Sliding Window-based Approximate Triangle Counting over Streaming Graphs with Duplicate Edges[1]

Xiangyang Gou

摘要

实际应用中的图具有天然的动态性，因此流图分析在各个领域中越来越重要。三角形计数则是计算图的聚类系数的关键，然而在具有边重复和边过期的真实流图中近似三角形计数仍然是一个尚未解决的问题。本文提出 SWTC 算法来解决这个问题。SWTC 提出了一种固定长度的切片策略，在内存有限的情况下同时解决无偏采样和基数估计问题。从理论上证明了在给定内存上界的情况下，SWTC 在样本图规模和三角形估算精度上的优越性。

关键词：streaming graphs；approximate triangle counting

1 引言

1.1 选题背景

三角形计数是图的基本问题之一，大量的应用如社区发现，异常探测，垃圾邮件检测都是以三角形计数为基础的。大图上的精确三角形计数需要消耗大量的时间，因此一般使用近似算法来进行计算。随着互联网的不断发展，数据量不断增加，图数据也有了新的组织形式。在社交网络和通信网络中，图数据往往被组织成图流的形式，即 Streaming Graphs。图流可以理解为一系列不断到达的边，这些边可能会出现重复。比如在社交网络中，两个用户可能会有多次的交流，于是在表示用户交流信息的图流中就会出现重复边。之前的图流三角形计数算法都假设图流中没有重复边，但在实际应用中，重复边的出现会对算法的效率和准确性造成一定的影响。本文的目的是提出能够用于有重复边的图流三角形近似计数算法。另外，处理图流时往往使用滑动窗口模型，即只分析最近一段时间内到达的边组成的动态图，故该算法也基于滑动窗口。同时，算法要保证内存消耗的上限，要能够支持边过期和两种三角形计数方法（binary counting 和 weighted counting），算法要能够在以前算法的基础上提高采样率和降低三角形计数误差。综上，本文提出了 SWTC 算法。

1.2 选题依据

该论文于 2021 年发表在数据库领域顶级期刊 SIGMOD，解决了图流计算中的重复边三角计数问题，本文有如下技术贡献：

1. 在基于滑动窗口的近似三角计数问题中，针对重复边问题提出了 SWTC 算法，SWTC 算法引入两个 Hash 函数（或随机函数）分别将边映射到不同的 substream 和 priority，将重复边映射到同一位置和状态，达到处理重复边的目的。另外，根据使用的映射函数，算法可以用于两种三角计数方法（binary counting 和 weighted counting）。

2. 在 SWTC 算法中引入 slice strategy，将图流划分为与窗口等长的 slice，且由于 slice 位置固定，故能够使用 HyperLogLog 实现基数计算，在滑动窗口和 slice 同时对图流进行切分时，滑动窗口内图

流的采样率提升，从而降低了三角形计数的误差。

1.3 选题意义

本文针对滑动窗口重复边三角形计数问题提出的解决方案，以及对滑动窗口内边的基数计算，通过采样图对原图三角形数量的估算方法，都是比较有效的方法，在综合理解和改进之后可以尝试推广到其他问题上，同时复现本论文能够让我理解基本的三角计数计算方式，对流图的结构更加清晰熟悉，对图的结构属性更加了解，为将来在 streaming graph 领域上的研究起到了很好的推动作用。

2 相关工作

2.1 BPS[2]

一种基于滑动窗口采样方法，使用优先级采样（选择滑动窗口中优先级最大的边），但由于存在边过期，故并不是每个滑动窗口中都能成功获取采样边。BPS 通过将过期的边设置为 test item 来作为后续到达的边的优先级的阈值，大于 test item 直接被选为采样边，否则需要等 test item 二次过期后才能选择当前优先级最大的边作为采样边，这样保证了无偏采样，限制了空间消耗上限，确定了采样率下限。

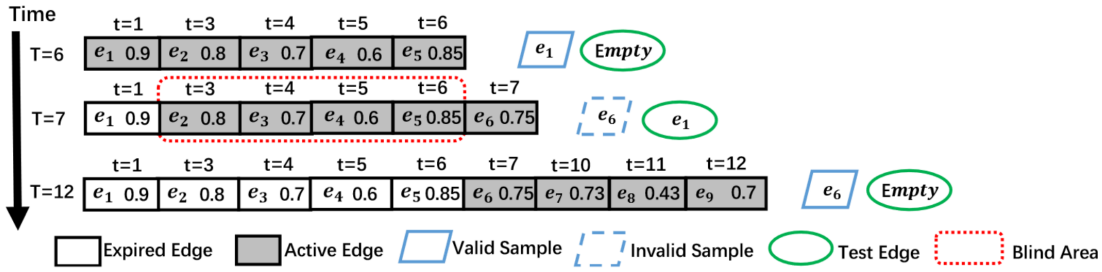


图 1: BPS 算法示意图

如上图所示，BPS 的采样率在 $\left[\frac{|W_{T-N}^{T}|}{|W_{T-2N}^{T}|}, 1-\frac{|W_{T-2N}^{T-N}|}{|W_{T-3N}^{T}|}\right]$ 之间，即 50% 到 66%

