软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(9):4349−4372 [doi: 10.13328/j.cnki.jos.007256] [CSTR: 32375.14.jos.007256]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

高精度滑动窗口模型下的图流三角形近似计数算法*

苟向阳[1,2], 邹 磊[2], 于 旭[1]

[1](香港中文大学 系统工程与工程管理系, 香港 999077)
[2](北京大学 王选计算机研究所, 北京 100871)
通信作者: 邹磊, Email: zoulei@pku.edu.cn

摘 要: 近年来, 图流分析在研究领域和工业领域都变得愈发重要. 图流是从数据源持续高速到达的边序列, 这些边组成了一个不断变化的动态图. 在图流上可以进行多种不同的分析, 而三角形计数是其中最基础的操作之一. 由于图流数据规模大, 更新速度高, 在图流上进行精确三角形计数效率较低, 而且并不必要. 因为大部分三角形计数应用都允许一定的误差, 所以, 图流上的近似三角形计数一直都是研究热点之一. 研究基于采样的滑动窗口模型下的图流近似三角形计数. 滑动窗口模型只关注最近到达的图流数据, 较早的图流数据被认定为过期. 它被广泛应用于不同的工业场景和研究工作中. 将一种“采样前计数”的方法与该问题场景下最新的算法结合, 并提出一套策略以应对由于边过期产生的困难. 使用真实数据集展开广泛的实验以测试提出的 CBS 算法. 实验结果表明, CBS 相比目前最好的工作, 估算误差降低了 70% 以上.

关键词: 图流; 三角形计数; 近似算法
中图法分类号: TP391

中文引用格式: 苟向阳, 邹磊, 于旭. 高精度滑动窗口模型下的图流三角形近似计数算法. 软件学报, 2025, 36(9): 4349–4372. http://www.jos.org.cn/1000-9825/7256.htm
英文引用格式: Gou XY, Zou L, Yu X. Approximate Graph Stream Triangle Counting Algorithm Based on Sliding Window Model with High Accuracy. Ruan Jian Xue Bao/Journal of Software, 2025, 36(9): 4349–4372 (in Chinese). http://www.jos.org.cn/1000-9825/7256.htm

Approximate Graph Stream Triangle Counting Algorithm Based on Sliding Window Model with High Accuracy

GOU Xiang-Yang[1,2], ZOU Lei[2], YU Xu[1]

[1](Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong 999077, China)
[2](Wangxuan Institute of Computer Technology, Peking University, Beijing 100871, China)

Abstract: In recent years, graph stream analysis has gained increasing importance in both research and industry. A graph stream is a continuous sequence of edges received from a data source at a high speed. Those edges form a dynamic graph that is continuously changing. Various analyses can be performed on graph streams. Among them, triangle counting is one of the most basic operations. However, the large volume and high update speed of graph streams make it inefficient to count triangles accurately on them. It is also unnecessary, as most applications for triangle counting can tolerate small errors. Therefore, approximate triangle counting in graph streams has been a hot research topic. This study focuses on sample-based approximate triangle counting in graph streams with a sliding window model. Sliding window models focus on the most recent edges in a graph stream and consider older edges as expired. They are widely applied in various industrial scenarios and research. This study combines a count-before-sample strategy with the state-of-the-art approximate triangle counting algorithm and designs a set of novel strategies to deal with the difficulty brought by edge expiration. Extensive experiments are conducted on real-world datasets to evaluate the proposed algorithm. Results prove that the algorithm decreases

* 基金项目: 国家自然科学基金 (61932001, U20A20174); 香港研资局优配研究金 (14205520)
收稿时间: 2024-01-30; 修改时间: 2024-05-05; 采用时间: 2024-07-08; jos 在线出版时间: 2024-12-31
CNKI 网络首发时间: 2025-01-02

the estimation error of the state-of-the-art method by more than 70%.

Key words: graph stream; triangle counting; approximate algorithm

图是一种用于表示大规模实体和实体之间关联关系的数据形式, 被广泛应用在社交网络, 知识图谱, 电子商务等领域. 近年来, 随着互联网的不断发展, 图数据的动态性不断提高. 在很多应用中, 图数据被组织为图流的形式. 图流是一个持续到达的数据项组成的序列, 每个数据项表示图中的一条边, 以及边上的时间戳等信息, 这些数据项组成了一个持续变化的动态图. 例如, 在社交网络中, 用户通信数据便组成了一个持续更新的图流. 其中每一个点代表一个用户, 而每一条边表示两个用户之间的一次通信, 而边上的时间戳标识了通信发生的时间. 这些通信数据不断产生, 便组成了一个持续到达的边序列. 再例如, 网络通信中, 每个数据包可以被建模为以源 IP 和目的 IP 为顶点的一条边, 而这些数据包便组成了一个持续不断的图流. 图流数据具有更新速度快, 数据规模大的特点. 在大型社交网络, 例如 Twitter 中, 每秒产生的通信数据可以达到 1 万条以上, 而图流数据总规模可以达到数亿甚至数十亿[1], 而在大型数据中心, 每个路由器每秒需要处理数十万的数据包[2].

虽然图流数据是持续不断到达的, 实际应用往往只关注最近的数据. 对这些数据的分析有助于跟进最新的热门话题或者及时发现刚刚发生的异常行为. 因此, 很多应用使用滑动窗口模型[3]以抽取最近一段时间的数据, 例如时间戳在最近 72 h 内的边, 将其视为有效. 而更早的数据则被视为过期. 例如, 根据此前的研究工作[4], 阿里巴巴公司使用滑动窗口模型维护最近 72 h 内的电商交易数据, 在其中执行查询算法以实时检测洗钱行为. 而更早的数据则被不被纳入分析.

三角形计数是最常见的图数据分析方法之一, 在社区发现[5], 话题检测[6], 异常检测[7]等问题[8–11]中都发挥着重要作用. 在图流场景下, 需要在图流数据不断更新的过程中实时维护三角形数目. 由于图流数据的更新速度很高, 精确三角形计数的速度难以满足应用需求. 另一方面, 精确算法要求在内存中完整保留图流数据, 在图流规模达到数亿甚至 10 亿的情况下, 这对服务器是一个沉重的负担. 而且一些应用场景需要在路由器和交换机等嵌入式设备中进行网络测量[12], 这些设备中的内存资源稀缺, 使得精确算法无法部署. 高内存占用也限制了使用 GPU, ASIC 等新硬件加速的可能性. 此外, 三角形计数的相关应用都可以允许一定的误差, 这使得图流上的近似三角形计数称为近年来的研究热点之一. 本文将研究滑动窗口模型下的图流上的三角形近似计数算法.

现有的图流三角形近似计数算法[13–15]往往基于采样估算, 也就从图流中采样一部分的边组成一个样本图, 基于样本图中的三角形数目来估算完整图流中的三角形数目, 而没有被采样的边将不被保存. 目前最新的滑动窗口模型下的三角形近似计数算法是 Gou 等人[16]提出的 SWTC 算法, 该算法使用限定大小的存储空间维护一个滑动窗口中的无偏样本图, 并对滑动窗口中的完整边数进行估算, 之后根据估算出的完整边数和样本图的大小, 将样本图中的三角形数目按比例放大以估算滑动窗口中的三角形数目.

本文进一步提出将一种“采样前计数”的策略与 SWTC 算法结合以提升其精确度, 提出了新的滑动窗口模型下的图流估算方法 CBS. 采样前计数策略目前已被广泛应用于全动态模型下的三角形近似计数算法[17–19]. 与滑动窗口模型不同, 全动态模型假设图流中包含边添加和边删除两种操作, 前者在图流数据中加入新边, 而后者删除一条已有的边, 所有未被删除的边均被认为有效. 采样前计数策略根据图流中每一条新加入的边在样本图中形成的三角形数目, 估算其在完整图流中形成的新三角形的数目, 之后再决定是否采样该边. 通过累计每一条新边产生的新三角形数目, 算法可以得到完整图流中三角形总数的估算值. 对于图流中的边删除操作, 算法使用类似的方法估算该次删除导致的图流中三角形减少的数目, 将其从三角形总数中减去. 相比仅使用样本图中三角形数目的估算方法, 采样前计数策略充分利用了未被采样的边的拓扑信息, 具有更高的估算精度.

然而, 将采样前计数策略从全动态模型迁移到滑动窗口模型时, 会在处理边过期时出现困难. 滑动窗口模型的边过期和全动态模型的边删除存在差别. 全动态模型中的删除是显式的, 每一次边删除都来自图流中明确的指令告知. 因此算法可以得知何时删除了哪条边. 而滑动窗口模型中的边过期是隐式的, 算法在图流中只会收到新添加的边, 而这些边具有自己的时间戳. 随着时间推移, 当一条边的时间戳不再处于滑动窗口中时, 它便会过期. 这样的过期不会有指令告知. 图流数据是被实时处理的, 一条边到达时若未被采样, 便不会被保存. 对于这样的边, 我们无

© 中国科学院软件研究所 http://www.jos.org.cn

法在它过期时及时发现. 采样前计数策略要求在每次边删除时估算该边导致的三角形数目减少值, 无论被删除的边是否被采样, 这一点在滑动窗口模型下难以实现.

为了处理这一问题, 本文设计了基于时间区间划分的策略来处理数据过期. 本文为图流中的每一个三角形赋予一个时间戳, 其被定义为三角形 3 条边中的最小值. 此外, 本文定义 3 条边时间戳的最小值和最大值相差不超过 N 的三角形为有效三角形, 其中 N 为滑动窗口的长度. 本文证明了在任意时刻, 滑动窗口中的边组成的三角形集合等同于时间戳落入滑动窗口的有效三角形集合. 本文将时间轴划分为长度为 N/d 的区间, 在图流更新的过程中使用采样前计数策略估算时间戳落入每一个区间内的有效三角形数目. 之后将与滑动窗口相交的区间中的有效三角形数目之和 τ 作为滑动窗口中的三角形的数目的估算值. 这一策略通过区间切换实现了的过期处理. 每当时间推移到一个区间分割点时, 就会有一个旧的区间不再与滑动窗口相交, 而时间戳在该区间中的有效三角形全部已经过期, 其数目将被从估算值中减去. 由于与滑动窗口相交的区间的并集是滑动窗口的超集, τ 的期望值大于滑动窗口内三角形数目的真实值. 为了实现无偏估算, 我们又提出误差修正策略 ETM (expired triangle-count modification). 该策略根据这些区间内的过期样本边的拓扑信息, 估算出 τ 和滑动窗口中的三角形的数目的差值, 将其从 τ 中减去以保持估算的无偏性. 最终 CBS 算法在保证无偏估算的基础上, 达到比现有算法 SWTC 更高的精确度. 实验证明, CBS 算法的估算误差比 SWTC 下降了 70% 以上.

总结来说, 本文有以下贡献点.

(1) 提出滑动窗口上固定空间消耗的三角形近似计数算法 CBS, CBS 将现有的 SWTC 算法和采样前计数策略结合, 并且设计基于时间区间划分的策略来解决采样前计数不支持数据过期的问题.

(2) 进一步设计误差修正策略 ETM, 该策略针对 CBS 算法的估算期望值和滑动窗口内三角形数目真实值之间的偏差进行进一步修正, 使得 CBS 算法成为无偏估算算法.

(3) 在真实数据集对 CBS 算法进行实验评估, 实验结果表明 CBS 算法相比现有算法在相同空间消耗下误差降低 70% 以上.

本文的第 1 节将给出研究问题的正式定义. 第 2 节将介绍图流三角形近似计数的相关工作. 第 3 节将具体介绍 CBS 算法. 第 4 节中将给通过实验对比 CBS 算法与现有工作以验证其优势. 最后在第 5 节总结全文.

1 研究问题定义

● 图流 (graph stream): 图流是一个从数据源持续不断到达的数据项组成的序列 $S = \{e_1, e_2, e_3, \ldots\}$, 每一个数据项包含一条边 (v_i, v_j) 以及该数据项到达的时间戳 t.

图流中的边可以为有向边或者无向边, 在三角形计数问题中一般不考虑边的方向, 所以本文假定图流中的边都是无向的. 在第 3.4 节中会讨论有向三角形计数问题. 需要注意的是, 图流的流量不是稳定不变的, 在有些时间点可能有多条边到达, 在有些时间点则可能没有任何边到达. 由于图流数据是持续到达的, 算法无法对完整的图流数据进行分析. 此外, 实际应用中一般更关注最新的数据. 因此, 大部分图流应用都使用滑动窗口模型划分出近期的图流数据, 将其视为有效, 而滑动窗口之外的数据被视为过期. 滑动窗口的具体定义如下.

● 滑动窗口 (sliding window): 假设当前时间为 T, 一个长度为 N 的滑动窗口 是图流中时间戳在 $(T-N, T]$ 区间内的边组成的集合.

滑动窗口内的边被称为有效边, 而滑动窗口之外的边被称为过期边. 此外, 我们称 T 时刻滑动窗口中的边组成的图为快照图 (snapshot graph). 随着时间推移, 图流中不断有新边被加入滑动窗口, 而滑动窗口中的旧边则因为过期而被删除, 导致快照图不断发生变化. 本文和之前的图流算法研究一样, 不考虑图流数据的乱序和延迟等问题. 也就是说, 假设图流中每条新边到达时, 其时间戳等于当前时刻 T, 且不小于之前到达的边的时间戳. 此外, 在下文中, 出于论述简单考虑, 使用正整数来表示图流中每条边的时间戳和滑动窗口的长度. 本文关注快照图上的三角形计数问题.

● 三角形计数 (triangle counting): 滑动窗口模型下的三角形计数要求在图流更新的过程中实时维护快照图中

© 中国科学院软件研究所 http://www.jos.org.cn

的三角形数目.

图 1 展示了一个图流、滑动窗口和快照图的示例, 其中滑动窗口长度为 6, 当前时刻为 13, 因此只有时间戳在 (7,13] 内的边会被包含在滑动窗口中. 在快照图中也标注了每条边的时间戳. 下文中, 在不引起歧义的情况下混合使用“滑动窗口中的三角形数目”与“快照图中的三角形数目”.

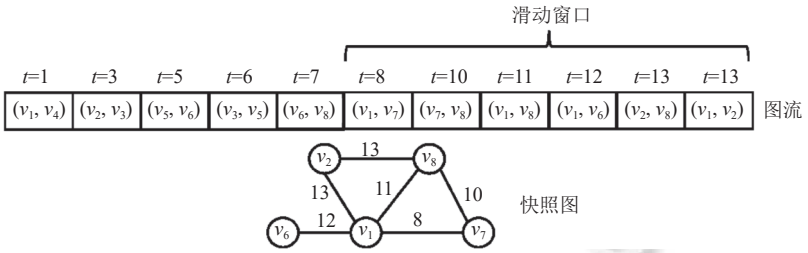


图 1 图流示例

在图流中, 同一条边 (v_i,v_j) 可能在不同时刻多次出现, 这种情况被称为重复边 (duplicate edge). 在三角形计数问题中, 有二项计数 (binary counting) 和带权计数 (weighted counting) 两种处理重复边的方法, 定义如下.

