图，其尺寸是未知的，甚至是无限的。在无限流中，聚合所有边来准确计算整个图上的三角形数量是不现实的。同时，为了充分利用图中的时间信息，挖掘网络知识，本文研究带有窗口的图三角计算问题，分析在多个有重叠的窗口内形成的三角形的数量及变化趋势。为解决这一问题，希望能设计出一种能够快速准确计算窗口内全局和局部三角形数量的“一遍”流式算法。

对于一系列按序到达的可重复无向边$e_1, e_2, \cdots, e_m$，给定窗口大小Δt_1和滑动距离Δt_2，准确计算多个窗口$[t, t+\Delta t_1]$，$[t+\Delta t_2, t+\Delta t_1+\Delta t_2]$，$[t+2\Delta t_2, t+\Delta t_1+2\Delta t_2]\cdots$内的全局和局部三角形数量。窗口大小可以用不同方式定义，可以是边数(如过去的1万条边)，也可以是时间(如过去一个月内的边)。对于每一个窗口，都可将其视为一张由窗口内的边聚合而成的图。该问题在多个不同的时间点计算过去窗口大小的边组成的图中全局和局部三角形数量。

图三角计算问题的流式算法较多，以往的流式算法将整个图看作一个窗口，窗口内包含了图中所有的边，是单一窗口内三角形计数。即使是滑动窗口，如何确定窗口参数也是一个很大的问题，通过观察来确定窗口参数是一种很好的方法，因此观察多种大小的窗口比观察单一窗口更重要。同时，可以观察滑动窗口间三角形数量随时间的变化，研究不同时间的三角形数量及变化情况，以获取更多的信息。

该问题的难点如下：1)边以流式输入，流过的边无法再次获得，而计算三角形时需要查找边与其他两条边组成的三角形，获取历史数据并进行计算。2)在多图中，边是可重复的并以不同频率出现，如 a,a,b,b,c,c,…和 a,b,c,c,a,b,c,…。对于以往的算法，多图流会造成计算结果的偏差，而在算法中不希望因此产生误差。3)如何删除滑出边组成的三角形，且破坏三角形后如何维护计数结果并保持其准确性？在窗口滑动过程中，伴随着数据的滑入和滑出，对于滑出数据，需要删除与滑出数据相关的三角形计数。如何准确删除并维护计数结果，对保证三角形计算结果的准确性至关重要。

为解决此问题，设计了两种基于 FLINK 的滑动窗口内图三角计数算法，并进行了理论和实验分析，在理论和实践上证明了两种算法能够实现窗口内三角形的准确计数。这两种算法对窗口内的某一时间 t，在边集中记录之前流过的属于窗口的边，利用边集实现在边流一遍流过的条件下快速准确地计算窗口内三角形的数量。我们首先提出一种基于 FLINK 窗口操作的滑动窗口内三角形计数算法(FLINK 是一个面向数据流处理和批量数据处理的分布式开源计算框架，支持高吞吐、低延迟、高性能的流处理)。该算法使用 FLINK 提供的 window 操作，但在该算法中存在着冗余计算和额外的资源消耗。为改善这种现象，本文提出了基于滑动窗口的三角形增量计数算法，该算法可以实现 window 操作的功能，同时避免了相邻两个窗口间重合边的大量重复计算；此外，该算法对传统滑动窗口进行了改进，如滑动计数窗口(30,5)每 5 个元素计算过去 30 个元素，传统滑动窗口每收集到 5 个数据就认为窗口结束输出计算结果，将而该算法中，每 30 个元素被认为是一个窗口的结束，输出计算结果，将 5 作为窗口滑动后的开始条件。图 1 给出了在实验数据集上不同窗口大小时的全局三角形数量。

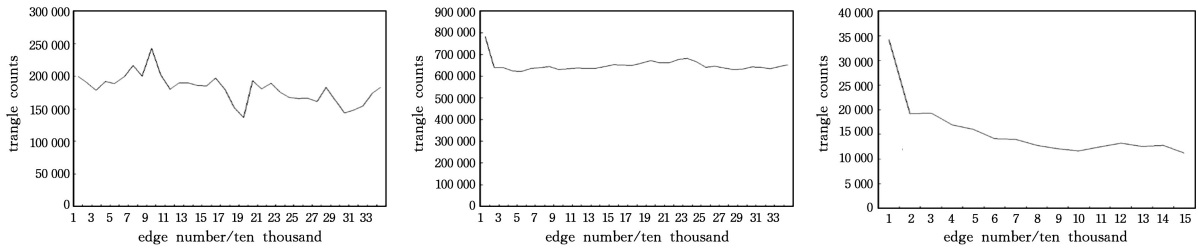


图 1 窗口参数为(100000,10000)时不同数据集上窗口内的全局三角形数量

Fig. 1 Global triangle counts in window on different datasets with window parameters (100000,10000)

本文第 2 节介绍了相关工作；第 3 节给出了本文中使用的符号和问题的定义；第 4 节描述了所提算法并给出了算法理论分析；第 5 节展示了实验数据和分析；最后总结全文。

2 相关工作

对于图三角计算问题，已经提出了很多流式算法。在假设有限存储空间内不能存储所有边的前提下，流算法通过采样近似计算三角形数量。在多种采样方法中，使用最多的是边采样[5-19]方法。关于边采样方法，Bera 等[7]提出了一种空间复杂度更低的近似随机采样算法。Lim 等[8]提出利用非采样边的均匀边采样方法 MASCOT，其核心思想是无论新接收到的数据是否被采样都需要更新计数值。FURL 算法[9]利用 MASCOT[8]的设计思想，在其基础上提高了准确率并将其推广到多图。WRS 算法[10]存储等候室内的最新边，对剩余边