故 SWTC 基于此采用了新的采样策略对其进行优化，提高了平均采样率。同时该论文证明在有限内存情况下，若样本大小固定为 k，其空间复杂度为 O(log(n)k)，n 为滑动窗口中的边数，n 难以估计，故无法在滑动窗口中保持固定大小的样本。

2.2 WRS[3]

WRS 对图流进行采样并计算三角形，主要使用蓄水池方法（一种一次性扫描流数据并无偏采样的方法）进行采样，使用删除补偿来解决图流的边过期问题，同时利用三角形形成的时间局部性，使用缓冲区存储最新的边来提高采样图中三角形的数量从而提高三角形预测的准确性。但是 WRS 是只基于权重的算法，而无法支持更为普遍的无权重的三角形计数。而 SWTC 算法具有很好的扩展性，能够运用于普通的和权重的图。

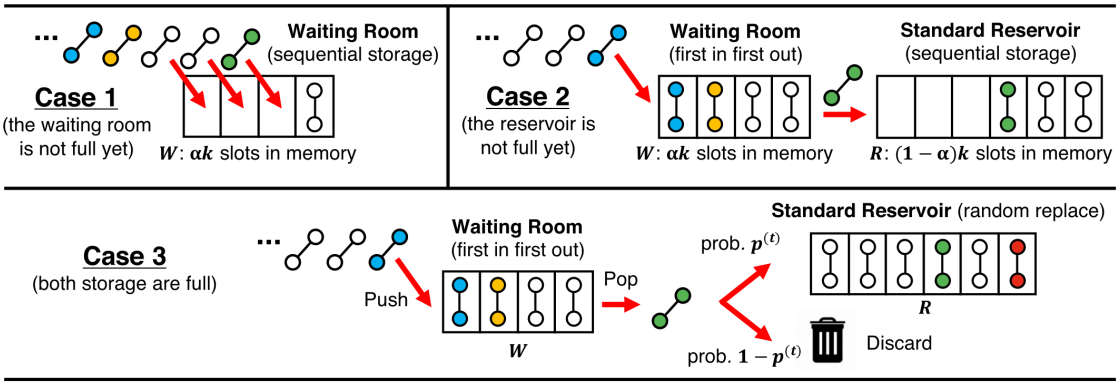


图 2: WRS 算法示意图

2.3 PartitionCT[4]

一种单次处理流的算法，在不存储整个图的情况下快速获取包含边重复的流式图中的三角形数量。为不同边随机分配不同的优先级，并将图流中的边哈希到 k 个桶（用户指定参数），重复边将被哈希到相同的桶中并拥有相同的优先级。但是 PartitionCT 无法用于具有边过期的基于滑动窗口的流式图。SWTC 沿用了 PartitionCt 中处理重复边的方法，并与其他方法结合使其能够用于处理具有边过期的流式图。

3 本文方法

3.1 本文方法概述

本文使用的方法是 SWTC 算法，基于滑动窗口的有重复边的图流，使用流图抽样得到一个子图，并根据子图中的三角形数计算出原图的三角形数。SWTC 方法分为两部分，一是采样方法，二是三角形计算方法。采样方法基于滑动窗口，并使用 slice 策略提高了边采样率，从而在三角形计算时得到更加精确的结果。三角形计算方法使用了 hyperloglog 算法，显著地提高了三角形计数的效率。

3.2 采样方法

使用固定长度切片策略，将流图的时间线划分为多个固定长度的切片（切片的长度与滑动窗口相同），每个切片的分割点称为 landmark，于是当前滑动窗口最多与两个切片重叠，具体有三种情况，如下图所示