- 二项计数 (binary counting): 计数由不同边组成的互异三角形的数目, 相同两点间的多条重复边不会重复参与计数.
- 带权计数 (weighted counting): 计数包含重复边的所有三角形的数目, 相同两点间的重复边会被视为独立边, 可以组成不同的三角形.

图 2 展示了一个含重复边的快照图, 图中边 (v_1,v_8) 在滑动窗口中出现两次, 时间戳分别为 9 和 12. 边 (v_7,v_8) 也出现两次, 时间戳分别为 8 和 10. 若使用二项计数, 图中只有一个三角形. 而使用带权计数, 则有 4 个三角形, 分别为 $[(v_1,v_8)^1,(v_7,v_8)^1,(v_1,v_7)]$, $[(v_1,v_8)^1,(v_7,v_8)^2,(v_1,v_7)]$, $[(v_1,v_8)^2,(v_7,v_8)^1,(v_1,v_7)]$ 和 $[(v_1,v_8)^2,(v_7,v_8)^2,(v_1,v_7)]$, 其中 $(v_1,v_8)^1$ 和 $(v_1,v_8)^2$ 分别表示和间两条不同的重复边, $(v_7,v_8)^1$ 和 $(v_7,v_8)^2$ 同理.

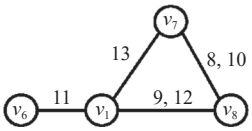


图 2 含重复边的快照图

本文使用带权计数方法. 一方面, 采样前计数策略不能与二项计数结合. 二项计数要求在计数三角形时过滤掉重复边, 而采样前计数策略需要对图流中每条新边进行三角形计数. 而我们无法判断这条新边是否为重复边. 使用采样前计数策略的现有工作也都采用了带权计数[17–19]. 另一方面, 带权计数在应用中也更加常见. 例如在社交网络中进行社区检测时, 用户之间重复通信的次数显然是寻找活跃社区的重要指标, 因此在计数三角形时, 应当采用考虑了重复通信 (重复边) 的带权计数.

由于带权计数不需要过滤重复边, 无论图流中是否存在重复边, 算法都不需要做出改变. 出于论述简单考虑, 下文不再考虑重复边问题, 假设图流中的边都是互异的. 但是在实验中, 也测试了算法在带有重复边的数据集上的效果.

此外, 本文中使用的采样前计数策略之前主要被使用于全动态模型. 在此也给出全动态模型的定义, 以便和滑动窗口模型进行区分.

- 全动态模型 (fully dynamic model): 在全动态模型中, 图流上的每一个数据项除了包含边 (v_i,v_j) 以及时间戳 t 外, 还包含一个操作 o ($o=+$ 或 $-$), 用于区分该数据项为边添加或者边删除. 当 $o=+$ 时, 表示加入一条新边, 而当 $o=-$ 时, 表示删除之前图流中出现的一条边. 在全动态模型中, 快照图中包含图流中目前所有未被删除的边.

全动态模型中的边删除是由新数据项触发的, 这样的删除被称为显式删除 (explicit deletion), 而滑动窗口中的

© 中国科学院软件研究所 http://www.jos.org.cn

边过期则是随着时间推移自发发生的, 是一种隐式删除 (implicit deletion). 这导致在全动态模型中, 算法总是能够知晓所有边删除, 而滑动窗口模型中, 若算法没有保存一条边, 则无法发现该边的过期. 这也导致了将采样前计数策略用于滑动窗口模型时的困难.

本文使用采样算法进行三角形数目估算, 首先给出采样算法中的几个重要概念: 样本边、样本集和样本图.

- 样本边: 图流中的每条新边到达时, 采样算法会判断是否保存该边, 如果该边被保存, 便被称为样本边. 反之, 如果这条边没有被采样算法保存下来, 则被称为非样本边. 由于图流是实时处理的, 到达时未被保存的非样本边后续无法再次被获取.

需要注意的是, 样本边也不是始终被保存的. 一方面, 样本边过期时会被算法删除, 另一方面, 近年来的三角形近似计数算法, 例如 SWTC, TRIEST 和本文 CBS 等, 是固定空间消耗的, 它们的采样算法会限制样本边的总数. 因此当一条新边被采样时, 另一条旧的样本边可能会被删除, 或者说旧样本边被新样本边替换了. 我们会在第 2.2 节中详细介绍 SWTC 算法如何替换样本边.

- 样本集: 任意时刻, 所有样本边组成的集合被称为样本集.
- 样本图: 任意时刻, 所有样本边组成的图被称为样本图.

2 相关工作

大规模图数据上的三角形计数问题已经被研究了数十年. 相比精确计数算法[20–24], 近似估算算法[25–28]在时间和空间上更加高效. 近年来在大规模图流数据上的三角形近似计数算法包括 Ahmed 等人[13]的通用图流采样和估算框架, Pavan 等人[14]基于邻居采样的估算算法, Jha 等人[29]基于双边 (wedge) 采样的估算算法以及 Tsourakakis 等人提出的基于固定概率采样的三角形估算算法[30]. 上述工作的一个主要缺陷是无法控制算法的空间占用, 也就是说算法消耗的内存会随着图流更新变化, 且无法给出上限预测. 而之后由 De Stefani 等人[17]提出的 TRIEST 算法可以将算法的空间占用控制在一个给定的阈值之内. TRIEST 使用了固定上限的蓄水池采样 (reservoir sampling), 并使用了随机配对 (random pairing)[31]策略来在蓄水池采样中支持边删除, 因此 TRIEST 适用于全动态模型. 然而, 由于随机配对策略需要知晓每一次边删除, TRIEST 不适用于滑动窗口模型. 因为滑动窗口模型下算法无法知晓非样本边的过期. 在该工作中, De Stefani 等人也首次提出了采样前计数策略, 但他们只将该策略运用在了不含边删除的场景下, 作为 TRIEST 的一种更加精确但是应用场景受限的变体. 后续的工作 ThinkD[18] 进一步提出了将采样前计数应用于含删除场景的方法, 提出了基于采样前计数且能够支持全动态模型的三角形估算算法. 后续工作如 WRS 等[19,32]则进一步对其进行了改进. 但是出于和 TRIEST 相同的原因, 这些算法都不适用于滑动窗口模型, 且这些算法只能处理带权计数. Wang 等人[15]提出具有固定空间占用且适用于二项计数的三角形估算算法 PartitionCT, 但是他们的方法既不能处理全动态模型中的显式删除也不能处理滑动窗口模型中的边过期. Gou 等人[16]提出的 SWTC 算法则提出了适用于滑动窗口模型的三角形估算算法, 该算法基于权重采样方法, 具有固定空间占用, 且既能处理带权计数又能处理二项计数. 近年来的最新工作也在研究了更多场景下的三角形近似计数问题, 例如, Zhang 等人[33]研究了超图图流上的三角形近似计数问题, 超图是传统结构的一种变体, 其中的每条边可以包含多个顶点. Yang 等人[34]研究了 Master-Worker-Aggregator 架构下的分布式场景下的三角形近似计数问题, 后续他们又进一步研究了云计算场景下的分布式三角形近似计算问题[35]. 此外, Wang 等人[36]将强化学习融入了采样算法中, 提出了用于图流上的子图计数的近似算法, 该算法也适用于三角形计数. 他们使用强化学习为图流中每条边赋予权重, 根据边权重进行带权采样, 权重越高的边在子图计数中的重要性越高, 被采样概率也越高. 之后他们根据采样出的样本集进行子图计数. 然而, 该研究工作不能处理边过期, 无法应用于本文研究的滑动窗口模型. 下文将对和本文密切相关的 SWTC 算法和采样前计数策略进行更详细的介绍. 此外, 我们也将介绍固定概率采样方法, 该方法是一种适用于滑动窗口模型的简单采样算法, 在多种已有工作中得到了应用. 但是其与最近几年的工作如 TRIEST 和 SWTC 相比, 具有无法控制空间占用的缺陷, 在空间紧张的情况下不易适用. 但是由于采样机制简单, 更新速度更高, 所以在空间较为宽裕的情况下可以作为替代方案. 在第 3.5 节中, 本文也将提出基于固定概率采样

© 中国科学院软件研究所 http://www.jos.org.cn

的 CBS 算法变体.

2.1 固定概率采样算法

固定概率采样算法 (fixed probability sampling) 即使用固定概率 r 对图流中的每一条边进行采样. 每当图流中有新边到达时, 算法运行一个满足概率为 r 的伯努利分布的随机函数, 当函数返回值为 1 时, 将新边加入样本集合, 否则便丢弃该边. 该采样方法被之前的许多三角形估算工作采用[28,30]. 可以使用固定概率采样维护样本图, 计数样本图中的三角形数目, 并将其乘以 $1/r^3$ 来估算快照图中的三角形数目. 固定概率采样可以适用于滑动窗口模型, 只需要实时删除样本集合中的过期样本边即可. 但是在滑动窗口模型下, 固定概率采样算法的样本集大小为 $r\times|W|$, 其中 $|W|$ 为滑动窗口中的总边数. 虽然滑动窗口在时间尺度的长度 N 是固定的, 但其中边数 $|W|$ 会随着图流的流量, 也就是单位时间到达的新边数量, 而发生较大改变. 这导致固定概率采样算法的样本集大小会随着图流流量而不断变化, 我们无法提前预测其上限. 因此, 也无法给出其空间占用的上限, 这会导致在空间紧张的应用场景中难以对内存分配进行规划.

2.2 SWTC 算法

Gou 等人[16]提出的 SWTC 算法是滑动窗口模型下具有固定空间占用的三角形近似估算算法. 在图 3 中给出了算法的框架图. 该方法基于优先级采样 (priority sampling)[37]. 它首先使用一个随机函数 $H(\cdot)$ 将图流中的每一条边划分到 K 个子流中的一个, 其中 K 为提前设定的参数. 之后 SWTC 使用另一个服从均匀分布的随机函数 $G(\cdot)$ 为每一条边 e 分配一个在 $(0,1)$ 内的优先级 $G(e)$, 之后, 算法尝试在每个子流中找到滑动窗口内优先级最高的边作为样本边. 最终这些样本边便组成了一个大小不超过 K 的样本集. 由于子流划分和优先级赋予都是随机的, 所以每条边都有相同的概率被采样, 该样本集是一个无偏的样本集. 通过将 $H(\cdot)$ 和 $G(\cdot)$ 替换为哈希函数, 算法也可以支持二项计数语义.

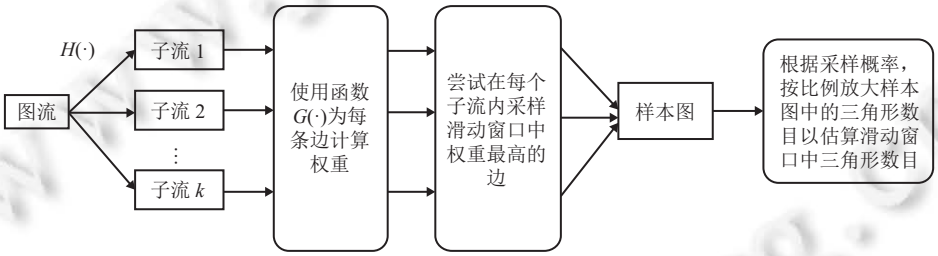


图 3 SWTC 算法框架

在该算法流程中, 主要的技术挑战有两点, 如何使用固定空间维护每个子流中在滑动窗口内的优先级最高的边, 以及如何计算滑动窗口中每个三角形被加入样本图的概率. 后者需要在根据样本图中三角形数目估算滑动窗口中三角形数目时使用, 而为了计算该概率, 主要挑战在于估算滑动窗口中的边数. 下面分别对这两点挑战及 SWTC 的解决办法进行介绍.

- 子流中样本边的维护: 具体来说, 该阶段的挑战来自样本边的过期. 在没有边过期时, 维护优先级最高的边较为简单. 只需要使用一个变量 ϵ 来维护目前优先级最大的边, 每当子流中有新边 e 到达时, 对比 $G(e)$ 和 $G(\epsilon)$, 并在 $G(e)$ 更高时使用 e 替换 ϵ. 然而, 当 ϵ 过期时, 如何选择其后继则会变得很困难. 子流中存在 ϵ 之后到达的边, 它们仍未过期, 但是由于优先级低于 ϵ 而没有被选为样本边, 算法没有保存它们, 也无法判断 ϵ 过期后滑动窗口中优先级最高的边是否在它们之间, 因此难以重新选出样本边. 举例来说, 在图 1 中, 假设图流中时间戳在 $(0,6]$ 内的 4 条边的优先级分别为 $G(v_1,v_4)=0.9$, $G(v_2,v_3)=0.6$, $G(v_5,v_6)=0.8$, $G(v_3,v_6)=0.2$. 在时刻 6, 滑动窗口内优先级最高的边是 (v_1,v_4), 算法将其保存为样本边, 而非样本边则没有被保存. 在时刻 7, 边 (v_1,v_4) 过期, 而算法没有保存滑动窗口内其他 3 条边, 因此无法找到当前滑动窗口中优先级最高的边. 在时刻 8, 边 (v_1,v_7) 到达时, 也无法确定其优先级和滑动窗口内其他 3 条边优先级的大小关系, 难以确定是否应当采样 (v_1,v_7).

© 中国科学院软件研究所 http://www.jos.org.cn

为了解决这一问题, SWTC 采用了图流切片的策略. 它选择一系列的路径点 (landmark), 将图流切分为长度为 N 的定长片段. 在任意时刻 T, 滑动窗口至多与最近的两个片段相交. 图 4 展示了一个示例. 当前时刻为 13, 滑动窗口为 (7,13], 和片段 2, 即 (6,12] 以及片段 3, 即 (12,18] 相交. 注意片段 3 是一个仍未结束的片段, 后续新到达的边仍会被加入该片段. 而等到 18 时刻, 滑动窗口将只与片段 3 相交.

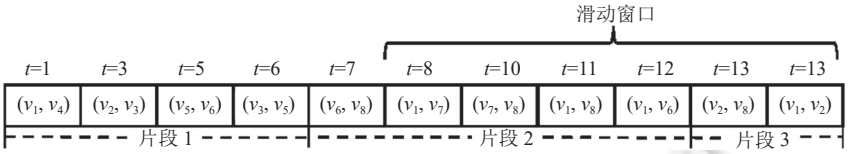


图 4 SWTC 图流切片示例

在任意时刻, SWTC 记录在每个子流中记录最近两个片段 SL_{old} 和 SL_{new} 中优先级最高的边 β 和 ϵ. 例如在图 4 中, 它保存片段 2 中优先级最高的边和片段 3 中目前优先级最高的边. 由于片段的起点和终点都是固定的, 在寻找其中优先级最高的边时不需要处理边过期, 只需要从片段起点开始使用变量 ϵ 记录目前优先级最高的边, 在每次新边 e 到达时对比 e 和 ϵ 的优先级, 并在 e 优先级更高时使用它替换 ϵ, 持续此操作直到片段终点. 对于 β 和 ϵ, 存在下述 3 种情况, 如图 5 所示.