进行标准池采样，利用时间局部性来提高准确率。文献[11-12]解决了有添加和删除边的动态图形流中的全局和局部三角形计数问题。Bulteau 等[11]了提出调整采样的方法，实现了在有恒定传递系数的图中每条边具有恒定处理时间。文献[13]采用分层采样来解决海量图流中计算稀疏图形的问题。PartitionCT[14]算法高速地均匀采样，并将每边的采样成本从 $O(\log k)$ 减少到 $O(1)$。Tri-Fly[15]，DiSLR[16]和 REPT[17]利用多台机器进行分布式流计算，相比单机流算法它们具有更高的准确度和速度。Lorenzo 等[18]提出了 TRIEST 算法，该算法使用池采样[20]和随机配对两种采样方案在有限存储内计算三角形数量。文献[19]也解决了在有限内存条件下有效计算三角形数量的问题。Tangwongsan 等[21]提出一种专为共享内存的多核机器设计的算法，该算法可以高效地利用并行性和内存层次结构。

除边采样方法外，还有楔形采样[6]和边点多采样[5]等采样方法。Jha 等[6]给出了一种可以处理重复边和时间窗口的三角形计数算法，该算法使用楔形采样方法对边进行采样，即每次对流入数据采样时同时采样相邻的两条边。当流中包含重复边，该算法也可以一遍估计多个时间窗的三角形计数。Kavassery-Parakkat 等[5]提出邻域采样算法，该算法通过边点多采样，即随机选取多个点和边并收集将采样点和采样边连接起来的边，来提高准确率。

流式算法大多用于近似计算，但其大方差特性导致了计算精度较低。对于一遍流算法，在边连续到达的图流模型中，仅允许一次实验，大的方差导致估计的数量和真实的三角形计数之间存在较大差异。在以流形式给出的图中，边具有时间性，而现有流算法倾向于聚合所有边以构建图，舍弃了时间信息。此外，现实生活中的图包含重复边，而一些流算法中假设输入为简单图，需要对现实数据进行过滤处理，因此消耗了额外的时间空间。本文研究在多图流上带有窗口的图三角形计算问题，设计了两种基于 FLINK 的滑动窗口内三角形计数算法，准确计算窗口内全局和局部三角形数量。

3　符号和问题定义

对于一系列按序到达的无向边 $e_1, e_2, \cdots, e_m$，e_t 表示时刻 t 到达的边，$t \in \{1,2,\cdots\}$，这些边可以是重复的，如 $e_2 = e_{59} = e_{123} = (u,v)$，$(u,v)$ 表示两个不同节点 u 和 v 之间的无向边。设置窗口大小 Δt_1 和滑动距离 Δt_2（$\Delta t_2 < \Delta t_1$），窗口大小可以用不同的方式定义，可以是边数（如过去的 1 万条边），也可以是时间（如过去一个月内的边），$G_t = (V_t, E_t)$ 表示窗口 $[t-\Delta t_1, t]$ 内的点和边组成的图。使用二元组 (u,c) 表示节点 u 及其局部三角形数量 c，当 $u = *$ 时，c 为全局三角形数量，结果集 $\delta_t = \{(u,c),\cdots\}$ 表示窗口 $[t-\Delta t_1, t]$ 内的三角形计算结果的集合。表 1 列出了文中使用的符号及其描述。

滑动窗口内三角形计数问题指，给定图流和窗口参数，计算多个窗口 $[t, t+\Delta t_1]$，$[t+\Delta t_2, t+\Delta t_1+\Delta t_2]$，$[t+2\Delta t_2, t+\Delta t_1+2\Delta t_2]\cdots$ 内的全局和局部三角形计数结果。

假设存在一个图形流(1,2)，(1,5)，(2,3)，(3,7)，(7,8)，(5,6)，(2,5)，(3,4)，(1,3)，(1,4)，(2,3)，(3,4)，(7,9)，(5,6)，给定窗口参数(10,2)，则输出为：(*,3) (1,3) (2,2) (5,1) (3,2) (4,1)；(*,2) (1,2) (3,2) (4,2)；(*,2) (1,2) (3,2) (4,2)（分号分隔开不同窗口的计数结果）。图 2 形象地展示了该问题的输入和输出。

表 1　符号表

Table 1　Notations

符号	描述
$(e_1, e_2, \cdots)$	图流
e_t	时间 t 到达的边
(u,v)	u 和 v 之间的无向边
Δt_1	窗口大小
Δt_2	滑动距离
$G_t = (V_t, E_t)$	窗口 $[t-\Delta t_1, t]$ 内的点和边组成的图
(u,c)	节点 u 及 u 的局部三角形数量 c 当 $u = *$ 时，c 为全局三角形数量
$\delta_t = \{(u,c),\cdots\}$	窗口 $[t-\Delta t_1, t]$ 内三角形计算结果集
$N[u]$	节点 u 的邻居集合
E	边集
ε_D	删除边集
ε_A	滑动边集
ε_{An}	新滑动边集
ε_{Dn}	新删除边集
ε_{AD}	动态滑动边集
$c[u]$	节点 u 的三角形计算个数
$c[*]$	全局三角形计算个数
$\lvert\Delta\rvert$	全局三角形个数
$\lvert\Delta[u]\rvert$	节点 u 的局部三角形个数
$\lvert\Delta t_1\rvert$	窗口大小为 Δt_1 时窗口内边的数量
$\lvert\Delta t_2\rvert$	滑动距离为 Δt_2 时滑动部分的边数