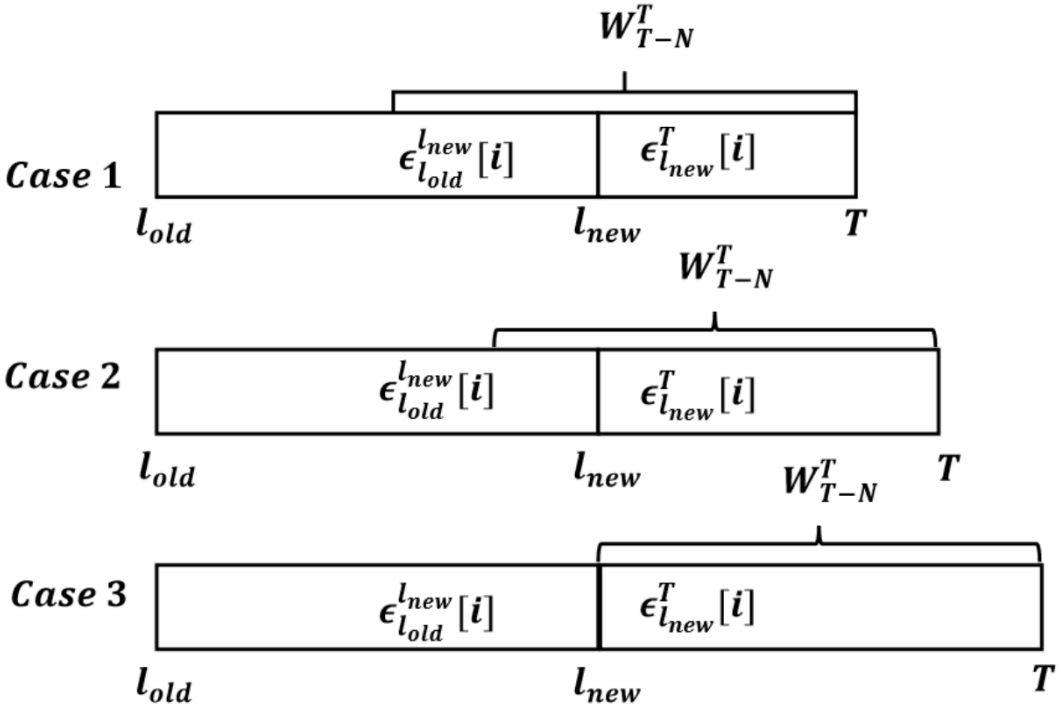


图 3: SWTC 采样方法示意图

其中 l_{old} 和 l_{new} 表示 slice 的分界点，$\varepsilon_{l_{old}}^{l_{new}}$ 表示 l_{old} 到 l_{new} 这一切片中的最大优先级。

Case1:

滑动窗口覆盖两个 slice，且两个 slice 最大边都在窗口内，此时比较优先级大小可选出采样边。

Case2:

滑动窗口覆盖两个 slice，但前一个 slice 的最大边不在滑动窗口内，即已过期。

若 $\varepsilon_{l_{old}}^{T}[i] \leqslant \varepsilon_{l_{old}}^{l_{new}}[i]$，则 $\varepsilon_{l_{old}}^{l_{new}}[i]$ 对应的边是滑动窗口内优先级最大的边，成为采样边。

若 $\varepsilon_{l_{old}}^{T}[i] > \varepsilon_{l_{old}}^{l_{new}}[i]$，则无法确定 $\varepsilon_{l_{old}}^{l_{new}}[i]$ 是否为滑动窗口内优先级最大的边，采样失败。

Case3:

滑动窗口与 slice 重合，此时窗口中最大边即为采样边，在下一时刻，新的 slice 到达。

令 $l_{old} = l_{new}$，即将之前最新的 slice 移动到前面，变为旧的 slice。

令 $l_{new} = \emptyset$，即后一个 slice 置空等待新的边达到。

若之前采样失败，即 Case2 中 $\varepsilon_{l_{old}}^{T}[i] > \varepsilon_{l_{old}}^{l_{new}}[i]$，此时没有 $\varepsilon_{l_{old}}^{T}[i]$ 继续限制 $\varepsilon_{l_{old}}^{l_{new}}[i]$，可将其设有效采样边。

SWTC 算法除了使用 hash 函数 G 为每条边生成优先级外，还使用另一个 hash 函数 H 将流图划分为 k 个 substreams（k 为用户指定参数），在每个 substream 中保存每个窗口具有最大优先级的边，故所有 substreams 的采样边共同组成最多 k 条边（理想情况，即全部采样成功）的采样图。注意在处理带有权值的流式图的边时，G 和 H 都要替换为随机函数，因为对于具有权值的边计算三角形时，不存在边重复问题，重复的边会被看成多个平行边。

SWTC 算法的采样方法的伪代码如下所示:

Procedure 1 Processing new edge in the SWTC

Input: edge e = (s,d)
Output: updated sample

```
p ← H(e)
if ε^T_{l_new}[p] = e then
    Update the timestamp of ε^T_{l_new}[p]
else
    if ε^T_{l_new}[p] = NULL or G(ε^T_{l_new}[p]) ⩽ G(e) then
        G_s.remove(ε^T_{l_new}[p]) /*call Algorithm 3*/
        ε^T_{l_new}[p] ← e
        if ε^{l_new}_{l_old}[p] = NULL or G(ε^{l_new}_{l_old}[p]) ⩽ G(e) then
            G_s.add(e) /*call Algorithm 2*/
            G_s.remove(ε^{l_new}_{l_old}[p]) /*call Algorithm 3*/
        end
    end
end
```

基于抽样的三角形近似计算的精度取决于样本图的大小，样本图 G_s 越大，估计结果越准确，也就是有效采样边越多，三角形估计的误差会越小。在吞吐量稳定的流式图中，

对于 baseline（BPS 算法），采样率为 $\left[\frac{|W^T_{T-N}|}{|W^T_{T-2N}|}, 1-\frac{|W^{T-N}_{T-2N}|}{|W^T_{T-3N}|}\right]$，即 50%≤p≤66%，平均接近 60%。

对于 SWTC，采样率为 $\frac{|W^T_{T-N}|}{|W^T_{l_{old}}|}=\frac{|W^T_{T-N}|}{|W^{l_{new}}_{l_{old}}+W^T_{l_{new}}|}$，即 50%≤p≤100%，平均值为 75%。

可以看到 SWTC 算法的采样率相比于 BPS 算法有一个较大的提升，故 SWTC 算法采样得到的子图能够更加准确地反映原图的结构，从而提高对原图三角形计数的准确度。

3.3 采样算法的优化 (Vision Counting)