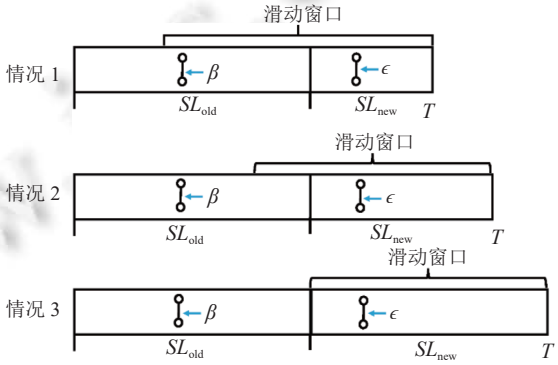


图 5 SWTC 子流采样的不同情况

情况 1: 滑动窗口与 SL_{old} 和 SL_{new} 同时相交, 且 β 和 ϵ 同时在滑动窗口中, 这时选择二者中优先级较高的边, 它一定是滑动窗口中优先级最高的边, SWTC 算法将其作为该子流中的样本边.

情况 2: 滑动窗口依然与 SL_{old} 和 SL_{new} 同时相交, 但是 β 已经过期. 若 $G(\beta) > G(\epsilon)$, SWTC 放弃在该子流中选择样本边. 因为片段 SL_{old} 中可能存在边 e, 其没有过期, 而且 $G(\beta) > G(e) > G(\epsilon)$. e 应当被选为样本边, 但是没有被保存. 若 $G(\beta) \leqslant G(\epsilon)$, SWTC 选择 ϵ 作为样本边, 因为片段 SL_{old} 中的边优先级都低于 $G(\beta)$, 因此也低于 $G(\epsilon)$. 而 ϵ 又是片段 SL_{new} 中优先级最高的边, 因此它是最近两个片段中优先级最高的边, 自然也是滑动窗口中优先级最高的边. SWTC 将其选为样本边.

情况 3: 时间推移到路径点, 滑动窗口只和片段 SL_{new} 相交. 这时 ϵ 为滑动窗口中优先级最高的边, 算法将其选为样本边.

在情况 3 之后, 滑动窗口与新的片段相交, 当前的 SL_{new} 变为 SL_{old}, 情况返回第 1 种, 3 种情况交替出现.

作者证明了在任意时刻, 一个子流中获得有效样本的概率为 $p = |W|/|SL_{\text{new}} + SL_{\text{old}}|$. 其中 $|W|$ 和 $SL_{\text{old}} + SL_{\text{new}}|$ 分别表示滑动窗口以及最近两个片段中的边数. 而 K 个子流中获取的样本边的总数期望值则为 pK.

● 三角形采样概率的计算: 由于 SWTC 算法维护的样本集是滑动窗口的无偏样本集, 作者证明了在任意时刻 T, 滑动窗口中任意 i 条边被加入样本集的概率为 $p(i) = \binom{m}{i} \Big/ \binom{|W|}{i}$. 其中 m 表示 T 时刻样本边的数目. 因此, 滑动

© 中国科学院软件研究所 http://www.jos.org.cn

窗口中的一个三角形被加入样本图的概率即为 $p(3)$, 将样本图中的三角形数目除以该概率, 便可以估算滑动窗口中的三角形数目. 然而, 为了计算 $p(3)$, 需要估算滑动窗口中的边数 $|W|$. 如第 2.1 节所述, 虽然滑动窗口在时间尺度的长度是固定的, 但是其中的边数会随着图流流量而变化. 一方面, 算法无法发现滑动窗口中非样本边的过期, 另一方面, 在二项计数语义下, 需要估算去除重复边的互异边数目, 这都导致了估算的困难. SWTC 首先计算与滑动窗口相交的两个片段内的边数 $|SL_{\text{old}}+SL_{\text{new}}|$, 然后再根据 $|SL_{\text{old}}+SL_{\text{new}}|$ 进一步估算滑动窗口中的边数 $|W|$. 下面分别对这两个步骤进行介绍.

为了估算 $|SL_{\text{old}}+SL_{\text{new}}|$, SWTC 借助了经典的基数估算算法 Hyperloglog sketch[38]. Hyperloglog sketch 需要为每一条边计算一个服从概率为 1/2 的几何分布的权重, 并在每个子流中保存目标区间内的权重最大值. 结合 K 个子流中保存的最大权重, 它便可以估算出目标权重内的边数. 原本的 Hyperloglog sketch 使用哈希函数进行子流划分和权重生成, 以估算互异边的数目, 但如果将哈希函数替换为随机函数, 它也能支持含重复边的计数. SWTC 已经为每条边计算了一个服从均匀分布的优先级 $G(e)$, 可以将该优先级转换为服从概率为 1/2 的几何分布的权重, 计算方式为 $R(e)=\lceil -\log(1-G(e))\rceil$. 不难发现, $G(e)$ 越大, $R(e)$ 也越大, 因此. 在一个子流 S_i 中, 最近两个片段 SL_{old} 和 SL_{new} 内 $G(e)$ 最大的边, 也就是 $R(e)$ 最大的边, 而这条边就是该子流中保存的 β 和 ϵ 中的优先级较高者. 我们可以根据这条边的优先级, 直接计算出子流 S_i 中在 $SL_{\text{old}}+SL_{\text{new}}$ 内的最大 $R(e)$, 记为 R_i. 所有 K 个子流的 R_i 组成了一个 Hyperloglog sketch, 可以使用文献 [38] 中的提供的公式估算 $|SL_{\text{old}}+SL_{\text{new}}|$, 具体公式为 $|SL_{\text{old}}+SL_{\text{new}}|=\dfrac{\alpha_K K^2}{\sum_{i=1}^{K}2^{-R_i}}$, 其中 $\alpha_K=0.7213/(1+1.079/K)$.

有了 $|SL_{\text{old}}+SL_{\text{new}}|$, 就可以进一步估算滑动窗口内的边数 $|W|$. 作者证明一个子流中获得有效样本的概率为 $p=|W|/|SL_{\text{new}}+SL_{\text{old}}|$, 而这一概率又可以通过 m/K 计算, 也就是用产生有效样本的子流数目 m 除以子流总数 K. 因此可以按照 $|W|=m/K\times|SL_{\text{old}}+SL_{\text{new}}|$ 来估算滑动窗口内的边数.

当使用哈希函数实现子流划分函数 $H(\cdot)$ 和优先级计算函数 $G(\cdot)$ 时, 可以估算出滑动窗口内去重后的互异常边的数目, 用于二项计数语义下的估算, 而使用随机函数时, 可以估算出包含重复边在内的边数, 用于带权计数语义下的估算. 这也和上文中采样方法的设置一致.

- 滑动窗口内三角形数目的估算: 有了每个三角形被加入样本图的概率 $p(3)$, 我们可以将滑动窗口内的三角形数目则估算为 $\tau/p(3)$, 其中 τ 为样本图中的三角形数目. SWTC 算法在图流更新的过程中持续更新 τ 的值, 每当有新样本边被加入样本集时, 算法计数该边引入的三角形数目并累加到 τ 上. 而当样本边因过期而被删除, 或者被同一个子流中优先级更高的新边替换时, 算法计数因为该边删除而减少的三角形数目并从 τ 中减去.
- 改进方案——分组异步采样: SWTC 中的样本边数目会随着时间推移发生周期性变化. 当前时间恰好在路径点时, 滑动窗口只和一个片段相交 (即情况 3), 样本边数目达到最大值, 之后则逐步降低, 在即将到达下一个路径点时达到最小值. 为了避免样本集大小的波动带来估算准确率的不稳定, Gou 等人[16]又提出了名为分组异步采样的改进方案. 该方案中, SWTC 将 K 个子流分为若干组, 每组采用不同的路径点进行图流切分, 以此来保证各组的样本边数目此消彼长, 总体的样本集合大小则保持稳定.

下文中, 我们将以 SWTC 的采样算法为基础, 构建 CBS 算法.

2.3 采样前计数策略

采样前计数策略最早在 TRIEST[17]文献中被提出, 在后续工作 ThinkD[18] 中被进一步改进. 该策略的核心思想是充分利用图流中每一条边的拓扑信息, 估算该边的添加或者删除给全图三角形数目带来的增加值或减少值. 假设已有一个样本图 G_s, 对于图流中每条新加入的边 e, 该策略估算 e 在快照图中构成的新三角形集合 Δ 的大小. 为了估算 $|\Delta|$, 该策略查找 e 在 G_s 中构成的三角形集合 Δ_s. 可知 Δ 中的一个三角形被计入 Δ_s 的充要条件为该三角形除了 e 之外的两条边均被采样, 因此 $|\Delta|=|\Delta_s|/p(2)$, 其中 $p(2)$ 表示图流中给定的两条边被采样的概率, 具体值由采样算法决定. 该策略使用一个计数器 τ 计数快照图中的三角形数目估算值, 并在每条新边 e 到达时按照上述方法估算其在快照图中构成的三角形数目 $|\Delta|$, 将其累加到 τ 上, 之后再调用采样算法, 决定 e 是否被加入样本图 G_s. 采

© 中国科学院软件研究所 http://www.jos.org.cn

样前计数策略可以处理全动态模型中的显式删除, 对于每一条删除的边 e, 该策略计算 e 在 G_s 中构成的三角形数目 $|\Delta_s|$, 之后估算 e 在快照图中构成的三角形数目为 $|\Delta_s|/p(2)$, 这些三角形会因为 e 的删除而消失, 所以需要从 τ 中减去该值. 之后检查 e 是否为样本边, 若是, 则将其从 G_s 中删除. 查询时, 该策略直接返回 τ 作为三角形数目估算值. 将采样前计数策略应用在滑动窗口模型时的主要困难在于处理边过期. 该策略需要知晓每一次边的删除, 无论其是否为样本边. 全动态模型中, 删除是由图流中的数据项指明的, 因此算法总能知晓. 而在滑动窗口中, 边删除, 或者说边过期, 是随着时间推移自发发生的, 非样本边没有被保存, 因此无法知晓其何时过期. 为了处理这一问题. 下文将提出处理过期数据的全新策略.

3 CBS 算法

本节将提出高精度的滑动窗口模型下的图流三角形近似计数算法 CBS. CBS 算法结合 SWTC 算法和采样前计数策略, 并提出处理数据过期的策略. 实验表明, 本算法相比 SWTC 算法在相同空间下取得了更高的估算精度. 我们将在第 3.1 节介绍基础版本的 CBS 算法, 并在第 3.2 节进一步提出误差修正策略 ETM, 进一步提高估算的精确度并保证其无偏性.

3.1 基础版 CBS 算法

基础版 CBS 算法结合 SWTC 和采样前计数, 并使用时间切片的策略以处理过期数据. 算法选择一系列固定的路径点, 将时间轴切分为长度为 N/d 的区间 (interval). 这一切片策略与 SWTC 中的方案类似, 称切分后的片段为区间 (interval) 以便与 SWTC 采样算法中的图流片段 (slice) 区分. 此外, 本文称该方法为时间划分而非图流切片, 因为下文中会研究时间戳落入每个区间的三角形, 而非仅关注时间戳在该区间内的图流数据项. 在任意时刻 T, 滑动窗口至多与 $d+1$ 个区间相交, 这些区间的并集可以表示为 $(\lfloor Td/N\rfloor\times N/d-N,T]$, 是滑动窗口 $(T-N,T)$ 的超集.

图 6 展示了 CBS 算法中一个时间划分的示例, 其中每个区间的长度为 3, 而滑动窗口的长度为 6. 当前时刻为 13, 滑动窗口与 3 个区间相交, 也就是区间 3, 4 和 5. 这 3 个区间的并集是 (6, 13] (注意区间 5 是 (12, 15], 但由于当前时间是 13, 所以其中只包含到时刻 13 为止的数据), 而滑动窗口为 (7, 13]. 而等到时刻 15, 滑动窗口只与 2 个区间相交 (区间 4 和 5), 这 2 个区间的并集刚好为滑动窗口.

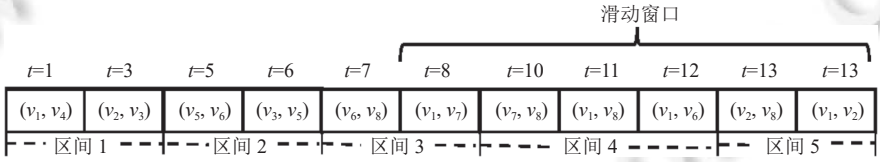


图 6 CBS 区间切分

进一步的, 我们定义图流中的有效三角形如下.

● 有效三角形: 给定一个长度为 N 的滑动窗口以及当前时刻 T, 图流中的有效三角形由图流中的 3 条边 $[(v_i,v_j),(v_i,v_k),(v_j,v_k)]$ 组成, 这 3 条边都在 T 时刻之前出现过, 且 3 条边的时间戳的最大值和最小值之差不超过 N.

在时刻 T, 图流中的有效三角形包括目前在滑动窗口中的, 以及曾经在滑动窗口中出现过的三角形. 此外, 定义有效三角形的时间戳如下.

● 有效三角形的时间戳: 一个有效三角形的时间戳为它的 3 条边的时间戳的最小值.

有如下定理.

定理 1. 在 T 时刻, 一个三角形在快照图中的充要条件是该三角形是一个时间戳在 $(T-N,T]$ 之内的有效三角形.

证明: 若该三角形在快照图中, 则它一定由滑动窗口中的边组成, 因此 3 条边的时间戳都在 $(T-N,T]$ 之内, 所以组成该三角形的 3 条边的时间戳之差不超过 N, 其中的最小值也在 $(T-N,T]$ 之内, 因此该三角形是一个时间戳

© 中国科学院软件研究所 http://www.jos.org.cn

在 $(T-N,T]$ 之内的有效三角形. 反之, 对于一个时间戳在 $(T-N,T]$ 之内的有效三角形, 其 3 条边的时间戳都在区间 $(T-N,T]$ 内, 也就是在滑动窗口中, 因此这 3 条边都在快照图中, 该三角形也一定在快照图中.

为计算快照图中的三角形, 需要计数时间戳在 $(T-N,T]$ 之内的有效三角形的数目. 为实现这一目标, CBS 使用 SWTC 算法维护滑动窗口内的无偏样本集合, 同时使用采样前计数策略估算时间戳落在每一个区间内的有效三角形的数目. 之后将与滑动窗口相交的区间内的有效三角形数目加和, 这些三角形是时间戳在 $(\lfloor Td/N\rfloor\times N/d-N,T]$ 内的有效三角形. 使用它来作为时间戳在 $(T-N,T]$ 内的有效三角形数目的近似值. 由于每个区间的起点和终点是固定的, 计数区间内的有效三角形数目时不需要处理数据过期.