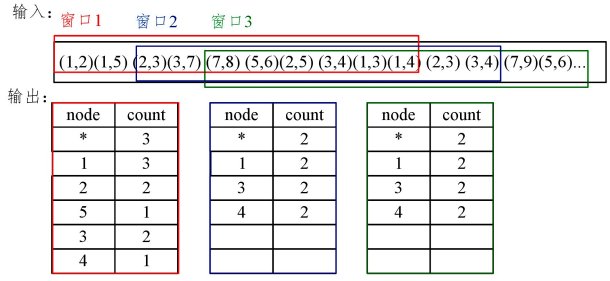


图 2　滑动窗口内三角形计数示意图（电子版为彩色）

Fig. 2　Sketch of triangle counting with sliding window

4　基于 FLINK 的滑动窗口内三角形计数算法

图流中含有重复边是三角计数的主要问题。在多图流中，边以不确定的频率和顺序出现。对于时间窗内三角计数问题，如何准确计算各窗口内的三角形数量，最简单的方法是先划分窗口数据，然后对每个窗口分别计算三角形数量，如先按节点对边进行分组，接着验证每组内任意两条边是否具有能形成三角形的第三条边。使用这种方法划分窗口数据时需要在内部缓冲数据元，直到窗口被认为已经准备好被处理，然后才能加载窗口内数据进行计算，因此会消耗更多的资源和时间，降低了计算性能。同时希望在流式数据上能“实时”计算三角形数量，但是该方法需要先获取数据进行离线处理，无法满足需求。因此，提出两种基于 FLINK 的滑动窗口内三角形计数算法。

4.1　基于 FLINK 窗口操作的三角形计数算法

基于近似流算法的设计思想，首先设计了一种基于 FLINK 窗口操作的滑动窗口内三角形计数算法（见算法 1）。该算法将近似流算法中的采样替换为保存操作，使用 FLINK

提供的 window 操作，在每个窗口到达时递增地聚合它们的数据元。数据元聚合时，查找共同邻居节点组成三角形，更新计算结果并保存数据。该方法在计算时，对每一个流入的数据进行计算，而不是聚合所有边后计算三角形，实现了三角形的增量计算。

算法 1 基于 FLINK 窗口的三角形计数算法

输入：$\{(e_1,e_2,\cdots),\Delta t_1,\Delta t_2\}$

输出：$\delta_t=\{(u,c),\cdots\}$

1. 创建窗口 w；
2. 初始化：$\varepsilon=\emptyset,\delta_t=\emptyset$；
3. for $e=(u,v)$：
4. ADDCOUNT(e,ε,δ)；
5. $\varepsilon\leftarrow\varepsilon\cup e$；
6. return δ_t.

该算法主要分为两步来创建窗口和计算：第一步使用 FLINK 提供的 window 操作构建窗口，该窗口可以是时间窗口或计数窗口，窗口长度可以是确定的边数或时间间隔（如天、年、月等），window 操作自动将符合窗口条件的数据划分到同一窗口内；然后，在窗口内执行窗口处理函数以计算三角形数量。对于每个共同邻居节点，全局和局部三角形数量值加 1，其处理过程如算法 2 所示。

算法 2 ADDCOUNT(e,ε,δ)

输入：(e,ε,δ)

输出：δ

1. for $n\in N[u]\cap N[v]$：
2. $(n,c)<=(n,c+1)$；
3. $(m,c)<=(m,c+1)$ /* $m\in\{u,v,*\}$ */；
4. return δ.

为解决流过的边不可再次获得这一问题，算法中使用边集记录历史数据，用于边计算时查找邻居节点，每个窗口各自维护一个边集，在窗口开始执行时初始化边集和结果集为空集。

更新计算结果时，可使用 *sum* 记录 e 组成的三角形个数，当 $sum>0$ 时更新全局和 u 和 v 的计数结果，避免多次更新操作，缩短处理时间。

4.2 基于滑动窗口的三角形增量计数算法

基于 FLINK 窗口的三角形计数可以简便地实现滑动窗口内的三角形计数，但其没有注意到相邻时间窗内存在大量重复边这一特点，忽略了窗口间的重合性，在每一个窗口内对重合数据分别进行计算，造成了大量计算冗余。

性质 1 滑动窗口相邻两窗口间具有大量重复数据，相邻窗口具有重复性。

性质 2 滑出数据和滑入数据间的相同数据不需要重复计算其在非滑动数据集内形成的三角形。

以相邻时间窗$[a,b,c,\cdots,f]$和$[c,\cdots,f,a,c]$为例，在删除$\{a,b\}$在窗口内形成的三角形并添加$\{a,c\}$形成的三角形的过程中，对于边 a，在删除和添加过程中都执行了一次在集合$\{b,c,\cdots,f\}$中查找邻居节点以更新计数的操作，而计数结果不变。

为充分利用相邻时间窗数据的重合性，对 FLINK 的窗口机制进行优化，通过计算相对窗口的滑入数据和滑出数据完成窗口内三角形的计算任务，取代原方案中对窗口内所有数据的计算处理。利用相邻窗口间数据的重合性，增量计算下一窗口内的三角形数量。同时考虑性质 2，在删除流出边添加流入边之前，更新由流出边组成的删除边集和由流入边组成的滑动边集，删除两个集合中的重复边并更新重复边相关三角形计数，对于重复边不执行在非滑动数据集上查找三角形以更新计数的操作，进一步减少计算量。由此设计了基于滑动窗口的三角形增量计数算法。

