LSketch：面向时间敏感查询的标签增强型图流草图

Yiling Zeng、Chunyao Song、Yuhan Li 和 Tingjian Ge

摘要——图流表示现实应用中的数据交互。图流挖掘在网络安全、社交网络分析和交通管控等领域发挥着重要作用。然而，图流数据量庞大且动态性强，给其高效存储及后续查询分析带来了巨大挑战。现有研究采用草图对图流进行概括。我们提出了适用于异构图流的 LSketch，它能够有效保留现实场景中图流携带的标签信息，从而增强草图的表达能力。此外，随着图流随时间不断演变，过旧的边可能会失去实际意义。因此，我们将滑动窗口模型引入 LSketch，以自动移除过期边。LSketch 使用次线性存储空间，并能够以较高的准确率支持基于结构的查询和时间敏感查询。我们在四个真实数据集上进行了大量实验，结果表明，在查询准确率和时间效率方面，所提出的方法优于当前最先进的方法。

索引词—异构图流，图摘要，滑动窗口，近似查询。

arXiv:2304.02897v1 [cs.DB] 6 Apr 2023

1 引言

在N大数据时代，现实世界网络中的数据正以越来越快的速度生成。截至2022年第一季度，脸书拥有29.1亿月活跃用户，其中79%的月活跃用户每天都会使用脸书 [1]。这些活跃用户在平台上产生海量数据。在此基础上，许多可用图建模的现实应用得到了广泛研究[1], [2], [3]。然而，过去的大多数研究更关注静态图。近些年来，针对不同应用场景的需求，人们提出了基于大规模图基础设施的流 [4],[5], [6], [7], [8]。这类图流以边的形式按序更新，用于对图进行建模，是静态图的扩展，可模拟现实场景中网络的持续演化。与静态图挖掘类似，大规模图流分析具有重要的实际意义。例如，在社交网络中，探索节点之间的连接有助于预测用户可能的好友，或检测假新闻的源头；在交通网络中，拥堵路段预测和路线规划也能从此类分析中受益。

由于图流规模庞大且变化速率过高，通常无法准确计算边或节点的频率。因此，近似查询被用于解决这一问题，并受到越来越多的关注。在以往的研究中，已经提出了一系列适用于近似处理的数据结构，包括布隆过滤器 [9] 及其变体 [1], [2], [8]，和草图 [10], [11], [12], [13], [14], [15]。此类数据结构能够显著减少存储空间，并在准确率损失有限的情况下实现极快的查询速度，是大数据应用的有效方法，尤其适用于流式场景。目前，它们已广泛用于查找 top-k 项 [16],[17]、查找重频项 [18], [19]、近似权重估计 [20], [21]以及三角形计数 [22], [23]。

此外，异构图中的节点和边承载着丰富的信息。忽略这些信息可能导致严重的知识损失。例如，在社交网络中，用户可以视为节点，边则表示用户之间的通信/交互。用户可以根据兴趣划分到不同的社群，因此拥有各种个人标签——即节点上的标签。用户之间的通信也可以根据通信强度进行分类，例如频繁、中等和不频繁。然而，目前大多数草图技术只考虑不带标签的节点和边，而实际应用中的数据往往在节点和边上带有标签。少数现有研究的查询准确率较低，无法满足实际场景的需求。

此外，大多数现有的草图技术在维护草图时既未考虑插入速率的影响，也未区分在不同时间戳到达的项目。一方面，图流的更新频率代表了网络的活跃状态，因此草图应能够携带与时间相关的信息。另一方面，在不同时间戳到达的项目在实际应用中具有不同的重要性。越接近当前时间，项目的价值越高。

总而言之，标签和时间戳都是现实世界图流的重要特征，因此我们应该构建一种能够很好地处理异构图流的时间敏感草图。

• *Y. Zeng*、*C. Song* 和 *Y. Li* 任职于南开大学计算机学院天津市网络与数据安全技术重点实验室，地址：中国天津市 *300350*。电子邮件：{*yilingzeng, yuhanli*}*@mail.nankai.edu.cn, chunyao.song@nankai.edu.cn*

• *T. Ge* 任职于马萨诸塞大学洛厄尔分校计算机科学系。电子邮件：*ge@cs.uml.edu*。

1. https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/

1.1 相关工作

由于数据的快速增长，图流模型已被用于数据分析和查询处理。McGregor对图流领域主要是理论性的研究进行了出色的综述 [24] 。概要构建问题已在数据流领域得到广泛研究。早期的图流摘要技术通过多个哈希函数将数据项进行线性投影，映射到更低维的空间中，以独立存储每个数据项，但忽略了数据项之间的连接。然而，这类草图并不适用于图数据场景。它们没有考虑图流中边的潜在关联，而是将边建模为一系列相互独立的数据项，因此无法保留图流的底层结构。因此，它们只能支持边查询，而不能支持任何基于拓扑的查询。这一系列工作包括CountSketch [25],CountMin Sketch [26], gSketch [27],AMS [10],Ada-sketch[28] 等。

为克服这些早期研究的缺点，研究人员提出了图草图，以改进上述用于图流的经典草图。TCM [29], gMatrix [30] 和LGS [31] 使用三维草图来支持基于图结构的查询。它们应用哈希函数，将图的顶点集映射为矩阵宽度范围内的一个整数，从而可以利用其两个端点的哈希值将该项（即一条边）定位到矩阵中。尽管它们能够支持基于图拓扑的查询，但仍然存在准确率较低的问题。为解决这一问题，GSS[32], [33] 引入了一系列技术，例如平方哈希和多个房间，以处理顶点度分布不均的问题，从而提高矩阵的容量，并实现迄今为止最高的查询准确率。