在图 6 中, 时间戳在区间 3, 即 $(6,9]$ 中的有效三角形有 2 个, 分别是 $[(v_6,v_8),(v_1,v_8),(v_1,v_6)]$ 和 $[(v_1,v_7),(v_1,v_8),(v_7,v_8)]$, 时间戳在区间 4, 即 $(9,12]$ 中的有效三角形有 1 个, 也就是 $[(v_1,v_8),(v_1,v_2),(v_2,v_8)]$. 目前还没有时间戳在区间 5 中的三角形, 但是需要注意的是, 时间戳在一个区间内的有效三角形不一定只包含该区间内的边, 例如三角形 $[(v_6,v_8),(v_1,v_8),(v_1,v_6)]$ 中, 边 (v_1,v_8) 和边 (v_1,v_6) 在区间 4 中, 但是由于边 (v_6,v_8) 在区间 3 中且其时间戳最小, 所以该三角形的时间戳依然在区间 3 中.

算法 1 给出了 CBS 算法处理图流中一条新边的流程.

算法 1. CBS 处理图流新边.

输入: 图流中的新边 $e=(v_i,v_j)$, 样本图 G_s, 三角形计数器 $\tau_0-\tau_d$;
输出: 更新后的样本图和三角形计数器.

//计数新边在样本图中构成的三角形
1. $\Delta_s=\varnothing$
2. **for** $v_k\in G_s.FindNeighbor(v_i)\cap G_s.FindNeighbor(v_j)$ **do**
3. $\quad\delta=[(v_i,v_j),(v_i,v_k),(v_j,v_k)]$
4. $\quad\delta.ts=\min\{(v_i,v_j).ts,(v_i,v_k).ts,(v_j,v_k).ts\}$
5. $\quad\Delta_s=\Delta_s\cup\{\delta\}$
//增加每个三角形所在区间内的三角形计数器
6. **for** each $\delta\in\Delta_s$ **do**
7. $\quad l=\left\lfloor\frac{Td}{N}\right\rfloor\times\frac{N}{d}$
8. $\quad$**if** $\delta.ts>l$ **then**
9. $\quad\quad x=0$
10. $\quad$**else**
11. $\quad\quad x=\left\lfloor\frac{(l-\delta.ts)d}{N}\right\rfloor+1$
12. $\quad\tau_x+=\frac{1}{p(2)}$
13. $Gs.TrySample(e)$ //对新边运行采样算法

CBS 维护 $d+1$ 个三角形计数器 $\tau_0-\tau_d$, 分别记录与滑动窗口相交的 $d+1$ 个区间内的有效三角形数量的估算值, 初始值全部为 0. 其中最新的区间编号为 I_0, 其三角形数目估算值保存在计数器 τ_0, 而倒数第 2 个区间则编号为 I_1, 其三角形数目估算值保存在计数器 τ_1, 依次类推. 对于图流中的一条新边 $e=(v_i,v_j)$, CBS 算法首先查找 e 在样本图 G_s 中形成的三角形集合, 如算法 1 的第 1–5 行所示. 算法寻找点 v_i 和 v_j 在 G_s 中的每一个共同邻居 v_k, 也就是满足 $(v_i,v_k)\in G_s$ 且 $(v_j,v_k)\in G_s$ 的点 v_k, 算法将三角形 $[(v_i,v_j),(v_i,v_k),(v_j,v_k)]$ 加入样本图中形成的三角形集合 Δ_s, 并将其时间戳设置为 3 条边时间戳的最小值. 需要注意的是, 算法虽然查找了 e 可以在 G_s 中产生的三角形, 但是并不把 e 加入 G_s, e 是否能成为样本图中的样本边, 需要在进行三角形计数之后再由采样算法来决定. 然后, 算法

© 中国科学院软件研究所 http://www.jos.org.cn

枚举 Δ_s 中的每一个三角形, 根据其时间戳计算它所在的区间. 根据时间戳计算所属区间的方法如算法 1 的第 7–11 行所示. 假设三角形时间戳为 ts, 可以计算出最新的路径点为 $l=\lfloor Td/N\rfloor\times N/d$, 而该三角形所在的区间编号则是:

$$\begin{cases} x=0, & ts>l \\ x=\left\lfloor\dfrac{(l-ts)d}{N}\right\rfloor+1, & ts\leqslant l \end{cases} \tag{1}$$

当 $ts>l$ 时, 说明该时间戳在最新的区间, $x=0$. 否则, 根据 ts 和 l 之间的距离计算, 当 $ts\leqslant l$ 且和 l 之间的距离不满一个区间长度, 也就是 N/d 时, 其在倒数第 2 个区间, $x=1$. 而 ts 和 l 之间的距离在 1–2 倍的区间长度之间时, 其在倒数第 3 个区间, $x=2$. 依次类推. 因此得到公式 (1).

此时, 该三角形所在的区间便是 I_x, 将 τ_x 增加 $1/p(2)$, 其中 $p(2)$ 表示滑动窗口中任意两条边被加入样本图的概率, 对于 SWTC 采样算法, 该值为 $p(2)=\binom{m}{2}\Big/\binom{|W|}{2}=\dfrac{m(m-1)}{|W|(|W|-1)}$, 其中 $|W|$ 和 m 分别为滑动窗口中的边数以及样本图中的边数, 前者可以通过 SWTC 算法估算. e 在快照图中产生的三角形被加入 Δ_s 的概率, 便是该三角形除了 e 之外的两条边均被采样的概率, 也就是 $p(2)$. 因此, 对于 Δ_s 中的每一个三角形, 需要将其所在区间的三角形计数器增加 $1/p(2)$, 以此估算 e 的加入在该区间内产生的有效三角形的数量. 这也是采样前计数的基本思想. 我们会在第 3.3 节对此估算方法的无偏性进行详细证明. 最后, CBS 算法再调用 SWTC 算法的采样流程, 来尝试对 e 进行采样. 此采样过程中, e 可能会替换掉样本图中的一条原有样本边, 但不需要根据被替换出的样本边对三角形计数器进行任何修改. 因为 CBS 的三角形计数器是通过估算每条新边到达时在快照图中增加的三角形数目来更新的, 而不是像 SWTC 一样根据样本图中三角形数目更新, 样本边的换出不会对其造成影响.

算法 2 则给出了当时刻从 T 增加 ΔT 时处理过期数据的流程.

算法 2. CBS 处理过期数据.

输入: 当前时刻 T, 时间增量 ΔT, 样本图 G_s, 三角形计数器 $\tau_0-\tau_d$;
输出: 删除过期数据后的样本图和三角形计数器.

1. $y=\left\lfloor\dfrac{(T+\Delta T)d}{N}\right\rfloor-\left\lfloor\dfrac{Td}{N}\right\rfloor$
2. **if** $y>0$ **then**
//经历了 y 个新区间, 后移旧区间的计数器
3. **for** $i=d-y;\ i\geqslant 0;\ i--$ **do**
4. $\tau_{i+y}=\tau_i$
//初始化前 y 个计数器, 用于计数新区间的三角形
5. **for** $i=0;\ i\leqslant\min(d,y-1);\ i++$ **do**
6. $\tau_i=0$
7. 删除 G_s 中时间戳不大于 $T+\Delta T-N$ 的边
8. $T=T+\Delta T$

CBS 算法首先检查 $\lfloor(T+\Delta T)d/N\rfloor-\lfloor Td/N\rfloor$ 的值, 如果其为 y 且大于 0, 则说明时间又经过了 y 个区间, 抛弃最早的 y 个区间 $\tau_{d-y+1}-\tau_d$ 的三角形计数器, 并将其余的三角形计数器向后移动 y 位, 空出来的计数器 $\tau_0-\tau_{y-1}$ 则用于在后续计数新增的 y 个区间的有效三角形数目, 将其初始化为 0. 之后算法查找样本图中所有的过期边, 也就是时间戳不在 $(T+\Delta T-N,T+\Delta T]$ 内的边, 将其删除. 最后算法更新当前时刻 $T=T+\Delta T$. 实际应用中, 该流程一般由系统时钟触发, 每当时钟向前推进一个时间单元时, 更新 T 并触发一次数据过期. 而在第 4 节的实验中, 由于本文使用的是历史图流数据集而非实时数据, 选择由数据驱动更新而非由系统时钟驱动. 具体来说, 按照时间戳顺序处理图流数据, 并将当前时刻 T 设置为处理为已经处理的最大时间戳. 每当有新边读入时, 首先更新当前时刻 T,

© 中国科学院软件研究所 http://www.jos.org.cn

并且调用一次过期处理函数, 之后再调用算法 1 处理新边.

在查询时, 算法返回所有 $d+1$ 个三角形计数器的和 $\sum_{i=0}^{d}\tau_i$ 作为滑动窗口中三角形数目的估算值.

CBS 算法事实上是以区间替换的方式实现了三角形计数值的批量过期. 每当到达一个路径点, 就会有一个区间不再与滑动窗口相交, 将其中的有效三角形数量从计数器中删除, 该操作代表了对于时间戳在该区间内的有效三角形的批量过期操作. 例如在图 6 中, 等到时刻 15, 滑动窗口不再与区间 3, 也就是 $(6,9]$ 相交, 该区间内的有效三角形数目便会从计数器中减去. 由于估算的其实是滑动窗口的超集 $(\lfloor Td/N\rfloor\times N/d-N,T]$ 内的三角形数目, 所以该估算的期望值大于滑动窗口内的三角形数目. 但是通过将区间切分得足够小, 也就是设置足够大的 d, 我们可以将误差控制在较小的范围内. 第 3.2 节还会提出进一步的改进策略, 实现无偏估算, 也就是让估算期望值恰好等于滑动窗口内的三角形数目.

3.2 误差修正算法 ETM

基础版 CBS 算法使用时间戳在最近 $d+1$ 个区间内的有效三角形数量 τ 来作为滑动窗口中三角形数量的估算值, 而前者在期望上大于后者. 在本节中, 我们进一步提出误差修正算法 ETM (expired triangle-count modification) 来实现无偏估算, 也就是让估算值的期望等于滑动窗口中三角形的数目.

如第 3.1 节所述, 最近 $d+1$ 个区间的并集是 $(\lfloor Td/N\rfloor\times N/d-N,T]$, 相比滑动窗口 $(T-N,T]$, 多出了时间段 $(\lfloor Td/N\rfloor\times N/d-N,T-N]$. 例如在图 6 中, 最近 $d+1$ 个区间是区间 3–5, 它们的并集是 $(6,13]$, 相比滑动窗口 $(7,13]$, 多出的时间段为 $(6,7]$. 只要能估算出时间戳在该时间段内的有效三角形数目, 并将其从 τ 中减去, 便能让 τ 的期望值等于滑动窗口中的三角形数目. 从另一个角度来说, 该时间段也就是与滑动窗口相交的最早的区间 I_d 中不属于滑动窗口的部分, 而该时间段内的有效三角形数目, 也就是 I_d 已经过期的有效三角形数目, 使用一个误差修正计数器 $\tau_{\exp}$ 来记录该值.

然而, 如第 2.3 节所述, 不能使用采样前计数策略来估算过期三角形的数目, 因为它要求算法知晓每一次边过期, 无论其是否被采样. 而由于 CBS 算法没有保存非样本边, 这一点无法实现. 因此, 本文选择仅使用样本边的信息来估算 $\tau_{\exp}$ 的值. 每当滑动窗口移动时, 算法记录被采样进 G_s 的过期三角形集合 Δ_s. 每一个过期三角形被加入 Δ_s 的概率等于它的 3 条边都被采样的概率, 也就是 $p(3)=\binom{m}{3}\Big/\binom{|W|}{3}=\dfrac{m(m-1)(m-2)}{|W|(|W|-1)(|W|-2)}$, 其中 $|W|$ 和 m 分别为滑动窗口中的边数以及样本图中的边数. 因此, 算法将 $\tau_{\exp}$ 增加 $|\Delta_s|/p(3)$, 以此来实现对于 I_d 内过期三角形数目的无偏估算. 在查询时, 把 τ_0 到 τ_d 的 d+1 个计数器加和后, 从中减去 $\tau_{\exp}$ 的值, 从而实现对于滑动窗口中三角形数目的无偏估算.

值得注意的是, 当时间经过路径点时, 需要把 $\tau_{\exp}$ 设置为 0. 因为 $\tau_{\exp}$ 记录的是与滑动窗口相交的最旧的区间内的过期三角形数目. 经过路径点时, 原来最旧的区间 I_d 不再与滑动窗口相交, 算法也会抛弃它的有效三角形计数值, 而原来的 I_{d-1} 会变成现在的最旧的区间 I_d, 需要重置 $\tau_{\exp}$, 并且重新开始计数新的 I_d 中的过期三角形数目.

算法 3 展示了使用 ETM 改进后的 CBS 处理过期数据的算法.

算法 3. 改进版 CBS 处理过期数据.

输入: 当前时刻 T, 时间增量 ΔT, 样本图 G_s, 三角形计数器 $\tau_0-\tau_d$, 误差修正计数器 $\tau_{\exp}$;
输出: 删除过期数据后的样本图和三角形计数器.

1. $y=\left\lfloor\dfrac{(T+\Delta T)d}{N}\right\rfloor-\left\lfloor\dfrac{Td}{N}\right\rfloor$
2. **if** $y>0$ **then**
//经历了 y 个新区间, 后移旧区间的计数器并初始化前 y 个计数器
3. $\quad\tau_{\exp}=0$
4. $\quad$**for** $i=d-y;\ i\geqslant 0;\ i--$ **do**
5. $\quad\quad\tau_{i+y}=\tau_i$

© 中国科学院软件研究所 http://www.jos.org.cn

6. **for** $i=0;\ i\leqslant \min(d,y-1);\ i++$ **do**
7. $\tau_i=0$
8. $E_{\exp}\leftarrow G_s$ 中时间戳不大于 $T+\Delta T-N$ 的边
9. $l=\left(\left\lfloor\frac{(T+\Delta T)d}{N}\right\rfloor-d\right)\times\frac{N}{d}$
10. **while** $E_{\exp}\neq\varnothing$ **do**
11. $e\leftarrow E_{\exp}$ 中时间戳最小的边
12. **if** $e.ts>l$ **then**

//计数该过期边组成的三角形

13. $v_i,v_j\leftarrow e$ 的两个端点
14. $\varDelta_s=\varnothing$
15. **for** $v_k\in G_s.FindNeighbor(v_i)\cap G_s.FindNeighbor(v_j)$ **do**
16. $\delta=[(v_i,v_j),(v_i,v_k),(v_j,v_k)]$
17. $\Delta_s=\Delta_s\cup\{\delta\}$
18. $\tau_{\exp}+=\frac{|\varDelta_s|}{p(3)}$
19. 从 G_s 和 $E_{\exp}$ 中删除 e
20. $T=T+\Delta T$

与原方法相比, 该方法额外增加了对于 $\tau_{\exp}$ 计数器的更新. 当前时刻增加 ΔT 时, 首先计算是否会经过路径点并产生新的区间. 如果 $y=\lfloor(T+\Delta T)d/N\rfloor-\lfloor Td/N\rfloor$ 大于 0, 说明经过了 y 个新区间, 首先将 $\tau_{\exp}$ 设置为 0, 因为其中统计的是旧的 I_d 内的过期三角形数目, 现在旧的 I_d 已经不再与滑动窗口相交, $\tau_{\exp}$ 的旧值也不再需要. 之后算法使用算法 2 相同的流程更新三角形计数器 $\tau_0-\tau_d$. 之后算法收集 G_s 中的过期边集合 $E_{\exp}$, 并计算更新后 I_d 的左端点 l. 算法按时间戳从小到大的顺序扫描 $E_{\exp}$ 中的每条边 e, 当 $e>l$ 时, 计数其在 G_s 中形成的三角形数量, 并在除以概率 $p(3)$ 后累加到 $\tau_{\exp}$ 上. 注意可能出现 $e\leqslant l$ 的情况, 这时说明 e 不在 I_d 内, 而是在已经不与滑动窗口相交的更早区间内, 它形成的三角形也不在 I_d 内, 不需要对其进行计数. 之后算法从 G_s 中删除 e. 需要注意的是, 算法按照时间顺序处理过期边, 是为了避免重复计数 G_s 中包含多条过期边的三角形. 最后, 设置 $T=T+\Delta T$.