在两个 landmark 的交界处，由于需要插入新的采样边，删除旧的采样边，更新 l_{old} 和 l_{new} 的关系，重新计算三角形数量等操作，会出现一个计算峰值导致算法阻塞（即上图的 case3），故原文使用了 vision counting 技术把所有的计算分散到整个窗口，避免在边界处出现计算峰值。

具体而言，为采样边设置一个候选边，把当前不是采样边但是有可能在未来成为采样边的边加入采样图，但此时这些边称为无效采样边，并在 vc 中计数（真正的采样边在 tc 中计数）当它成为了真正的采样边时，再将其在 tc 中计数，称为有效采样边。于是在 landmark 交界处，只需要将无效采样边变为有效采样边即可，相当于把计算峰值平均到了整个滑动窗口。优化后，边的插入和删除操作都有对应的更改。

3.4 三角形计数估算

设滑动窗口内不同边数为 n，采样图中的有效采样边数为 m。原始图流中一个完整的三角形被采样的概率 $P=\frac{m(m-1)(m-2)}{n(n-1)(n-2)}$。采样图中的三角形数量为 t_c。

由以上条件可得，原始图中三角形数量为 $\frac{t_c}{P}$，m 与 t_c 可以直接获得，这里的难点在于如何获取 n。

由采样率 $\frac{|W^T_{T-N}|}{|W^T_{l_{old}}|}=\frac{n}{|W^T_{l_{old}}|}=\frac{m}{M}$（M 为非空 substreams 的数量，由于 substream 为空的概率很小，即 $M\approx k$，故直接用 k 代替 M），可得 n=$\frac{m}{M}|W^T_{l_{old}}|$。

问题转化为如何计算 $\left|W_{l_{old}}^{T}\right|$，即滑动窗口中边的基数估计问题，由于使用了切片策略将其转化为固定区间内的基数问题，且每条边被赋予了优先级，可以直接将其转换为 Hypperlog Sketch 结构（最先进基数估计算法）进行计算。具体的计算公式为：$\left|W_{l_{old}}^{T}\right| = \frac{a_k k^2}{\sum_{i=1}^{k} 2^{-R[i]}}$

其中 $a_k = \frac{0.7213}{\left(1+\frac{1.079}{k}\right)}$，k>128，$R[i] = \lceil -\log(1-\theta) \rceil$，$\theta$ 为 $W_{l_{old}}^{T}$ 中的最大优先级。

4 复现细节

4.1 与已有开源代码对比

本文算法的复现工作参考了论文原文作者提供的源代码[5]，主要参考了原文对图流相关数据结构的设计，例如 node，nodetable，edgetable，sampletable，sampler 等模块，在理解作者设计的数据结构的基础上对相关结构进行了整理和重构，删除掉了一些冗余的模块和功能，例如 asy_sample，directe-setting 等，加入了一些必要的结构，如 edge 和 edge_cand 等。在算法上，在深入理解源代码的基础上，重新调整了函数的相关结构，整合功能相似的函数，删除不必要的函数，增加了一些公共函数用于各个算法之间的共同调用。修改之后算法的冗余度明显降低（由 3k+ 行代码变为 2k 行代码左右）。

4.2 实验环境搭建

实验在一台 16 核 CPU（11th Gen Intel(R) Core(TM) i7-11700 @ 2.50GHz）和 16G 内存的机器上，在 CentOS 环境下运行，所有代码使用 C++ 编写，使用 GCC4.8.5 进行编译。

4.3 界面分析与使用说明

```
int main()
{
    unsigned int s, d, w, w_;          // s为起点边的id，d为终点边
    long long t;
    string t1, t2, p;
    double time_unit = 20;         // 窗口长度的单位，时间戳的极差/
    int gap = 4000000;

    for(int i = 30; i <= 50; i+=5)
    {
        string filename = "../../data/actor_d";
        filename+=to_string(i);
        filename+=".txt";

        for(int hindex =0;hindex<5; hindex++) // 使用不同的hash
        hash函数对三角形精确计数没有影响，故每个不同的窗口长度只需要
        {
          int sample_size = gap*0.04;         //采样率4%
          long wsize = gap*time_unit;       //窗口长度（时间戳）
            long count = 0;                   //记录时间

          //读入数据，生成结果输出文件（必须先建立文件夹）
          ifstream fin(filename.c_str());
          string index = "ex4/4M_d";
          index += to_string(i);
          index += "_h";
          index += char(hindex+'0');
          ofstream fout(index.c_str());
```

图 4: SWTC 算法说明

如图打开 main 函数，图中红框为需要每次运行需要修改的参数。

time_unit 为时间单位，设为输入数据的时间戳的极差/边的数量。

gap 为窗口长度，根据数据集的边的数量进行设置，其中实验使用的数据集 Actor（约 30M 的边数）设置窗口长度为 4M，数据集 StackOverflow（约 60M）设置窗口长度为 5M。

sample_szie=gap*rate，sample_size 即为样本数量，rate 为采样率，原文推荐将采样率设为 4%-6%。

hindex 为算法使用的 hash 函数的索引，分别为 BobHash 和 MurmurHash。

filename 为输入数据集的文件路径（注意输入的数据集需要先根据时间戳进行排序）。

index 为输出结果的文件路径。

4.4 创新点

1. 使用了不同的 hash 函数用于映射 substreams 和优先级，比较与原算法效果的区别，实验结果与原算法相近。

2. 使用不同的数据集对算法效果进行测试，发现该算法对有些数据集不一定适用，即算法表现出来某方面的效果并没有原文中表现的那么好。

5 实验结果分析

5.1 数据集

Dataset	Node	Edge
Stackoverflow	2601977	63497050
Yahoo network	33635059	561754369
Actor	382219	33115812
FF	10^9	18311282