该算法中流数据上只有一个窗口，窗口通过滑动实现流上多个窗口的效果。在窗口滑动过程中，记录滑入和滑出数据，删除与滑出数据相关的三角形并计算滑入数据组成的三角形，将原窗口内全部数据的计算改进为对滑入和滑出数据的计算。在计算滑入和滑出数据之前，删除滑入数据和滑出数据间的重复边，进一步减少边的三角形计算任务量。基于滑动窗口的三角形增量计数算法如算法 3 所示。

算法 3 基于滑动窗口的三角形增量计数算法

输入：$\{(e_1,e_2,\cdots),\Delta t_1,\Delta t_2\}$

输出：$\delta_t=\{(u,c),\cdots\}$

1. 初始化：$\varepsilon=\emptyset,\delta_t=\emptyset,\varepsilon_A=\emptyset,\varepsilon_D=\emptyset,\varepsilon_{An}=\emptyset,\varepsilon_{Dn}=\emptyset,\varepsilon_{AD}=\emptyset$；
2. for $e=(u,v)$：
3. if $e\in[t-\Delta t_1,t]$：
4. ADDCOUNT(e,ε,δ_t)；
5. $\varepsilon\leftarrow\varepsilon\cup e$；
6. else if $e\in[t,t+\Delta t_2]$：
7. $\varepsilon_A\leftarrow\varepsilon_A\cup e$；
8. else：
9. $\varepsilon_D=\{e|e\in[t-\Delta t_1,t-\Delta t_1+\Delta t_2]\}$；
10. UPDATEAD()；
11. for edge in ε_D：
12. if edge ε_{Dn}：
13. DELCOUNT$(edge,\varepsilon,\delta_t)$；
14. $\varepsilon_{Dn}\leftarrow\varepsilon_{Dn}/edge$；
15. else DELCOUNT$(edge,\varepsilon_D,\delta_t)$；
16. $\varepsilon\leftarrow\varepsilon/edge$；
17. $\varepsilon_D\leftarrow\varepsilon_D/edge$；
18. for edge in ε_A：
19. if edge ε_{An}：ADDCOUNT$(edge,\varepsilon,\delta_t)$；
20. else ADDCOUNT$(edge,\varepsilon_{AD},\delta_t)$；
21. $\varepsilon_{AD}\leftarrow\varepsilon_{AD}\cup edge$；
22. $\varepsilon\leftarrow\varepsilon\cup edge$；
23. $\varepsilon_{An}\leftarrow\varepsilon_{An}\cup edge$；
24. return ε_{An}.

在基于滑动窗口的三角形增量计数算法中，对于流入数据分别判断其是否属于第一个窗口，是否收集完滑动距离内的数据。对于第一个窗口的数据，边流入时查找共同邻居节点组建三角形并更新计数结果。符合滑动距离条件的数据收集完成后，删除滑入和滑出数据的重复边，从而得到新删除边集和新滑动边集。对于删除边集中的边，若该边在新删除边集内，表示该边不是重复边，需要计算其在窗口数据内形成的三角形并将其删除；若该边不在新删除边集内，表示该边是重复边，已经被删除，此时计算其在删除边集中形成的三角形并将其删除。删除边集随着计算更新，移除已经处理完的边。

同理，对于滑动边集中的边，若该边在新滑动边集中，表示该边不是重复边，需要计算其在窗口数据内形成的三角形并添加计算结果；若该边不在新滑动边集中，表示该边是重复边，已经被删除，此时计算其在动态滑动边集中形成的三角形并更新计数结果。动态滑动边集是时间戳早于正在处理的滑动数据的边数据集合，其随着滑动数据的处理而更新。在数据处理过程中，边集被动态维护，随着对滑出数据和滑入数据的处理，窗口移动，边集记录窗口内数据。

函数 UPDATEAD()用于删除ε_A和ε_D中的重复边（见算法 4），首先初始化新滑动边集$\varepsilon_{An}=\varepsilon_A$，新删除边集$\varepsilon_{Dn}=\varepsilon_D$；对新删除边集中的边$e_t$，在新滑动边集$\varepsilon_{An}$内查找$e_{t'}=e_t$，如果找到，则新删除边集移除边$e_t$，新滑动边集移除$e_{t'}$。该查找操作只执行一遍，即对于新滑动边集内的多条边$e_{t1}=e_{t2}=e_{t3}=e_t$，只删除第一次查找到的边e_{t1}。

算法 4　UPDATEAD()

输入：(ε_A，ε_D)

输出：(ε_{An}，ε_{Dn})

1. 初始化：$\varepsilon_{An}=\varepsilon_A$，$\varepsilon_{Dn}=\varepsilon_D$；
2. for ed in ε_{Dn}：
3.　　if ed in ε_{An}：
4. ε_{An}←ε_{An}/ed；/ * 只执行一次 * /
5. ε_{Dn}←ε_{Dn}/ed；/ * 只执行一次 * /
6.　return ε_{An}，ε_{Dn}.