改进后的 CBS 算法中, 处理图流新边的方法和算法 1 一致. 而查询时, 需要返回 $\sum_{i=0}^{d}\tau_i-\tau_{\exp}$ 的值.

需要注意的是, 估算 I_d 内过期三角形数目 $\tau_{\exp}$ 时只使用了样本边的信息, 相比采样前计数策略不够精准. 但是由于只用该方法估算不超过一个区间的小范围内的三角形数目, 其估算误差对整个滑动窗口内三角形数目估算的影响较小. 第 4 节的实验表明, 改进后的 CBS 算法能取得比 SWTC 算法低 70% 的平均误差.

3.3 数学分析

本节我们对 CBS 算法进行数学分析, 包括估算无偏性证明、方差分析和算法复杂度分析.

3.3.1 无偏性分析

定理 2. 结合了 ETM 误差修正之后的 CBS 算法能够提供滑动窗口内三角形数量的无偏估算.

证明: 假设改进后的 CBS 算法的三角形数目估算值为 τ, 则有:

$$E(\tau)=E(\tau_{\text{int}}-\tau_{\exp})=E(\tau_{\text{int}})-E(\tau_{\exp}) \tag{2}$$

其中, $\tau_{\text{int}}=\sum_{i=0}^{d}\tau_i$, 也就是 $d+1$ 个与滑动窗口相交区间内的有效三角形数目估算值之和. 而 $\tau_{\exp}$ 则是 CBS 对区间 I_d 内的过期有效三角形的数目估算值. $E(\cdot)$ 表示变量的期望值. 我们依次对它们的期望进行分析. 首先是 τ_{int}:

$$E(\tau_{\text{int}})=E\left(\sum_{i=0}^{d}\tau_i\right)=\sum_{i=0}^{d}E(\tau_i) \tag{3}$$

本文用 $\varDelta_i$ 表示时间戳在区间 I_i 内的有效三角形集合. 对于每一个三角形 $\delta\in\varDelta_i$, 设置一个变量 $\xi(\delta)$. 假设 δ 中

© 中国科学院软件研究所 http://www.jos.org.cn

时间戳最大的边为 e, 其时间戳为 $mt(\delta)$. 若 e 到达时, 其他两条边均在样本图中, 则 $\xi(\delta)$ 取 $1/p(2)^{mt(\delta)}$, 否则取 0. 注意由于滑动窗口中边数和样本图的大小随着时间在不断变化, 所以滑动窗口中任意两条边被加入样本图的概率 $p(2)$ 也在不断变化, 而 $p(2)^{mt(\delta)}$ 则表示 $mt(\delta)$ 时刻 $p(2)$ 的值. $\xi(\delta)$ 变量代表了 CBS 算法中三角形 δ 给计数器 τ_i 带来的增量. $\xi(\delta)$ 非 0 的充要条件是三角形 δ 的最后一条边 e 到达时, 其他两条边已在样本图中, 这时 δ 会被 CBS 算法的三角形搜索流程搜索到 (算法 1 的第 1–5 行), 而 $\xi(\delta)$ 的取值 $1/p(2)^{mt(\delta)}$ 则是在 δ 被搜索到之后, CBS 算法对于 δ 的时间戳所在区间内的计数器的增量 (算法 1 第 7–11 行, 注意算法 1 中省略了 $p(2)$ 的时间上标, 事实上使用的正是边 e 到达时刻的 $p(2)$ 值, 也就是 $p(2)^{mt(\delta)}$). 综上所述, $\xi(\delta)$ 表明了 CBS 算法运行时是否会搜索到三角形 δ, 以及搜索到的情况下对于计数器 τ_i 的增加值. 因此可以得出 $\tau_i=\sum_{\delta\in\Delta_i}\xi(\delta)$. 由于 CBS 算法保证了样本图是滑动窗口内边的无偏样本, 因此可知 $\xi(\delta)$ 取 $1/p(2)^{mt(\delta)}$ 的概率, 也就是 δ 的前两条边被采样的概率, 正是 $p(2)^{mt(\delta)}$. 所以有:

$$E(\tau_i)=E\left(\sum_{\delta\in\Delta_i}\xi(\delta)\right)=\sum_{\delta\in\Delta_i}E(\xi(\delta))=\sum_{\delta\in\Delta_i}\frac{1}{p(2)^{mt(\delta)}}\times p(2)^{mt(\delta)}=|\Delta_i| \tag{4}$$

因此:

$$E(\tau_{\text{int}})=\sum_{i=0}^{d}E(\tau_i)=\sum_{i=0}^{d}|\Delta_i|=|\Delta_{\text{ind}}| \tag{5}$$

其中, Δ_{ind} 表示与滑动窗口相交的区间内的有效三角形集合.

然后分析 τ_{exp}. 类似上文方法, 假设 Δ_{exp} 表示区间 I_d 内的过期有效三角形的集合. 对于每一个三角形 $\delta\in\Delta_{\text{exp}}$, 设置一个变量 $\xi(\delta)$. 假设 δ 中时间戳最小的边为 e, 结合有效三角形时间戳的定义, 可知 e 的时间戳就是 $\delta.ts$. 若 e 恰好过期时 (也就是 $\delta.ts+N$ 时刻), δ 中的所有 3 条边均在样本图中, 则 $\xi(\delta)$ 取 $1/p(3)^{\delta.ts+N}$, 否则取 0, 其中 $p(3)^{\delta.ts+N}$ 表示 $\delta.ts+N$ 时刻滑动窗口中任意 3 条边被加入样本图的概率. 由于样本图是滑动窗口内边的无偏样本, 可知 $\xi(\delta)$ 取 $1/p(3)^{\delta.ts+N}$ 的概率即为 $p(3)^{\delta.ts+N}$. 而结合 ETM 算法的流程, 可知 $\xi(\delta)$ 代表的是 $\tau_{\text{exp}}=\sum_{\delta\in\Delta_{\text{exp}}}\xi(\delta)$. 所以:

$$E(\tau_{\text{exp}})=E\left(\sum_{\delta\in\Delta_{\text{exp}}}\xi(\delta)\right)=\sum_{\delta\in\Delta_{\text{exp}}}E(\xi(\delta))=\sum_{\delta\in\Delta_{\text{exp}}}\frac{1}{p(3)^{\delta.ts+N}}\times p(3)^{\delta.ts+N}=\left|\Delta_{\text{exp}}\right| \tag{6}$$

因此, 有:

$$E(\tau)=E(\tau_{\text{int}})-E(\tau_{\text{exp}})=|\Delta_{\text{ind}}|-\left|\Delta_{\text{exp}}\right| \tag{7}$$

而 $|\Delta_{\text{ind}}|-\left|\Delta_{\text{exp}}\right|$ 即为滑动窗口中的三角形数量 $|\Delta|$. 因此 $E(\tau)=|\Delta|$, CBS 算法是无偏估算.

3.3.2 方差分析

接下来分析 CBS 算法的方差并和 SWTC 算法对比. 滑动窗口中的边数是随着图流流量而不断变化的, CBS 算法的方差不仅受到当前滑动窗口中的边数和快照图的拓扑结构的影响, 还会受到图流流量历史变化的积累影响. 这导致在真实场景下难以给出 CBS 方差的分析结果. 因此, 在这里考虑一种简化的场景: 图流的流量是稳定的, 不会随着时间而改变, 这种情况下, 滑动窗口中的边数 $|W|$, CBS 和 SWTC 算法采样出的样本边的数目 m, 以及滑动窗口中任意 i 条边被加入样本集的概率 $p(i)=\binom{m}{i}\Big/\binom{|W|}{i}$ 都是稳定值.

(1) SWTC 方差分析

首先分析 SWTC 算法的方差. 类比上文的无偏性分析部分, 给滑动窗口中的每个三角形 δ 赋予一个变量 $\chi(\delta)$, 该变量表示的是 SWTC 算法中, 三角形 δ 为 SWTC 中三角形总数估算值带来的增量. 若查询时刻, δ 的 3 条边均在样本集中时, $\chi(\delta)=1/p(3)$. 这时 δ 会被统计为样本图中的三角形, 而 SWTC 算法最终以样本图中的三角形数目除以 $p(3)$ 作为滑动窗口中三角形总数的估算值, 所以 δ 给三角形总数估算值带来了 $1/p(3)$ 的增加量. 而反之, 若 δ 任意一条边不在样本集中, 则 $\chi(\delta)=0$, 因此可知 SWTC 对于滑动窗口内三角形数目的估算值为:

$$\tau_{\text{swtc}}=\sum_{\delta\in\Delta}\chi(\delta) \tag{8}$$

其中, Δ 为滑动窗口中三角形的集合. 因此其方差可以如下分析:

$$Var(\tau_{\text{swtc}})=Var\left(\sum_{\delta\in\Delta}\chi(\delta)\right)=\sum_{\delta,\delta^*\in\Delta}Cov(\chi(\delta),\chi(\delta^*))=\sum_{\delta\in\Delta}Var(\chi(\delta))+\sum_{\delta,\delta^*\in\Delta,\delta\neq\delta^*}Cov(\chi(\delta),\chi(\delta^*)) \tag{9}$$

© 中国科学院软件研究所 http://www.jos.org.cn

更进一步地, 有 $Var(\chi(\delta)) = E(\chi(\delta)^2) - E(\chi(\delta))^2$. 由于 δ 的 3 条边均在样本集中的概率为 $p(3)$, 因此 $E(\chi(\delta)) = p(3)\times\frac{1}{p(3)} = 1$, 而 $E(\chi(\delta)^2) = p(3)\times\frac{1}{p(3)^2} = \frac{1}{p(3)}$, 因此 $Var(\chi(\delta)) = \frac{1}{p(3)} - 1$.

另一方面, $Cov(\chi(\delta),\chi(\delta^*)) = E(\chi(\delta)\chi(\delta^*)) - E(\chi(\delta))E(\chi(\delta^*))$, 其中 $E(\chi(\delta))E(\chi(\delta^*)) = 1$. 而对于 $E(\chi(\delta)\chi(\delta^*))$, 若 δ 与 δ^* 没有公共边, 则 $\chi(\delta)$ 与 $\chi(\delta^*)$ 都非 0 的充要条件是两个三角形的 6 条边都被采样, 概率为 $p(6)$. 若 δ 与 δ^* 有一条公共边, 则 $\chi(\delta)$ 与 $\chi(\delta^*)$ 都非 0 的充要条件是两个三角形的共 5 条边都被采样, 概率为 $p(5)$.

$$E(\chi(\delta)\chi(\delta^*)) = \begin{cases} \frac{p(6)}{p(3)^2}, & \delta\text{与}\delta^*\text{没有公共边} \\ \frac{p(5)}{p(3)^2}, & \delta\text{与}\delta^*\text{有一条公共边} \end{cases} \tag{10}$$

因此, 公式 (9) 可以进一步展开为:

$$Var(\tau_{\text{swtc}}) = |\varDelta|\left(\frac{1}{p(3)} - 1\right) + 2\eta\left(\frac{p(6)}{p(3)^2} - 1\right) + 2\lambda\left(\frac{p(5)}{p(3)^2} - 1\right) \tag{11}$$

其中, $|\varDelta|$ 为滑动窗口中的三角形数目, η 为滑动窗口中没有公共边的互异三角形对的数目 (无序对), λ 为滑动窗口中有一条公共边的互异三角形对的数目 (无序对).

(2) CBS 方差分析

对于 CBS 算法我们有:

$$Var(\tau_{\text{cbs}}) = Var(\tau_{\text{ind}} - \tau_{\text{exp}}) = Var(\tau_{\text{ind}}) + Var(\tau_{\text{exp}}) - 2Cov(\tau_{\text{ind}}, \tau_{\text{exp}}) \tag{12}$$

其中, τ_{exp} 和 τ_{ind} 正相关, 因为一个三角形被计入 τ_{exp} 的充要条件是它的 3 条边均被采样, 那么它的前两条边也会别采样, 因此它也会被计入 τ_{ind}. 所以 $Cov(\tau_{\text{ind}}, \tau_{\text{exp}})$ 为正数. 所以:

$$Var(\tau_{\text{cbs}}) \leqslant Var(\tau_{\text{ind}}) + Var(\tau_{\text{exp}}) \tag{13}$$

其中, τ_{exp} 的估算过程和 SWTC 算法相同, 一个三角形被纳入 τ_{exp} 的充要条件是它的 3 条边均被采样, 而这时它会给 τ_{exp} 带来 $1/p(3)$ 的增加量. 因此可以类比 SWTC 的分析来计算 $Var(\tau_{\text{exp}})$, 并得到相似的结果, 只是需要将公式 (11) 中的滑动窗中口三角形数目以及滑动窗口中互异三角形对的数目改为最旧的区间 I_d 中的过期三角形数目以及过期三角形对的数目. 图流流量稳定时, I_d 的长度为滑动窗的中的 $1/d$, 其中过期三角形数目小于滑动窗口中三角形数目的 $1/d$, 过期三角形对的数目也小于滑动窗口中三角形对数目的 $1/d$. 因此 $Var(\tau_{\text{ind}}) < \frac{1}{d}Var(\tau_{\text{swtc}})$. 为了让对比更加直观, 我们考虑 $d \to K$, 也就是说区间划分的数量 d 足够大, 接近子流数目 K, 这时 $Var(\tau_{\text{ind}})$ 只有 $Var(\tau_{\text{swtc}})$ 的数万分之一, 相比之下可以忽略不计, 因此有:

$$Var(\tau_{\text{cbs}}) \leqslant Var(\tau_{\text{ind}}) + Var(\tau_{\text{exp}}) \approx Var(\tau_{\text{ind}}) \tag{14}$$

我们接下来分析 $Var(\tau_{\text{ind}})$, 类比 SWTC, 可以得到:

$$Var(\tau_{\text{ind}}) = Var\left(\sum\nolimits_{\delta\in\bar{\varDelta}}\xi(\delta)\right) = \sum\nolimits_{\delta,\delta^*\in\bar{\varDelta}} Cov(\xi(\delta),\xi(\delta^*)) = \sum\nolimits_{\delta\in\bar{\varDelta}} Var(\xi(\delta)) + \sum\nolimits_{\delta,\delta^*\in\bar{\varDelta},\delta\neq\delta^*} Cov(\xi(\delta),\xi(\delta^*)) \tag{15}$$

其中, $\xi(\delta)$ 的定义如无偏性证明部分所述. $\bar{\varDelta}$ 表示的是与滑动窗口相交的 $d+1$ 个区间内的有效三角形的集合, 当图流流量稳定时, 该集合中三角形数目为滑动窗口中三角形数目的 1 到 $1+\frac{1}{d}$ 倍.