其中数据集 Stackoverflow 和 Yahoo network 有自带的时间戳，而 Actor 和 FF 数据集没有时间戳，故需要为 Actor 和 FF 数据集随机生成时间戳，FF 为一个符合幂律分布的合成的数据集。所有的原始数据集在输入到算法计算之前需要先对时间戳进行排序。

5.2 评估指标

为了观察算法运行过程中的各个指标的变化，故设置 checkpoint，每隔 1/10 窗口长度设置一个 checkpoint，在计算评价指标时，计算在所有 checkpoint 处计算得到的指标的平均值。同时，为了防止不同的 hash 函数对实验结果的影响，故每次实验都使用 5 个不同的 hash 函数，在计算结果时取其平均值。

实验使用三个评价指标，分别为

平均有效样本大小（Valid Sample Size）：当前的采样图中的有效采样边的数量。

平均有效样本百分比（Percentage of Valid Sample）：有效采样边数/子流数量，即采样率。

平均绝对百分比误差（MAPE）：在所有 checkpoint 对 APE=$\frac{|\hat{\tau}-\tau|}{\tau}$ 平均值得 MAPE，其中 $\hat{\tau}$ 为三角形估计值，τ 为实际三角形数量。

5.3 实验结果及其分析

5.3.1 实验 1：有效采样率对比

使用数据集 Actor 构造吞吐量稳定的数据流，过滤掉重复边，并安排时间戳使每个时间单位恰好有一条边，将滑动窗口长度设为 4M，采样率设为 4%，计算采样率。

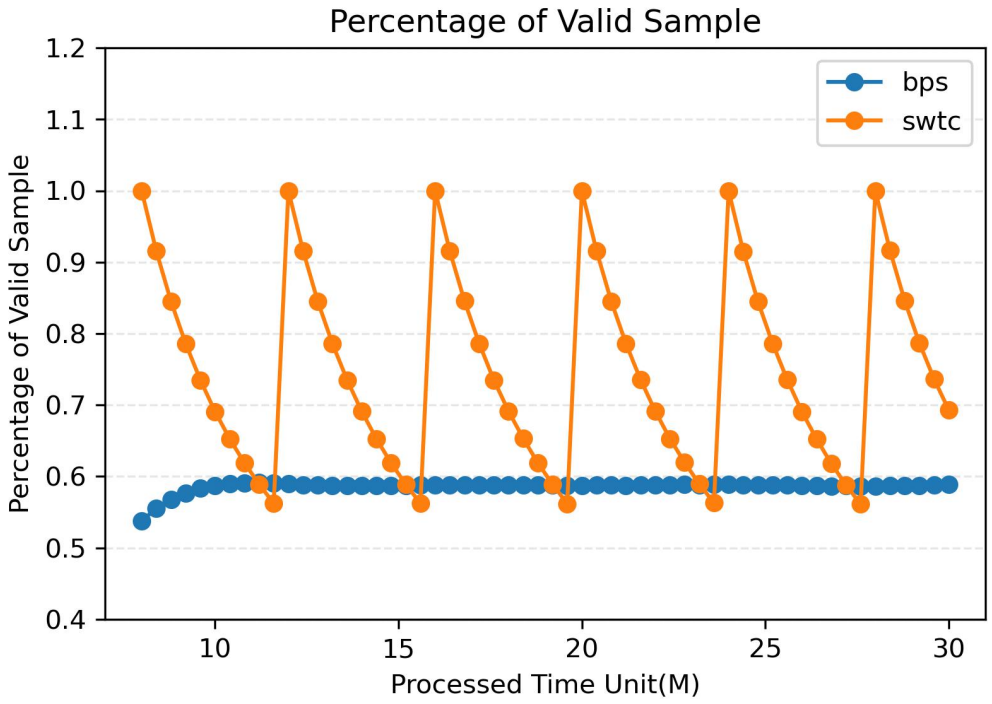


图 5: 有效采样率对比

实验结果如图所示，SWTC 算法的采样率随着时间的变化在 50% 到 100% 之间波动，大部分的检查点的采样率都比 BPS 算法高，只有几个检查点的采样率稍低于 BPS，平均采样率为 75%，而 BPS 的采样率变化不大，除了一开始的采样率较低，之后的维持在 60% 左右。故得到的结果符合前面对 BPS 和 SWTC 算法采样率的分析，同时与原文的实验结果一致。

由于本次实验将窗口长度设置为 4M，观察 SWTC 采样率的变化曲线，发现采样率每隔 4M 就会有一个从 100% 到 50% 的循环，说明窗口在移动时，刚开始窗口内边较少，采样率较高，随着采样边越来越多，采样率逐渐下降，直到窗口结束。