假设边 a，b，c 能组成三角形，且$t_a<t_b<t_c$，边 c 到达时组成三角形$\{a,b,c\}$，如图 3 所示。

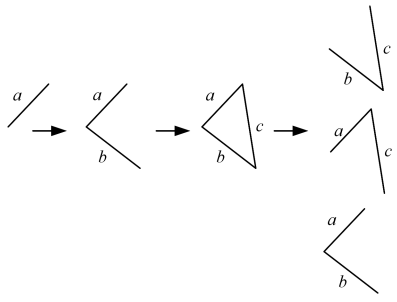


图 3　三角形形成与破坏过程

Fig. 3　Process of triangle formation and destruction

在滑动窗口中，当 a 要滑出时，删除其组成的三角形，在边集中查找共同邻居节点，找到与 b，c 组成的三角形，更新计数结果，移除 a。后续 b 滑出时，a 已被删除，不会再次找到与 a，c 组成的三角形，不会造成重复删除。对于三角形$\{a,b,c\}$，删除任一边即破坏了三角形，因此假设边 a，b，c 能组成三角形，且$t_a<t_b<t_c$，边 c 到达时组成三角形$\{a,b,c\}$，如图 3 所示。在滑动窗口中，当 a 要滑出时，删除其组成的三角形，在边集中查找共同邻居节点，找到与 b，c 组成的三角形，更新计数结果，移除 a。后续 b 滑出时，a 已被删除，不会再次找到与 a，c 组成的三角形，避免了造成重复删除。对于三角形$\{a,b,c\}$，删除任一边即破坏了三角形，因此在滑出边时需删除其组成的三角形，无论在三角形成的过程中边是以何顺序流入的。同时，在删除一条边时可能会对多个三角形造成破坏，因此在删除时需要删除所有相关三角形的计数结果，某一次错误删除都会造成最后的计算结果不准确。

例如，在处理滑入和滑出数据中重复边相关的三角形计数时，需谨慎处理其查找范围。计算重复边与滑入滑出数据组成的三角形时，根据重复边在滑动数据上的位置，需要准确处理边与滑动数据的关系。对于被删除数据，其位于删除边集时，需要计算其与删除边集中位于该边后的边组成的三角形；位于滑动边集时，需要计算其与滑动边集中位于该边前的边组成的三角形。DELCOUNT(e，ε，δ)删除计数函数的过程如算法 5 所示。

算法 5　DELCOUNT(e，ε，δ)

输入：(e，ε，δ)

输出：δ

1. for n∈N[u]⋂N[v]：
2.　　(n，c)<=(n，c−1)；
3.　　(m，c)<=(m，c−1) / * m∈{u，v，*} * /；
4. return δ.

4.3　理论分析

两种算法都能实现滑动窗口内三角形准确计数，即三角形计数算法给出的计算值等于三角形的实际数量，给出理论证明，并分析两种算法的时空复杂度。

4.3.1　准确性分析

定理 1　在任一时间窗内，基于 FLINK 窗口操作的滑动窗口内三角形计数算法给出的计算值等于三角形的实际数量，即 $c[u]=|\Delta[u]|$，$c[*]=|\Delta|$。

定理 1 的证明过程请见 OSID 码。

定理 2　在任一时间窗内，基于滑动窗口的三角形增量计数算法给出的计算值等于三角形的实际数量，即 $c[u]=|\Delta[u]|$，$c[*]=|\Delta|$。

定理 2 的证明过程请见 OSID 码。

4.3.2　复杂度分析

经过分析，给出两种算法的时空复杂度，如表 2 所列。表 2 中，$|\Delta t_1|$表示窗口大小 Δt_1的窗口内的边的数量，$|\Delta t_2|$表示窗口滑动部分边的数量。

表 2　时间和空间复杂度分析表

Table 2　Time and space complexity analysis table

算法	时间复杂度	空间复杂度
算法 1	$O(\lvert\Delta t_1\rvert^2.\lvert\delta\rvert)$	$O(\lvert\Delta t_1\rvert+\lvert\delta\rvert)$
算法 2	$O(\lvert\Delta t_1\rvert\cdot\lvert\Delta t_2\rvert.\lvert\delta\rvert)$	$O(\lvert\Delta t_1\rvert+5\lvert t_2\rvert+\lvert\delta\rvert)$

基于 FLINK 窗口操作的滑动窗口内三角形计数算法中，对窗口内的每一条边e_t，都要在边集ε_t中寻找共同邻居节点，随着边不断到来，边集不断扩大，最后达到窗口大小 Δt_1，寻找共同邻居节点过程中耗时应为$\sum_t(\varepsilon_t-1)$，对于每一个共同邻居，需在结果集中寻找其对应的计数结果，采用顺序查找耗时$|\delta|$，则基于 FLINK 窗口操作的滑动窗口内三角形计数算法的时间复杂度为 $O(|\Delta t_1|^2\cdot|\delta|)$。该算法只需记录窗口大小的边集和结果集，因此空间复杂度为 $O(|\Delta t_1|+|\delta|)$。基于滑动窗口的三角形增量计数算法中，耗时操作主要有调用 UPDATEAD()函数更新删除边集和滑动边集，以及新删除边集和新滑动边集内的边执行计数操作，因此时间复杂度为 $O(|\Delta t_1|\cdot|\Delta t_2|\cdot|\delta|)$。该算法需要保存除需记录窗口内的边集和结果集外，还需要保存删除边集和滑动边集、

新删除边集、新滑动边集和动态滑动边集，因此空间复杂度为 $O(|\Delta t_1|+5|\Delta t_2|+|\delta|)$。

5 实验

在 4 个数据集上对本文提出的两种算法进行测试实验，并对实验数据进行分析。从实践角度证明了两种算法均可实现滑动窗口内三角形数量的准确计算。

5.1 实验环境

实验时使用 JAVA 作为编程语言，实现了两种算法在计数滑动窗口内的全局和局部三角形计数，程序运行在配备 Intel Core i5-8500 CPU 3.00GHz×4 和 3.8GB RAM 的 Ubuntu 系统的虚拟机上，环境为 JDK1.8/ FLINK-1.7.2。选用 4 个数据集进行测试，这些数据集具有不同的点、边数量，具有点稀疏或边稀疏等特性，且数据集来自于不同的领域，数据集的详细信息如表 3 所列。