进一步的, $Var(\xi(\delta)) = E\left(\xi(\delta)^2\right) - E(\xi(\delta))^2$. 如第 3.3.1 节分析, $\xi(\delta)$ 有 $p(2)$ 的概率取 $1/p(2)$, 其他情况取 0, 因此 $E(\xi(\delta)) = p(2)\times\frac{1}{p(2)} = 1$, 而 $E\left(\xi(\delta)^2\right) = p(2)\times\frac{1}{p(2)^2} = \frac{1}{p(2)}$, $Var(\xi(\delta)) = \frac{1}{p(2)} - 1$.

而 $Cov(\xi(\delta),\xi(\delta^*)) = E(\xi(\delta)\xi(\delta^*)) - E(\xi(\delta))E(\xi(\delta^*))$, 其中 $E(\xi(\delta))E(\xi(\delta^*)) = 1$. 下面我们分析 $E(\xi(\delta)\xi(\delta^*))$. 假设 δ 中时间戳较小的两条边组成的集合为 s, δ^* 中时间戳较小的两条边组成的集合为 s^*, $\xi(\delta)$ 与 $\xi(\delta^*)$ 都非 0 的充要条件, δ 中的最后一条边到达时, s 中的两条边均被采样, 而 δ^* 中的最后一条边到达时, s^* 中的两条边也都均被采样. 若 δ 和 δ^* 没有公共边, 则 s 和 s^* 也没有交集, 要达成上述条件, 需要 $s\cup s^*$ 的 4 条边都被采样, 概率为 $p(4)$(注意这

© 中国科学院软件研究所 http://www.jos.org.cn

里我们做了一定近似, 忽略了样本边被替换的情况). 若 δ 与 δ^* 有一条公共边 e, 则 $\xi(\delta)$ 与 $\xi(\delta^*)$ 都非 0 的概率需要根据组成 δ 与 δ^* 的 5 条边的时间顺序而定. 当 $e \in s$ 且 $e \in s^*$ 时, s 和 s^* 有交集, $s \cup s^*$ 中只有 3 条边, 与 $\xi(\delta^*)$ 都非 0 的充要条件是这 3 条边被采样, 概率为 $p(3)$, 否则, 概率依然为 $p(4)$. 假设边的时间戳与拓扑无关时, $e \in s$ 且 $e \in s^*$ 的概率为 8/15, 因此有:

$$E(\chi(\delta)\chi(\delta^*)) = \begin{cases} \dfrac{p(4)}{p(2)^2}, & \delta\text{与}\delta^*\text{没有公共边} \\ \dfrac{8\times p(3)+7\times p(4)}{15\times p(2)^2}, & \delta\text{与}\delta^*\text{有一条公共边} \end{cases} \tag{16}$$

因此公式 (15) 可以进一步展开为:

$$Var(\tau_{\text{ind}}) = |\bar{\Delta}|\left(\frac{1}{p(2)}-1\right)+2\bar{\eta}\left(\frac{p(4)}{p(2)^2}-1\right)+2\bar{\lambda}\left(\frac{8\times p(3)+7\times p(4)}{15\times p(2)^2}-1\right) \tag{17}$$

其中, $|\bar{\Delta}|$ 与 $\bar{\eta}$, $\bar{\lambda}$ 分别表示与滑动窗口相交的 $d+1$ 个区间内的有效三角形数目, 没有公共边的互异三角形对的数目和有一条公共边的互异三角形对的数目, 当 $d \to K$ 时, 我们有 $|\bar{\Delta}| \approx |\Delta|$, $\bar{\eta} \approx \eta$, $\bar{\lambda} \approx \lambda$.

$$Var(\tau_{\text{cbs}}) \leqslant Var(\tau_{\text{ind}})+Var(\tau_{\text{exp}}) \approx Var(\tau_{\text{ind}}) = |\Delta|\left(\frac{1}{p(2)}-1\right)+2\eta\left(\frac{p(4)}{p(2)^2}-1\right)+2\lambda\left(\frac{8\times p(3)+7\times p(4)}{15\times p(2)^2}-1\right) \tag{18}$$

为了直观对比 CBS 算法和 SWTC 算法的方差, 在 $|W|$ 和 m 足够大的情况下, 将 $p(i)$ 近似为 r^i, 其中 $r = \dfrac{m}{|W|}$, 此时有:

$$Var(\tau_{\text{swtc}}) = |\Delta|\left(\frac{1}{r^3}-1\right)+2\lambda\left(\frac{1}{r}-1\right) \tag{19}$$

$$Var(\tau_{\text{cbs}}) \approx |\Delta|\left(\frac{1}{r^2}-1\right)+2\lambda\left(\frac{1}{r}-1\right)\times\frac{8}{15} \tag{20}$$

对比可以发现 $Var(\tau_{\text{cbs}}) < Var(\tau_{\text{swtc}})$. 在第 4 节中的实验也证实了以上分析.

3.3.3 空间复杂度分析

定理 3. 假设图流划分的子流数目为 K, CBS 算法的空间复杂度为 $O(K)$, 而时间复杂度为 $O(|E|K)$, 其中 $|E|$ 为图流中到达的总边数, 平均在图流中每一条边更新上的均摊复杂度为 $O(K)$.

证明: 与 SWTC 算法相比, CBS 算法新增的数据结构只有 $d+1$ 个三角形计数器, 其空间占用为 $O(d)$, 而 SWTC 算法产生的样本图中的大小为 $O(K)$, 实验设置中, d 是一个小于 K 的常数, 所以合计空间复杂度依然为 $O(K)$. 在时间复杂度上, 每次在新边到达时, 我们需要计数它在样本图中产生的三角形数目, 这一过程需要计算该边两个端点的邻居交集, 复杂度与样本图大小成正比, 也就是 $O(K)$. 而仅当该边被采样时, 其过期时会再触发一次三角形计数, 再次引入 $O(K)$ 的代价. 因此, 图流中每条新边产生的时间代价为 $O(K)$, 而图流中边到达的边总数为 $|E|$ 时, 总计时间复杂度为 $O(|E|K)$.

3.4 有向三角形计数

在上文中, 将三角形计数中的边都视为无向边, 这也是大部分现有工作的问题定义方式. 而考虑边方向后, 有向三角形计数事实上属于一种更广泛的被称为范式计数 (motif counting) 的研究问题. 有向三角形可以被 7 种范式, 如图 7 所示. 前 2 种范式只包含单向边, 后 5 种则包含单向边和双向边. 注意这里定义的双向边是被标记为双向的单一边, 并非图流中两条反向边的组合, 否则其组成的范式将为四边范式而非三边的三角形范式. 此定义与之前的工作相同[14]. 本文的 CBS 算法和 SWTC 算法均可以扩展到有向三角形计数中, 只需要在算法计数三角形时计数不同的有向三角形即可, 其他步骤和无向三角形计数相同.

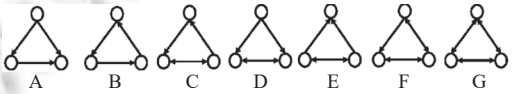


图 7 不同的有向三角形范式

© 中国科学院软件研究所 http://www.jos.org.cn

3.5 CBS-FP 算法

CBS 算法使用的采样方法为 SWTC 算法中的优先级采样方案, 也可以将其替换为固定概率采样方案, 也就是对于图流中的每一条新边, 运行一个满足概率为 r 的伯努利分布的随机函数, 当函数返回值为 1 时, 将新边加入样本集合, 否则便丢弃该边. 当使用固定概率采样时, 在估算三角形数目时, 需要把计算公式中的 $p(2)$ 和 $p(3)$, 也就是任意 2 条, 或 3 条边被加入样本集的概率, 替换为 r^2 和 r^3, 其他步骤和原版的 CBS 算法相, 称该变体为 CBS-FP 算法. 与 CBS 算法相比, CBS-FP 算法的优点在于采样算法简单, 因此运行速度很高. 而缺点有两点. 首先, CBS-FP 算法的样本集大小为 $r|W|$, 其中 $|W|$ 为滑动窗口内的边数, 该值随着图流的流量, 也就是单位时间内到达的边数而变化. 因此, 其内存占用随着图流流量而变化, 难以预估其内存占用峰值并在系统中预留足够内存. 其次, 即使能够预估其内存占用峰值并预留出来, 这部分内存在平时无法被充分利用, 导致算法在占用相同的预留内存的情况下, 平均误差低于 CBS 算法. 我们会在第 4 节对 CBS 和 CBS-FP 算法进行更详细的对比.

4 实验分析

在本节, 对 CBS 算法进行实验分析. 在第 4.1 和第 4.2 节, 会分别介绍实验使用的数据集和设置. 由于 SWTC 算法是目前已知的唯一一个滑动窗口上具有固定空间消耗的三角形数目估算算法, 因此主要将 CBS 算法与 SWTC 算法进行对比. 在第 4.3 节, 会对比 CBS 算法和 SWTC 算法的估算误差. 在第 4.4 节中, 分析 CBS 算法的估算效果随着滑动窗口大小的变化趋势, 并与 SWTC 算法对比. 而在第 4.5 节中, 进一步将 CBS 算法和固定概率采样算法对比, 后者被多种经典三角形近似计数算法如 MASCOT[28] 采用, 但是这些经典算法没有考虑滑动窗口模型, 将其进行一定调整以适应滑动窗口模型, 具体细节如第 2.1 节所述. 此外, 也比较了 CBS 算法使用固定概率采样时的变体 CBS-FP, 以及基于全动态模型的 WRS[19] 和 ThinkD[18]算法. 需要注意的是, 为了将全动态模型的算法迁移到滑动窗口模型, 需要保存滑动窗口中的所有边及其时间戳, 以判断每一条边何时过期. 在第 4.6 节中, 对 CBS 算法进行消融实验, 分析误差修正算法 ETM 的影响, 以及区间划分数量 d 的影响. 在第 4.7 节中, 对 CBS 和 SWTC 在有向三角形计数中的效果进行对比分析. 最后, 第 4.8 节将对不同算法的运行速度和内存消耗进行分析.

所有的实验均在一台具有 18 核 CPU (Intel Xeon CPU E5-2697 v4@2.3 GHz, 每核 2 线程) 和 192 GB 内存的服务器上进行, 服务器运行 CentOS 系统, 代码采用 C++ 编写并使用 GCC 4.8.5 编译, 运行时采用 O3 优化. 所有代码均已在 https://github.com/StreamingTriangle/CBS 上开源.

4.1 实验数据

实验采用如下 3 个源自真实世界的数据集,

StackOverflow[39]: 该数据集是技术交流社区 StackOverflow 上的用户交互数据. 数据集中的点代表用户, 边代表用户间的交互, 每条边具有时间戳, 表示交互发生的时间. 该数据集共有 63 497 050 条边, 2 601 977 个点. 该数据集存在重复边, 代表两个用户之间发生了多次交互. 数据集下载自图数据开源网站 Konect[40].

Wikipedia[41]: 该数据集描述了法语区维基百科的网页超链接关系, 数据集中的点代表网页, 而边代表网页之间的超链接关系. 该数据集共有 123 709 902 条边和 3 333 397 个点. 数据集中不含重复边. 数据集下载自图数据开源网站 Konect[40].

Yahoo[42]: 来自 Yahoo 公司的开源网络流数据集, 我们把 IP 地址作为点, IP 之间的通信作为边, 数据集包含 561 754 369 条边和 33 635 059 个点, 数据集中含重复边.

Wikipedia 数据集没有边上的时间戳, 因此我们随机为边生成了均匀分布的时间戳. 而 StackOverflow 和 Yahoo 数据集则采用了原始时间戳. 我们使用不同算法按照时间顺序处理数据集中的边, 以此来模拟真实图流场景. 此外, 对于含重复边的前两个数据集, 我们采用第 1 节所述的带权计数语义.

4.2 实验设置

- 滑动窗口设置: 为了让滑动窗口长度更加直观, 我们计算数据集中相邻两条边的平均时间间隔, 也就是用数据集中最大时间戳和最小时间戳之间的差值, 除以数据集内的总边数. 我们以这样的平均时间间隔作为滑动窗口

© 中国科学院软件研究所 http://www.jos.org.cn

长度的单位. 如此一来, 滑动窗口的长度也代表了滑动窗口中的平均边数.

● 测量点设置: 我们把当前时刻 T 设置为已经处理的最大时间戳, 算法按时间顺序处理数据集中的边, 此过程中 T 不断增大, 新边不断被加入滑动窗口, 而旧的边则不断过期, 导致滑动窗口向前持续滑动, 快照图及其中的三角形数目不断变化, 算法的估算误差也在不断波动. 我们每间隔 $N/50$, 也就是每当 T 增加滑动窗口长度 N 的 1/50 时, 设置一个测量点. 我们在这些测量点测量不同算法对当前时刻快照图中三角形数目的估算值 τ^* 和真实值 τ 之间的相对误差 $|(\tau^*-\tau)/\tau|$. 之后, 我们计算平均相对误差和最大相对误差作为算法估算误差的评价指标, 分别定义如下.

平均相对误差: 所有测量点的相对误差的平均值.

最大相对误差: 所有测量点的相对误差的最大值.

需要注意的是, 我们不会采用数据集中的前 50 个测量点, 这时滑动窗口中的边数过少, 存在大量三角形数目为 0 的测量点, 导致无法计算相对误差. 在表 1 中, 列出了实验使用的 3 个数据集在默认滑动窗口设置下, 所有测量点的平均三角形数量. 此外, 每组实验都使用不同的随机数种子跑 10 轮, 取 10 轮的平均值作为最终实验结果. 需要注意的是, 由于 Yahoo 数据集总规模和滑动窗口长度都很大, 在其上的计数, 尤其是计数精确三角形数目, 十分耗时, 为了保证实验能及时完成, 本文只使用了前 300 个测量点.

表 1 不同数据集的平均三角形数量

数据集及滑动窗口长度	StackOverflow (4M)	Wikipedia (10M)	Yahoo (20M)
平均三角形数量	61 502 435	42 883 353	53 787 454

● 算法参数设置: CBS 算法与 SWTC 算法都采用了基于子流划分的固定空间采样. 其中的子流数目 K 代表了算法存储的边数以及样本图的最大大小, 也代表了算法的空间占用大小. 而 CBS 相比 SWTC, 在数据结构上的唯一区别是增加了 d 个三角形计数器, 而这 d 个三角形计数器只占数十 Byte, 相比算法总空间消耗的数十 MB 可以忽略, 因此, 我们可以认为两个算法的空间消耗在相同的 K 的情况下是相同的. 为了直观反映样本大小和滑动窗口总边数的关系, 我们进一步定义了样本比例.