5.3.2 实验 2：窗口长度对结果的影响

对于数据集 Actor，固定采样率为 4%，窗口长度从 3M 到 6M，间隔 0.5M。对于数据集 StackOverflow，固定采样率为 4%，窗口长度从 4M 到 4.9M，间隔为 0.1M。计算有效采样边和 MAPE。

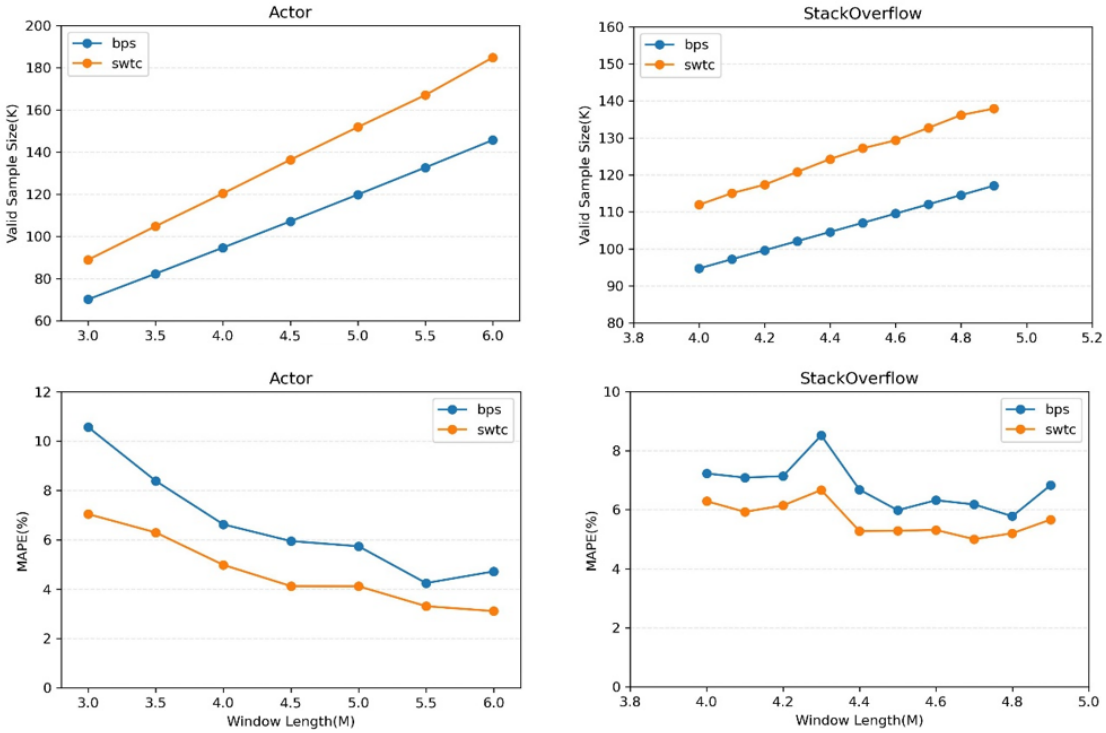


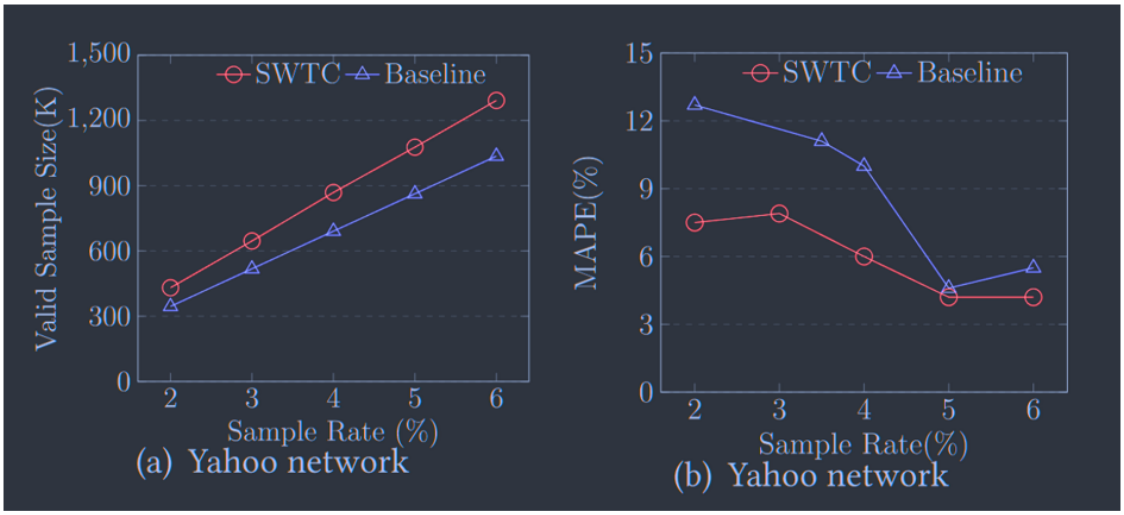
图 6: 不同窗口长度对比

实验结果如图所示，在有效采样边方面，可以看到有效采样边随窗口的增大而增大，与窗口的长度呈正相关，这是因为窗口长度的增加带来了更大的 k 值（采样率不变时，采样边数随窗口长度增加而增加），而 SWTC 总是比 BPS 算法有更大的有效样本量，且他们之间的差距随着采样率的变化也有所不同。平均而言，SWTC 中的样本量比 BPS 大 30%。