表 3 数据集

Table 3 Datasets

数据集	点	边	描述
BerkStan	685230	6649470	Web
Patent	3774768	16518947	Citation
Flickr	2302925	22838276	Friendship
Stackoverflow	2601977	63497050	Temporal

5.2 效果评估

将窗口视为由一定数量的过去边定义，使用多种窗口参数在数据集上进行实验，并对实验数据进行分析。在接下来的研究中，分析了窗口大小、滑动距离、数据分布和数据流速对窗口处理时间的影响，以及不同窗口大小对三角形数量的影响，研究了各数据集窗口内三角形数量的变化。

图 4 展示了 BerkStan 数据集上不同窗口参数时两种算法的窗口平均处理时间。可以看出，基于 FLINK 的滑动窗口内三角形计数算法的窗口平均处理时间随着窗口长度的增加而增加，但其对滑动距离不敏感，窗口大小一定且只改变滑动距离时处理时间变化不大，其变化原因是数据分布和流速影响了处理时间。基于滑动窗口的三角形增量计数算法的窗口平均处理时间随窗口长度的增加而增长，同时窗口大小一定时其处理时间随滑动距离的增大而增长。此外，可以看出，当 $\Delta t_2<0.3\Delta t_1$ 时，基于滑动窗口的三角形增量计数算法比基于 FLINK 窗口操作的三角形计数算法的窗口平均处理时间更短，对于滑动距离与窗口大小相差较大的应用，使用基于滑动窗口的三角形增量计数算法具有更好的执行效果。

图 5 比较了使用相同窗口参数时两种算法在不同数据集上的窗口平均处理时间，该实验采用窗口处理时间的平均值来消除数据流速对处理时间的影响。图 6 为当窗口参数为(10000,1000)时不同数据集上窗口处理时间的变化图。可以看到，使用相同窗口参数时，窗口处理时间随数据集和窗口位置，即窗口内数据发生变化，数据分布影响着窗口数据的处理时间。

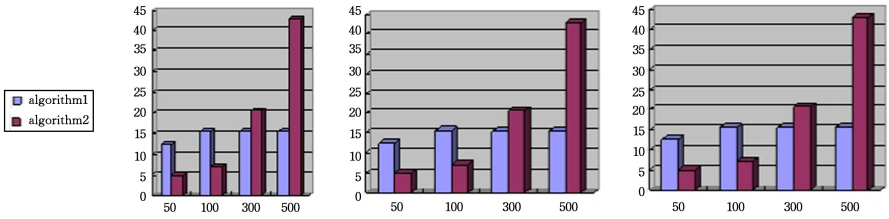


图 4 BerkStan 数据集上不同窗口参数时两种算法的窗口平均处理时间比较

Fig. 4 Comparison of average window processing time between two algorithms for different window parameters on BerkStan dataset

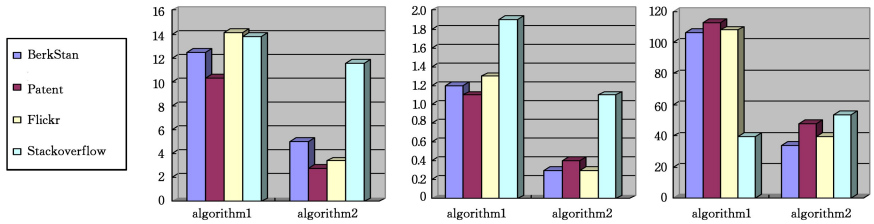


图 5 相同窗口参数时不同数据集上两种算法的窗口平均处理时间比较

Fig. 5 Comparison of average window processing time between two algorithms on different datasets with the same window parameters

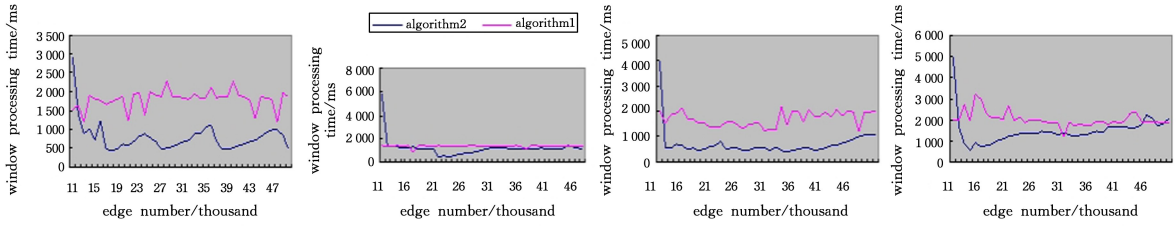


图 6 窗口参数为(10000,1000)时不同数据集上两种算法的窗口处理时间比较

Fig. 6 Comparision of window processing time between two algorithms on different datasets with the window parameters (10000,1000)

此外，本文给出了在 BerkStan 数据集上使用窗口参数(10000，1000)多次执行两种算法的窗口处理时间比较结果，分析了流速对窗口处理时间的影响。如图 7 所示，控制窗口参数、数据分布等其他因素不变，多次执行同一算法，同一窗口内相同数据的处理时间存在差异，可见流速对算法执行数据存在影响。

为了分析不同窗口大小对三角形数量的影响，关注窗口大小为 10 000，11 000，12 000，13 000，14 000，15 000 等且滑动距离为 1 000 时，窗口内全局三角形的数量，如图 8 所示。不难理解，窗口越大，窗口内三角形数量就越多。