● 样本比例: CBS 和 SWTC 算法中的子流数目 K 和滑动窗口长度 N 的比值, 其中滑动窗口长度以上文所述的平均时间间隔为单位.

样本比例反映了样本集的最大大小与滑动窗口中的平均边数之间的比值. 下文中, 通过设置采样比例来控制算法的空间占用. 在第 4.8 节中,本文也会列出样本比例和实际内存占用的对应关系.

此外, 对于 CBS 算法的区间划分数量 d, 设置的默认值为 10, 也就是说每个区间的大小为滑动窗口长度的 1/10.

4.3 误差对比

图 8 和图 9 分别展示了 SWTC 算法和 CBS 算法的平均相对误差和最大相对误差随着采样比率的变化趋势.

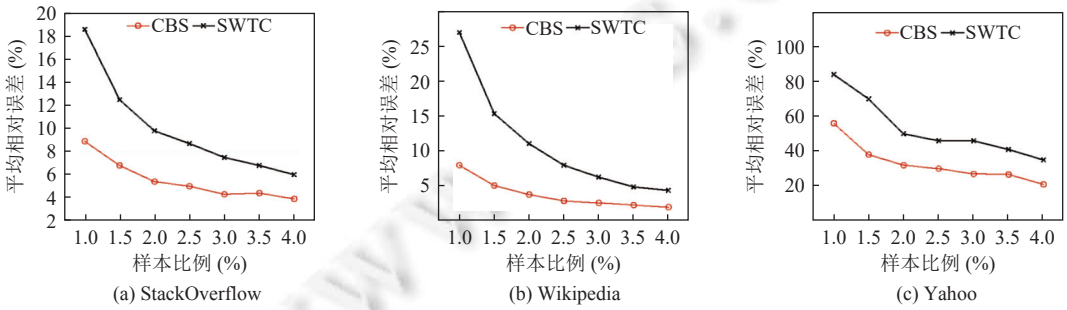


图 8 不同算法的平均相对误差

图 8 和图 9 中 StackOverflow、Wikipedia、Yahoo 这个数据集的滑动窗口长度依次是 4M、10M、20M. 从实验结果中我们可以看出, 两种算法的误差都会随着采样比例的增加而显著下降, 这是由于快照图大小不变的情况

© 中国科学院软件研究所 http://www.jos.org.cn

下, 更大的样本集合带来了更高的估算精度. 得益于采样前估算策略, CBS 算法的误差一直优于 SWTC 算法, 其平均相对误差降低了最高 70%, 而最大相对误差降低了最高 54%. 此外, 在样本比例较低时, CBS 算法的优势更加明显. 需要注意的是, Yahoo 数据集中的估算误差比其他两个数据集高, 是由于该数据集相比其他两个数据集更加稀疏 (从表 1 可以看出), 这导致在其上的估算难度更高.

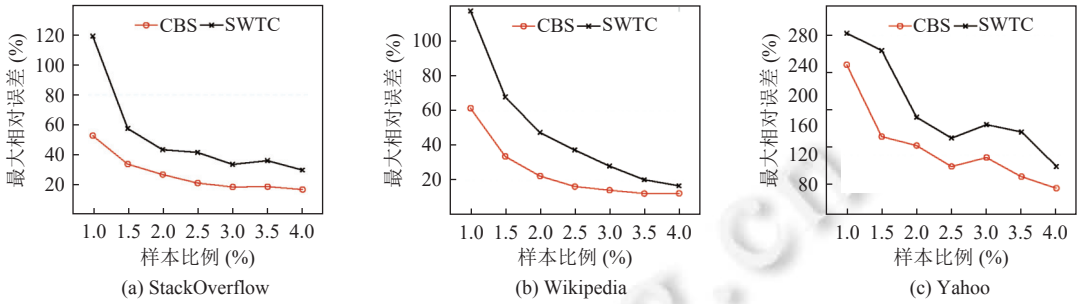


图 9 不同算法的最大相对误差

4.4 可扩展性对比

图 10 展示了 SWTC 算法和 CBS 算法的平均相对误差和最大相对误差随着滑动窗口长度的变化趋势. 实验采用的是 StackOverflow 数据集. 固定采样比例为 4%, 也就是说样本大小会随着滑动窗口长度线性增加. 可以看出, 随着滑动窗口长度的增加算法的误差都在降低. 这是由于更大的滑动窗口中, 快照图中三角形数目和样本图中的三角形数目都更多, 采样估算受随机误差影响更小. 此外, 得益于采样前估算策略, 在滑动窗口较小时, CBS 算法的表现会远好于 SWTC 算法.

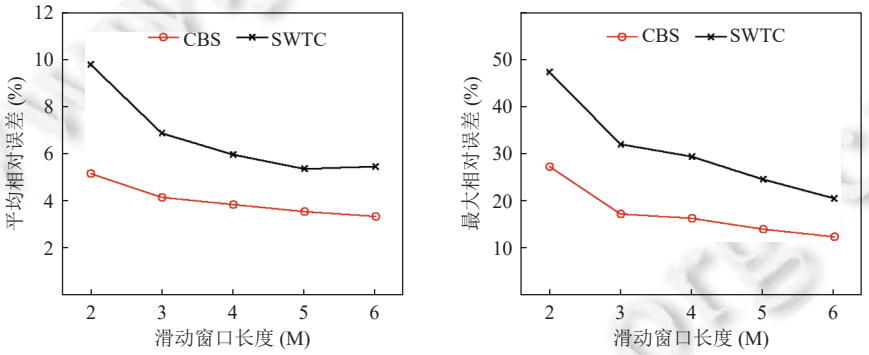


图 10 不同算法误差随滑动窗口变化趋势

4.5 与其他算法的误差对比

本节对比 CBS 算法、SWTC 算法以及固定概率采样算法 (简称为 FP) 和基于固定概率采样的 CBS 算法变体 CBS-FP, 以及基于全动态模型的 ThinkD 和 WRS 算法的估算误差. 本实验使用 StackOverflow 数据集, 设置滑动窗口长度为 4M. 提前扫描了一遍数据集, 计算出滑动窗口中的最大边数 (4.88M), 据此计算出 FP 算法和 CBS-FP 算法需要设置的采样概率, 以保证这两种算法的内存消耗上限和前两种算法保持一致. 为了将 WRS 和 ThinkD 算法从全动态模型迁移到滑动窗口模型, 需要按时间顺序保存滑动窗口中的所有边和时间戳, 以便在每条边过期时告知算法. 这会带来 153 MB 的存储开销. 因此, WRS 和 ThinkD 只有在可用内存大于 153 MB 时才能工作. 相比全动态模型算法, FP、CBS-FP 与 CBS、SWTC 算法之间的精度差距较小, 为了避免在一幅图中呈现时, 后 4 种算法的误差曲线难以区分, 本文用两组图来分别呈现 CBS、SWTC 和固定概率采样算法 FP、CBS-FP 的对比, 以及 CBS、SWTC 与全动态模型算法 WRS、ThinkD 的对比.

© 中国科学院软件研究所 http://www.jos.org.cn

图 11 展示了 CBS、SWTC 和 FP、CBS-FP 算法的对比. 我们调节 CBS 算法和 SWTC 算法的样本比例展开实验, 并同时调节 FP 和 CBS-FP 算法的采样概率, 使得 4 种算法的内存消耗保持一致. 从图 11 中可以看出, 采用固定空间消耗的两种算法的平均误差低于各自的固定概率采样变体, 最大误差则没有明显区别. 这是由于基于固定概率采样的方案需要按照流量峰值预留内存, 而非峰值时刻, 样本集合大小会随着流量降低而降低, 无法充分利用预留内存, 导致非峰值时刻的准确度不如固定空间消耗的方案. 但是最大误差主要受峰值时刻影响, 这时两类方案的样本大小相近, 因此最大误差区别较小. 此外, CBS-FP 和 CBS 算法的平均相对误差均低于 SWTC 算法. 但是需要注意的是, 实际应用中是无法预估峰值流量的, 所以固定概率采样方案的应用场景会更加受限.

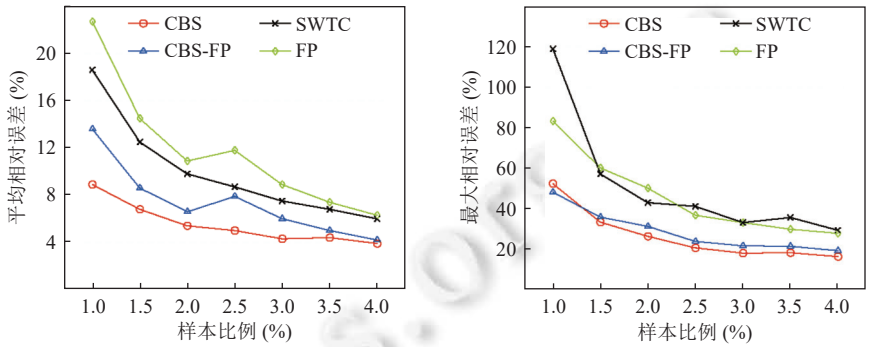


图 11 固定概率采样算法和固定空间消耗算法的误差对比

图 12 则展示了 CBS、SWTC 算法和全动态模型算法的 ThinkD 和 WRS 的对比, 通过调节算法的内存占用来进行实验. 可以看出, ThinkD、WRS 在内存达到 153 MB 以上时才开始工作, 这些算法需要 240 MB 以上的内存才能将平均误差控制在 2% 左右, CBS 和 SWTC 算法则只需要 80 MB 内存就能达到该效果. 这说明了将全动态模型算法直接迁移到滑动窗口模型是十分低效的. 而相比 SWTC 算法, CBS 算法具有更低的误差.

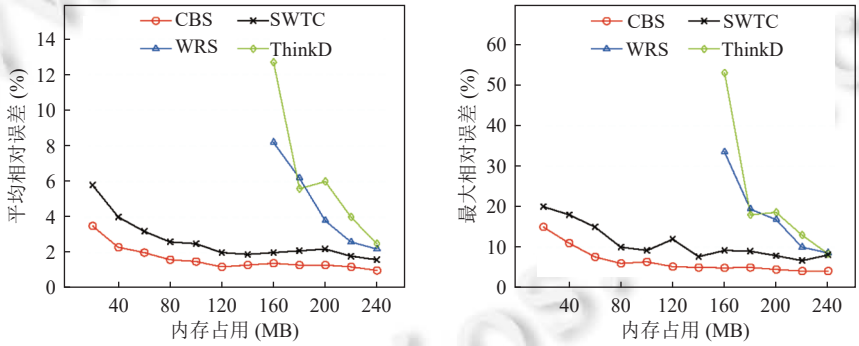


图 12 CBS、SWTC 算法与全动态模型算法的误差对比

4.6 消融实验

本节分析 CBS 算法的误差随着区间划分数量 d 的变化, 并分析 ETM 误差修正算法的影响. 我们测试了没有误差修正的 CBS 算法, 称为 CBS-bias 的估算误差, 和完整版的 CBS 算法对比, 如图 13 所示. 该图采用了指数坐标轴, 因为 CBS-bias 算法的误差变化范围极大. 实验采用 StackOverflow 数据集, 滑动窗口长度设置为 4M, 采样比例设置为 4%. 图 13 加入 SWTC 算法的误差作为对比. 由于 SWTC 算法中没有参数 d, 所以其误差曲线为一条水平线. 从图 13 中可以看出, CBS 和 CBS-bias 算法的误差都会随着 d 的增加而降低, 而 CBS-bias 算法收到的影响尤为明显. 由于缺乏误差修正, 当 d 较小时, CBS-bias 算法报告的事实上是一个远大于滑动窗口的区间中的三角形数

© 中国科学院软件研究所 http://www.jos.org.cn

目, 而随着 d 增大, 其估算的区间越来越接近滑动窗口, 误差快速降低. 在 $d \geqslant 20$ 时, CBS-bias 算法的误差低于 SWTC 算法. 而相比之下, 加入 ETM 误差修正策略后, CBS 算法的误差大大降低, 且在 d 较小时也能取得低误差. d 较小时, CBS 需要估算一个较大区间内的过期三角形数目, 而此估算只使用了样本边信息, 估算精度不如采样前计数策略, 因此误差高于 d 较大的情况. 但是 $d > 5$ 之后, 误差降低的趋势就已非常微弱.

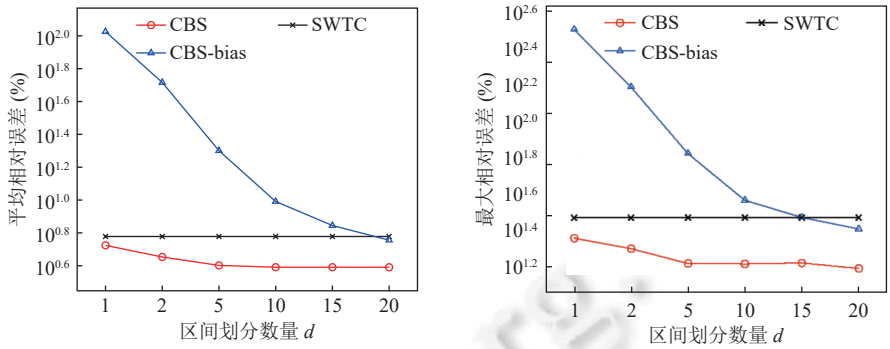


图 13 CBS 算法误差随区间划分数量变化趋势

4.7 有向三角形计数实验

本节评估 CBS 算法在有向三角形计数中的效果, 并和 SWTC 算法对比, 实验结果如图 14 所示. 在最大相对误差的实验图中采用指数坐标, 因为不同三角形的最大相对误差差距较大. 实验采用 StackOverflow 数据集, 设置滑动窗口长度为 4M, 样本比例为 4%. 由于 StackOverflow 数据集中不存在双向边, 但是边分为 3 种类别, 分别表示用户间的 3 种交互方式: 回答问题、评论问题和评论答案. 本文选择了 3 种边类别中的“回答问题”类别标记为双向边, 以此来支持 C–G 的 5 类三角形范式计数. 实验中, G 类三角形估算的误差较高, 这是由于 G 类三角形的数量很低. 我们也在样本比例 8% 的设置下进行了 G 类三角形计数, 可以看出误差得到了极大降低. 此外, CBS 算法的误差低于 SWTC 算法, 尤其是在误差较高的范式, 如 G 类三角形中.