在三角形估算误差方面，在 Actor 数据集上，随着窗口长度的增加，MAPE 有减小的趋势，这是因为当窗口内的三角形较多时，随机性的影响减小，估计结果变得稳定且准确。而在 StackOverflow 数据集上，随着窗口长度的增加，MAPE 没有很明显的减小趋势，只是在一个固定范围内波动，趋于稳定。总体而言，SWTC 比 BPS 算法的误差小 60% 左右，且 SWTC 的误差均低于 10%。实验结果与原文一致。

5.3.3 实验 3：采样率对结果的影响

原文使用 Yahoo 数据集，固定窗口长度为 35M，采样率为 2% 到 6%，间隔 1%。本文使用 Actor 数据集，固定窗口长度为 4M，采样率为 2% 到 6%，间隔 1%。计算有效采样边和 MAPE。



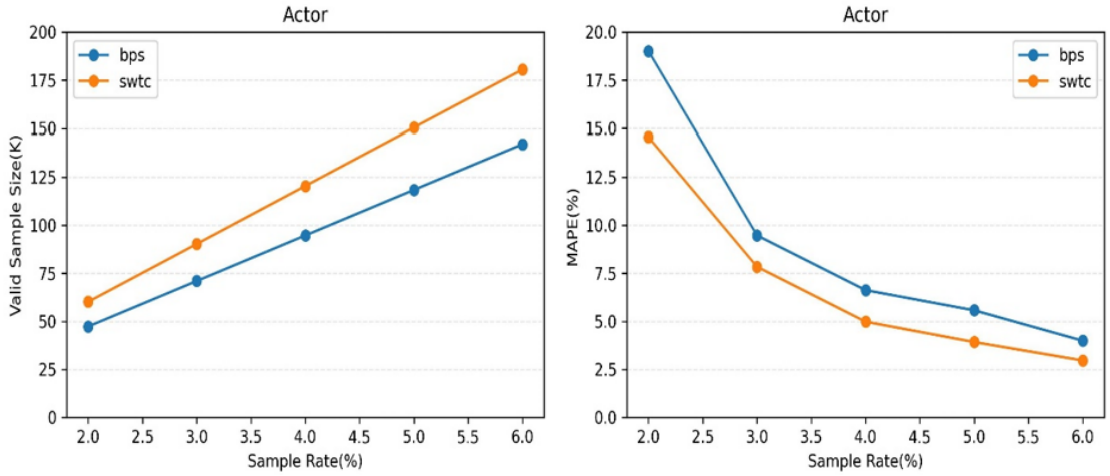
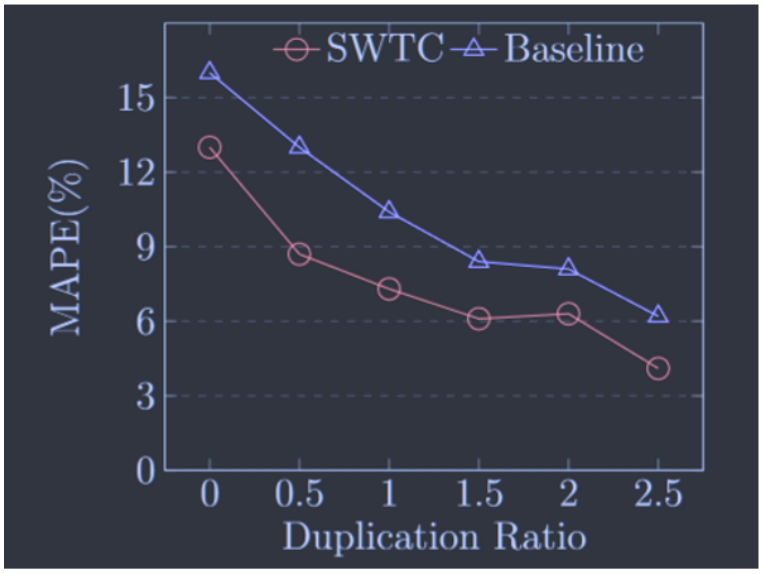


图 7: 不同采样率对比

如图所示，黑色背景的图为原文实验结果，白色背景的图为本文实验结果。在两个不同的数据集上，随着采样率的增加，有效采样边和 MAPE 都呈现出相同的趋势。首先是有效采样边与采样率呈正相关，与上一个实验一样，这也是由于 k 值，即采样边数的增加而产生的结果。随着采样率的增加，窗口内的有效采样边变多，使得被采样的三角形数量增加，从而减小了随机性和估计的误差，使结果趋于稳定和准确。同时，SWTC 算法的效果比 BPS 算法的效果好，但在不同的数据集上有所差异。例如在 Yahoo 数据集上，一开始 SWTC 的误差比 BPS 小很多，到后面两者的差距逐渐变小，甚至有一个点两者没有差距；而在 Actor 数据集中，两者在采样率较低的情况下，误差都很大（超过 10%），随着采样率的增加，误差快速降低，但两者之间的差距一直保持稳定。故在内存充足的情况下，最好将采样率定在 4% 以上，否则误差较大。