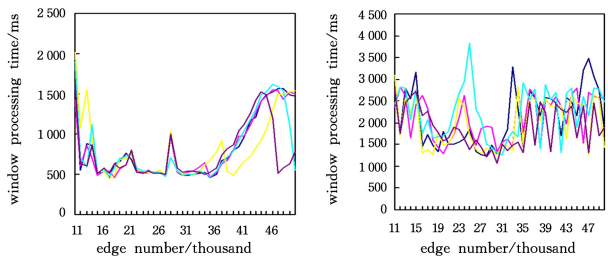


图 7　不同数据集上两种算法的窗口处理时间

Fig. 7　Processing time between two algorithms on different datasets

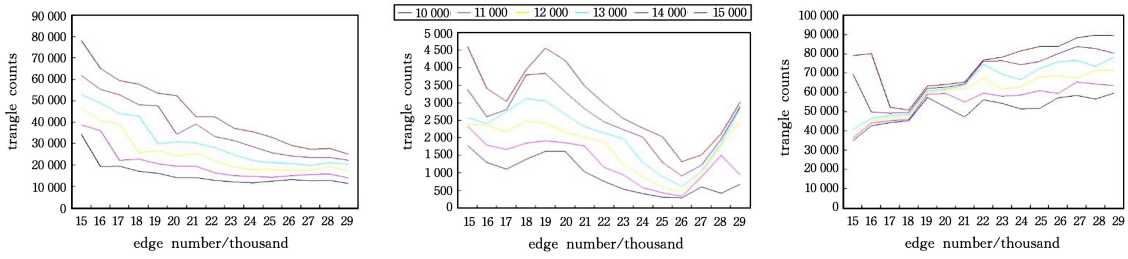


图 8　Stackoverflow，Flickr 和 BerkStan 数据集上不同窗口大小时的全局三角形个数

Fig. 8　Global triangle counts for varying window lengths for datatsets Stackoverflow，Flickr and BerkStan

在各数据集上，本文希望通过计算窗口内的三角形数量，来研究不同时间的三角形数量及变化情况，以了解三角形的趋势，获取更多的信息。图 1 给出了各数据集上窗口参数为(10 000，1 000)的滑动窗口内全局三角形的数量。在不同窗口内，BerkStan 数据集上的三角形数量较为稳定；Flickr 数据集上三角形数量存在多次上升下降的情况，且变化较大；Stackoverflow 数据集上三角形数量整体呈下降趋势。对于需要分析时间影响的应用，滑动窗口的实现具有重要意义。

5.3　本文算法与 TRIEST 算法的对比分析

将本文算法与 TRIEST 算法进行对比。TRIEST 算法是图三角形计数领域中一种先进的近似流式算法，该算法可以最大限度地利用存储空间进行三角形计算，当图规模小于其存储空间时，可以实现准确计算。为了能够与 TRIEST 进行对比，设置小的窗口参数，在 TRIEST 计算前，将数据集切分成窗口大小的多个数据集，TRIEST 在切分后的数据集上执行，以实现“窗口”计算的效果。此外，在分析较大的窗口参数时，本文算法与 TRIEST 在准确性上存在差异。通过分析可以看出，窗口参数较小时，本文算法和 TRIEST 都能实现窗口内三角形的准确计数，但本文算法的速度更快；当窗口参数较大时，TRIEST 只能实现近似计算，同时偏差较大，而本文中的算法 1 和算法 3 可以实现三角形的准确计数。

设置 TRIEST 的存储能力为 3 000，在数据集大小规模为 900 条边的图上运行 TRIEST 算法，记录 TRIEST 算法的执行时间。在切分前的 BerkStan 数据集上执行算法 1 和算法 3，记录窗口处理时间。图 9 给出了两种算法与 TRIEST 在“窗口”参数为(900，10)时窗口处理时间的比较结果，可以看到，除第一个窗口外，相比算法 1 和 TRIEST 算法，算法 3 具有十分明显的速度优势，能够更快计算出窗口内三角形的准确数量。

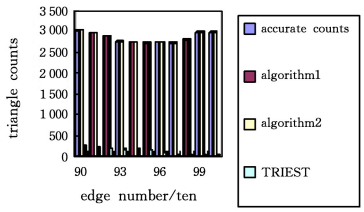


图 9　当窗口参数为(900，10)时两种算法与 TRIEST 算法的处理时间对比

Fig. 9　Comparison of window processing time between two algorithms and TRIEST with window parameters(900，10)

设置 TRIEST 的存储能力为 50，模拟数据集较大时 TRIEST 无法存储所有数据的情况。在该情况下，在多个数据集大小为 900 的图上执行计算，图 10 给出了本文提出的两种算法与 TRIEST 算法的计算能力的比较结果。可以看到，算法 1 和算法 3 都能实现窗口内三角形的准确计算，而 TRIEST 的准确性较低，其计算结果与真实结果之间存在较大偏差。

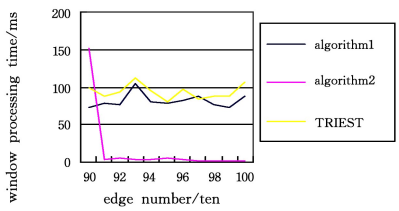


图 10　当窗口大小为 900、TRIEST 存储空间为 50 时两种算法与 TRIEST 计算能力的比较

Fig. 10　Comparison of computing performance between two algorithms and TRIEST when window size is 900 and storage of TRIEST is 50

结束语　本文提出了基于 FLINK 窗口操作的三角形计数算法和基于滑动窗口的三角形增量计数算法，两种算法均

能够准确计算多图流上全局和局部三角形数量。其可以实时准确地计算各种大小的流窗口，同时符合现实世界中图具有重复边的特性。仔细研究边代表的行为，了解三角形随时间的演变是一项有趣且有意义的工作。

参 考 文 献

[1] WELSER H T, GLEAVE E, FISHER D, et al. Visualizing the signatures of social roles in online discussion groups[J]. Journal of Social Structure, 2007, 8(2): 1-32.