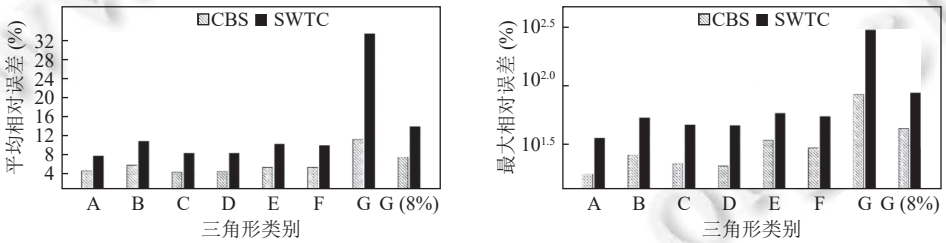


图 14 有向三角形计数估算误差

4.8 运行速度和内存消耗对比

本节对不同算法的实际内存消耗和运行速度进行实验分析. 实验采用 StackOverflow 数据集, 设置滑动窗口长度为 4M. 图 15 展示了 CBS 与 SWTC 算法的更新速度随着样本比例的变化, CBS-FP 和 FP 算法则按照第 4.6 节相同的方式调节采样概率, 使得其空间消耗上限和 CBS 以及 SWTC 算法保持一致. 更新速度的单位为每秒百万次新边插入 (million insertion per second, Mips). 从中可以看出, FP 算法的更新速度最高, 这是由于其采样和估算算法最为简单. 而 SWTC 算法的速度低于 FP 算法, 这是由于 SWTC 算法的采样方案更加复杂, 采样中不仅会加入新样本边, 还会替换掉原有样本边, 带来更大的数据结构更新代价和三角形数目更新代价. 而 CBS 算法和 CBS-FP 算法则更加低一些, 这时由于前两种算法只有新边被采样时才会触发三角形计数, 而使用了采样前计数策略的 CBS 和 CBS-FP 算法则对于所有新边都需要进行三角形计数, 计算代价更高. 但是之前的实验表明了 CBS 和 CBS-FP 算法在估算精度上具有很大优势, 速度上的牺牲时是取得高估算精度的必要代价. 在实际应用场景中可以根据

© 中国科学院软件研究所 http://www.jos.org.cn

寻求选择. 此外, 作为对比, 本文也测试了精确三角形计数算法的更新速度, 其结果为 0.0076 Mips, 比 CBS 算法慢 100 倍以上. 而且, 由于三角形计数的时间复杂度受到快照图大小的影响, 该速度会随着滑动窗口增大而快速下降. 当滑动窗口长度增加到 6M 时, 精确三角形计数算法的更新速度会进一步下降到 0.005 Mips, 而此时近似估算算法的速度依然在 0.4 Mips 以上. 由此可以看出估算算法在速度上的优势.

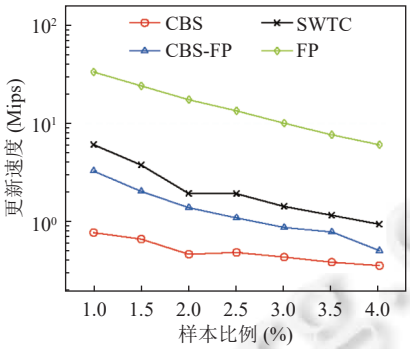


图 15 不同算法的更新速度对比

此外, 表 2 列出了 CBS 和 SWTC 算法的内存消耗值随着样本比例的变化, 滑动窗口大小依然固定为 4M. 可以看出其内存随着样本比例, 也就是算法中的子流数目 K 线性增长. 而相比之下, 精确三角形计数算法的最大内存消耗为 468 MB. 因此实验也验证了估算算法在内存消耗上的巨大优势.

表 2 CBS 与 SWTC 算法内存消耗随样本比例变化趋势

样本比例 (%)	1	2	3	4
内存消耗 (MB)	4.5	9	13.4	17.9

5 总 结

本文提出一种基于采样的滑动窗口模型下的图流三角形近似计数算法 CBS. 该算法将目前最好的滑动窗口模型下的三角形近似计数算法 SWTC 和采样前计数策略结合, 以取得更高的估算精度. 由于采样前计数策略无法处理滑动窗口模型下的边过期, 本文提出基于时间区间划分的原创过期处理方案, 并进一步提出误差修正方案 ETM. 最终实现的 CBS 算法在相同空间消耗下, 相比 SWTC 算法误差降低了 70% 以上. 随后我们使用真实数据集进行了大量实验, 实验结果验证了 CBS 算法的高估算精度.

References:

[1] Grewal A, Jiang J, Lam G, Jung T, Vuddemarri L, Li QN, Landge A, Lin J. Recservice: Distributed real-time graph processing at Twitter. In: Proc. of the 10th USENIX Conf. on Hot Topics in Cloud Computing. Boston: USENIX Association, 2018.

[2] Guha S, McGregor A. Graph synopses, sketches, and streams: A survey. Proc. of the VLDB Endowment, 2012, 5(12): 2030–2031. [doi: 10.14778/2367502.2367570]

[3] Datar M, Gionis A, Indyk P, Motwani R. Maintaining stream statistics over sliding windows. SIAM Journal on Computing, 2002, 31(6): 1794–1813. [doi: 10.1137/S0097539701398363]

[4] Qiu XF, Cen WB, Qian ZP, Peng Y, Zhang Y, Lin XM, Zhou JR. Real-time constrained cycle detection in large dynamic graphs. Proc. of the VLDB Endowment, 2018, 11(12): 1876–1888. [doi: 10.14778/3229863.3229874]

[5] Berry JW, Hendrickson B, LaViolette RA, Phillips CA. Tolerating the community detection resolution limit with edge weighting. Physical Review E, 2011, 83(5): 056119. [doi: 10.1103/PhysRevE.83.056119]

[6] Eckmann JP, Moses E. Curvature of co-links uncovers hidden thematic layers in the World Wide Web. Proc. of the National Academy of Sciences of the United States of America, 2002, 99(9): 5825–5829. [doi: 10.1073/pnas.032093399]

[7] Becchetti L, Boldi P, Castillo C, Gionis A. Efficient algorithms for large-scale local triangle counting. ACM Trans. on Knowledge

© 中国科学院软件研究所 http://www.jos.org.cn

Discovery from Data, 2010, 4(3): 13. [doi: 10.1145/1839490.1839494]
[8] Milo R, Shen-Orr S, Itzkovitz S, Kashtan N, Chklovskii D, Alon U. Network motifs: Simple building blocks of complex networks. Science, 2002, 298(5594): 824–827. [doi: 10.1126/science.298.5594.824]
[9] Kang U, Meeder B, Papalexakis EE, Faloutsos C. HEigen: Spectral analysis for billion-scale graphs. IEEE Trans. on Knowledge and Data Engineering, 2014, 26(2): 350–362. [doi: 10.1109/TKDE.2012.244]
[10] Yang Z, Wilson C, Wang X, Gao TT, Zhao BY, Dai YF. Uncovering social network Sybils in the wild. ACM Trans. on Knowledge Discovery from Data (TKDD), 2014, 8(1): 2. [doi: 10.1145/2556609]
[11] Li ZJ, Lu YT, Zhang WP, Li RH, Guo J, Huang X, Mao R. Discovering hierarchical subgraphs of k-core-truss. Data Science and Engineering, 2018, 3(2): 136–149. [doi: 10.1007/s41019-018-0068-2]
[12] Ding DM, Savi M, Pederzolli F, Siracusa D. Design and development of network monitoring strategies in P4-enabled programmable switches. In: Proc. of the 2022 IEEE/IFIP Network Operations and Management Symp. Budapest: IEEE, 2022. 1–6. [doi: 10.1109/NOMS54207.2022.9789848]
[13] Ahmed NK, Duffield N, Neville J, Kompella R. Graph sample and hold: A framework for big-graph analytics. In: Proc. of the 20th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM, 2014. 1446–1455. [doi: 10.1145/2623330.2623757]
[14] Pavan A, Tangwongsan K, Tirthapura S, Wu KL. Counting and sampling triangles from a graph stream. Proc. of the VLDB Endowment, 2013, 6(14): 1870–1881. [doi: 10.14778/2556549.2556569]
[15] Wang PH, Qi YY, Sun Y, Zhang XL, Tao J, Guan XH. Approximately counting triangles in large graph streams including edge duplicates with a fixed memory usage. Proc. of the VLDB Endowment, 2017, 11(2): 162–175. [doi: 10.14778/3149193.3149197]
[16] Gou XY, Zou L. Sliding window-based approximate triangle counting with bounded memory usage. The VLDB Journal, 2023, 32(5): 1087–1110. [doi: 10.1007/s00778-023-00783-3]
[17] De Stefani L, Epasto A, Riondato M, Upfal E. Trièst: Counting local and global triangles in fully dynamic streams with fixed memory size. ACM Trans. on Knowledge Discovery from Data (TKDD), 2017, 11(4): 43. [doi: 10.1145/3059194]
[18] Shin K, Oh S, Kim J, Hooi B, Faloutsos C. Fast, accurate and provable triangle counting in fully dynamic graph streams. ACM Trans. on Knowledge Discovery from Data (TKDD), 2020, 14(2): 12. [doi: 10.1145/3375392]
[19] Lee D, Shin K, Faloutsos C. Temporal locality-aware sampling for accurate triangle counting in real graph streams. The VLDB Journal, 2020, 29(6): 1501–1525. [doi: 10.1007/s00778-020-00624-7]
[20] Alon N, Yuster R, Zwick U. Finding and counting given length cycles. Algorithmica, 1997, 17(3): 209–223. [doi: 10.1007/BF02523189]
[21] Arifuzzaman S, Khan M, Marathe M. Patric: A parallel algorithm for counting triangles in massive networks. In: Proc. of the 22nd ACM Int'l Conf. on Information & Knowledge Management. San Franciso: ACM, 2013. 529–538. [doi: 10.1145/2505515.2505545]
[22] Wang X, Yang XC. Study of triangle counting algorithm with sliding windows based on FLINK. Computer Science, 2020, 47(10): 83–90 (in Chinese with English abstract). [doi: 10.11896/jsjkx.190900014]
[23] Hu XC, Tao YF, Chung CW. Massive graph triangulation. In: Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2013. 325–336. [doi: 10.1145/2463676.2463704]
[24] Park HM, Myaeng SH, Kang U. PTE: Enumerating trillion triangles on distributed systems. In: Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. San Francisco: ACM, 2016. 1115–1124. [doi: 10.1145/2939672.2939757]
[25] Bar-Yossef Z, Kumar R, Sivakumar D. Reductions in streaming algorithms, with an application to counting triangles in graphs. In: Proc. of the 13th Annual ACM-SIAM Symp. on Discrete Algorithms. San Francisco: Society for Industrial and Applied Mathematics, 2002. 623–632.
[26] Buriol LS, Frahling G, Leonardi S, Marchetti-Spaccamela A, Sohler C. Counting triangles in data streams. In: Proc. of the 25th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems. 2006. 253–262. [doi: 10.1145/1142351.1142388]
[27] Jowhari H, Ghodsi M. New streaming algorithms for counting triangles in graphs. In: Proc. of the 11th Annual Int'l Conf. on Computing and Combinatorics. Kunming: Springer, 2005. 710–716. [doi: 10.1007/11533719_72]
[28] Lim Y, Kang U. Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams. In: Proc. of the 21st ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Sydney: ACM, 2015. 685–694. [doi: 10.1145/2783258.2783285]
[29] Jha M, Seshadhri C, Pinar A. A space efficient streaming algorithm for triangle counting using the birthday paradox. In: Proc. of the 19th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Chicago: ACM, 2013. 589–597. [doi: 10.1145/2487575.2487678]
[30] Tsourakakis CE, Kang U, Miller GL, Faloutsos C. Doulion: Counting triangles in massive graphs with a coin. In: Proc. of the 15th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Paris: ACM, 2009. 837–846. [doi: 10.1145/1557019.1557111]
[31] Gemulla R, Lehner W, Haas PJ. Maintaining bounded-size sample synopses of evolving datasets. The VLDB Journal, 2008, 17(2): 173–201. [doi: 10.1007/s00778-007-0065-y]

© 中国科学院软件研究所 http://www.jos.org.cn

[32] Jung M, Lim Y, Lee S, Kang U. FURL: Fixed-memory and uncertainty reducing local triangle counting for multigraph streams. Data Mining and Knowledge Discovery, 2019, 33(5): 1225–1253. [doi: 10.1007/s10618-019-00630-6]

[33] Zhang LL, Zhang ZW, Wang GR, Yuan Y, Zhao KF. Efficiently counting triangles for hypergraph streams by reservoir-based sampling. IEEE Trans. on Knowledge and Data Engineering, 2023, 35(11): 11328–11341. [doi: 10.1109/TKDE.2023.3236335]

[34] Yang X, Song C, Yu MD, Gu JQ, Liu M. Distributed triangle approximately counting algorithms in simple graph stream. ACM Trans. on Knowledge Discovery from Data (TKDD), 2022, 16(4): 79. [doi: 10.1145/3494562]

[35] Yang X, Song C, Gu JQ, Li K, Li HW. A distributed streaming framework for edge-cloud triangle counting in graph streams. Knowledge-based Systems, 2023, 278: 110878. [doi: 10.1016/j.knosys.2023.110878]

[36] Wang KX, Long C, Yan D, Zhang J, Jagadish HV. Reinforcement learning enhanced weighted sampling for accurate subgraph counting on fully dynamic graph streams. In: Proc. of the 39th IEEE Int'l Conf. on Data Engineering (ICDE). Anaheim: IEEE, 2023. 1084–1097. [doi: 10.1109/ICDE55515.2023.00088]

[37] Babcock B, Datar M, Motwani R. Sampling from a moving window over streaming data. In: Proc. of the 13th Annual ACM-SIAM Symp. on Discrete Algorithms. San Francisco: ACM, 2002. 633–634.

[38] Flajolet P, Fusy É, Gandouet O, Meunier F. HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm. In: Proc. of the 2007 Conf. om Analysis of Algorithms. HAL, 2007. 137–156. [doi: 10.46298/dmtcs.3545]

[39] Stack overflow data source. 2024. http://konect.cc/networks/sx-stackoverflow/

[40] Kunegis J. KONECT: The Koblenz network collection. In: Proc. of the 22nd Int'l Conf. on World Wide Web. Rio de Janeiro: ACM, 2013. 1343–1350. [doi: 10.1145/2487788.2488173]

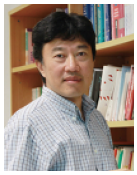
[41] Wikipedia Link data source. 2024. https://dumps.wikimedia.org/

[42] Yahoo data source. 2024. https://webscope.sandbox.yahoo.com/catalog.php?datatype=g

附中文参考文献:

[22] 王旭, 杨晓春. 基于 FLINK 的滑动窗口内三角形计数算法研究. 计算机科学, 2020, 47(10): 83–90. [doi: 10.11896/jsjkx.190900014]

苟向阳(1995－), 男, 博士, CCF 专业会员, 主要研究领域为图流数据, 图查询算法.



于旭(1958－), 男, 博士, 教授, 博士生导师, 主要研究领域为图数据库, 图挖掘, 社交网络, 大数据, 云计算, 数据库中的机器学习算法.

邹磊(1981－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为图数据库, 知识图谱, 面向图的软硬件协调计算, 图数据在大数据方面的应用.

© 中国科学院软件研究所 http://www.jos.org.cn