5.3.4 实验 4：重复率对结果的影响

原文基于 FF 数据集，窗口长度为 3M，采样率为 4%，生成重复率为 0% 到 2.5% 的符合幂律分布的合成数据集。本文基于 Actor 数据集，固定窗口长度为 4M，采样率为 4%，生成重复率为 0% 到 5% 的数据集，计算 MAPE。



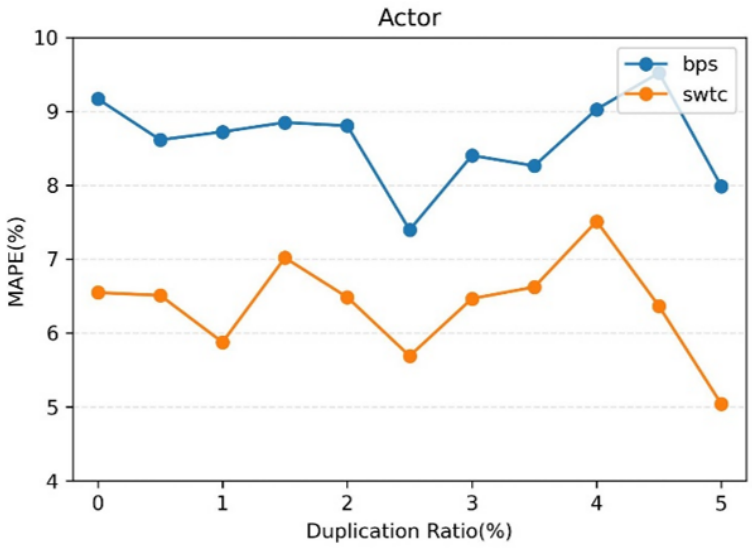


图 8: 不同重复率对比

如图所示，黑色背景的图为原文实验结果，白色背景的图为本文实验结果。在 FF 数据集上，MAPE 随着数据的重复率的增加而下降，这是因为数据的重复量增加导致流式图中互异边的数量相对减小，使得采样边的占比变大，相对数量增加，从而相当于间接提高了有效采样边的数量，导致 MAPE 降低。但是在 Actor 数据集上，随着数据重复率的增加，MAPE 并没有很明显的下降的趋势，而是在一个范围内波动，直到重复率达到 4% 之后才有下降的趋势，故不同的数据集对重复率的敏感程度不一样。总体来说，重复率的增加会导致三角形估计误差的降低。但是在有权重的图中，重复率对误差没有任何影响，因为重复边相当于多条互异的平行边。

6 总结与展望

具有重复边的基于滑动窗口的流式图三角形近似计算一直是一个未解决的问题。本文提出了 SWTC 算法，使用创新的固定长度切片策略对基于滑动窗口的流式图进行优先级采样，有效地提高了采样率，从而得到了更多的有效采样边，使三角形估算的精确度得以提高。理论分析和实际实验都表明，该方法生成的样本集更大，比现有的三角形计数算法准确性更高。

作为一个研一刚入门流式图方面研究的新生，阅读，理解和复现这篇论文使我收获良多，我不仅

学习了流式图采样，流式图三角形计数，基数估计等相关知识，还通过这篇论文完整的结构和详细的推理了解到了应该如何阅读，理解，甚至是撰写一篇论文；在实验，编码方面，本文作者给出的算法源代码使我受益匪浅，许多基本的问题，例如应该如何定义流式图的数据结构，应该如何将滑动窗口应用到图流中，都能在本文源代码中得到解答。

本文提出的 SWTC 算法，虽然效果较好，但是还是存在一些问题。例如算法需要手动设置滑动窗口长度和采样率，这些参数大多是通过经验性判断来进行选择，未来是否能够仅由用户指定内存大小来让算法自动决定窗口长度或采样率，且获得较好的采样和三角形计数效果。其次，SWTC 算法在三角形计数上的表现很好，但是在采样方面，获取到的采样图没有关注图的其他属性，例如节点度，图的连通性，是否可以通过在采样时考虑这些属性使得采样图更加接近真实图，从而帮助提高三角形计算的精度，这也是未来可以考虑改进的地方。

参考文献

[1] GOU X, ZOU L. Sliding Window-based Approximate Triangle Counting over Streaming Graphs with Duplicate Edges[J]., 2021: 645-657.

[2] GEMULLA R, LEHNER W. Sampling time-based sliding windows in bounded space[J]., 2008: 379-392.

[3] LEE D, SHIN K, FALOUTSOS C. Temporal locality-aware sampling for accurate triangle counting in real graph streams[J]. VLDB J., 2020, 29: 1501-1525.

[4] WANG P, QI Y, SUN Y, et al. Approximately Counting Triangles in Large Graph Streams Including Edge Duplicates with a Fixed Memory Usage[J]. Proc. VLDB Endow., 2017, 11: 162-175.

[5] Source code of swtc and the baseline method.[J/OL]., https://github.com/%20StreamingTriangleCounting/TriangleCounting.git..