[2] TSOURAKAKIS C E, DRINEAS P, MICHELAKIS E, et al. Spectral counting of triangles via element-wise sparsifification and triangle-based link recommendation[J]. Social Network Analysis & Mining, 2011, 1(2): 75-81.

[3] GLEICH D F, SESHADHRI C. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods [C]// 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York NY: ACM Press, 2012: 597-605.

[4] BECCHETTI L, BOLDI P, CASTILLO C, et al. Efficient semi-streaming algorithms for local triangle counting in massive graphs[C]// ACM Conference on Knowledge Discovery and Data Mining. New York NY: ACM Press, 2008: 16-24.

[5] KAVASSERY-PARAKKAT N, HANJANI K M, PAVAN A. Improved Triangle Counting in Graph Streams: Power of Multi-Sampling[C]// IEEE International Conference on Advances in Social Networks Analysis and Mining. Washington, DC: IEEE Computer Society Press, 2018: 28-31, 33.

[6] JHA M, SESHADHRI C, PINAR A. Counting Triangles in Real-World Graph Streams: Dealing with Repeated Edges and Time Windows[C]// 49th Asilomar Conference on Signals, Systems and Computers. Washington DC: IEEE Computer Society Press, 2015: 1507-1514.

[7] BERA S K, CHAKRABARTI A. Towards Tighter Space Bounds for Counting Triangles and Other Substructures in Graph Streams[C]// 34th Symposium on Theoretical Aspects of Computer Science. Dagstuhl, Germany, 2017, 11: 1-14.

[8] LIM Y, JUNG M, KANG U. Memory-efficient and accurate sampling for counting local triangles in graph streams: from simple to multigraphs [J]. ACM Transactions on Knowledge Discovery from Data, 2018, 12(1): 1-28.

[9] JUNG M, LEE S, LIM Y, et al. FURL: Fixed-memory and Uncertainty Reducing Local Triangle Counting for Graph Streams [J]. Data Mining and Knowledge Discovery, 2019, 33(5): 1225-1253.

[10] SHIN K. WRS: Waiting Room Sampling for Accurate Triangle Counting in Real Graph Streams[C]// 2017 IEEE International Conference on Data Mining. Washington, DC: IEEE Computer Society Press, 2017: 1087-1092.

[11] BULTEAU L, FROESE V, KUTZKOV K, et al. Triangle Counting in Dynamic Graph Streams[J]. Algorithmica, 2016, 76(1): 259-278.

[12] SHIN K, KIM J, HOOI B, et al. Think Before You Discard: Accurate Triangle Counting in Graph Streams with Deletions[C]// Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Lecture Notes in Computer Science. 2018: 141-157.

[13] DE STEFANI L, TEROLLI E, UPFAL E. Tiered Sampling: An Efficient Method for Approximate Counting Sparse Motifs in Massive Graph Streams[C]// 2017 IEEE International Conference on Big Data. Washington, DC: IEEE Computer Society Press, 2017: 776-786.

[14] WANG P, QI Y, SUN Y, et al. Approximately counting triangles in large graph streams including edge duplicates with a fixed memory usage[J]. Proceedings of the VLDB Endowment, 2017, 11(2): 162-175.

[15] SHIN K, HAMMOUD M, LEE E, et al. Tri-Fly: Distributed Estimation of Global and Local Triangle Counts in Graph Streams [C]// Pacific-Asia Conference on Knowledge Discovery and Data Mining, Lecture Notes in Computer Science. 2018: 651-663.

[16] SHIN K, LEE E, OH J, et al. DiSLR: Distributed Sampling with Limited Redundancy For Triangle Counting in Graph Streams [J]. arXiv: 1802.04249v1, 2018.

[17] PINGHUI W, PENG J, YIYAN Q, et al. A Streaming Algorithm of Approximating Global and Local Triangle Counts in Parellal[C]// 35th IEEE International Conference on Data Engineering. Washington, DC: IEEE Computer Society Press, 2019: 758-769.

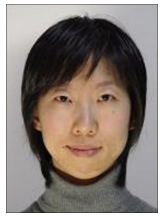
[18] LORENZO D S, ALESSANDRO E, MATECO R, et al. TRIEST: Counting Local and Global Triangles in Fully-dynamic Streams with Fixed Memory Size[J]. ACM Transactions on Knowledge Discovery from Data, 2017, 11(4): 1-50.

[19] JHA M. Counting Triangles in Graph Streams[J]. Encyclopedia of Algorithms, 2016: 458-464.

[20] VITTER J S. Random sampling with a reservoir [J]. ACM Transactions on Mathematical Software, 1985, 11(1): 37-57.

[21] TANGWONGSAN K, PAVAN A, TIRTHAPURA S. Parallel Triangle Counting in Massive Streaming Graphs[C]// ACM international conference on Information and Knowledge Management. New York NY: ACM Press, 2013: 781-786.

WANG Xu, born in 1997, postgraduate, is a member of China Computer Federation. Her main research interests include data stream processing and so on.



YANG Xiao-chun, born in 1973, professor, doctoral supervisor, is a member of China Computer Federation. Her main research interests include database theory and systems, data quality processing, text and data stream processing, and AI enabled data systems.