然而，上述大多数方法都是针对同质图流进行研究的。相比之下，实际应用中的数据通常携带大量标记信息。此外，这些草图不具备时间敏感性，无法判断网络的活跃状态。过于陈旧的边无法及时移除，这将影响后续查询。近年来，Hung 等人 [34]研究了在加权数据流的滑动窗口中识别重权重项目的问题。ECM-草图 [35] 允许在基于时间和基于计数的滑动窗口上有效汇总流数据

以概率准确率保证回答潜在的复杂查询。SBG-Sketch [36] 概括带标签图流，并自动平衡草图负载，以应对不可预测且高度不均衡的边标签频率。LGS [31]扩展TCM，通过引入滑动窗口模型来保留项目的时间戳，并自动处理边过期。此外，它能够高效存储顶点标签和边标签，但其查询准确率还不足以满足实际使用需求。

总之，我们选择 GSS（适用于同构图流且准确率最高的草图）和 LGS（适用于异构图流且准确率最高的草图）作为竞争方法，以展示我们的草图在保留标签信息和时间信息方面的优势。

1.2 我们的贡献

为解决上述问题，构建一种面向异构图流、带有滑动窗口的图摘要具有重要的实际意义；它能够支持与图拓扑相关的查询，同时实现高效率和高准确率。

我们的贡献总结如下：

1)我们提出了面向异构图流的 LSketch，它采用存储块划分和双计数器的思想存储顶点标签和边标签，而无需占用太多额外的存储空间。

2) 我们将由k 个子窗口组成的滑动窗口模型引入 LSketch。通过跟踪最新子窗口的开始时间，我们可以高效地自动处理边过期，使草图能够有效存储项目的时间信息。

3) 具体而言，我们设计了两种顶点标签编码策略，分别称为统一分块和偏斜分块。特别是，偏斜分块策略能够适应顶点标签的极端不均衡分布，并实现相近的高效率。

4) 我们展示了基于结构的查询在 LSketch上的一些示例实现，并在四个真实世界数据集上进行了广泛实验。结果表明，在理论保证下，LSketch 的查询准确率比 LGS 高出 1至 3 个数量级，同时仍能保持与其相当的查询效率。此外，它支持的查询类型多于 GSS。

2 预备知识

2.1 异构图流

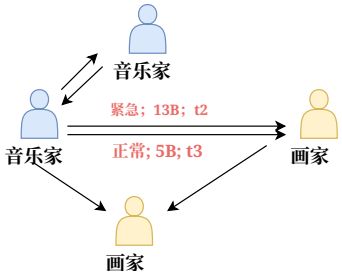


图 1. 社交网络中的异构图流

异构图流是一种实时更新图，由以条目形式到达的边构成 $e = (A, B; l_A, l_B, l_e; w; t)$，这意味着边 (A, B) 在时间 t 到达，权重为 w。此外，l_A 和 l_B 分别表示节点 A 和 B 的标签属性，而 l_e 表示边标签。随着时间推移，同一条目可能以不同权重多次出现。我们以图 1 为例。在社交网络中，用户之间的互动可以建模为边，因此用户之间各种互动的演变构成了图流，其中用户所属的社群可以作为其顶点标签，例如音乐家或画家。此外，边标签可以表示用户发送消息时选择的紧急程度。这意味着同一对节点 A 和 B之间可能出现带有不同标签的边，标签分别为 l_A 和 l_B。一条消息对应图流中的一个条目／边，

表 1 论文中使用的符号列表

符号	定义
e	图流的到达项
A/B	项 e的起始/结束节点
l_A/l_B	节点 A/B的顶点标签
l_e	item e的边标签
w	item e的权重
t	item e的时间戳
d	存储矩阵的宽度
b	存储块的宽度
m	某个顶点标签对应块的索引
$H(v)$	节点 v的计算哈希值
$f(v)$	节点 v的计算指纹
$s(v)$	节点 v的计算地址
r	采样序列的长度
s	采样单元的长度
$l_i(v)$	节点 v的地址候选列表
C	计数器 C 用于无标签限制的查询
P	计数器 P 用于有标签限制的查询
c	预定义素数列表的长度
W	整个滑动窗口的时间单位
W_s	一个子窗口的时间单位
k	子窗口的大小

其权重为消息的长度。本文中常用的符号列于表 1。

给定图流，我们可以定义其底层结构图 $G=(V,\ E)$，它是一个动态有向图，会随着每条边数据的到达而持续更新。这里，V 是顶点集，E 是边集。对于任意顶点 $v_i \in V$，都附有一个顶点标签 $l_V(v_i)$。同样，对于任意边$e_i \in E$，都附有一个边标签 $l_E(e_i)$。如前所述，图流中的边可能会多次出现，其权重和标签可能相同，也可能不同；因此，我们允许同一对节点之间存在多条边。

由于图流规模庞大且动态性很高，其存储和处理通常难以应对。传统数据结构（例如邻接矩阵）也无法直接应用。因此，我们需要构建一种实时图摘要模型——图摘要草图，以更少的存储空间和更快的响应速度处理图流。

底层结构图 $G=(V,\ E)$ 的图摘要草图表示为 $G_s=(V_s,\ E_s)$，其中 V_s 是顶点集，E_s 是边集。理想情况下，V_s 应显著小于 V，E_s 也应显著小于 E。为了实现减少存储空间并支持基于结构的查询这一目标，需要设计良好的数据结构，将图流中的边映射到草图中。

2.2 GSS：一种支持高准确率同构图流的新型草图

我们首先介绍基础草图 GSS [33]，它能够以高度压缩的方式存储同构图流。

GSS 将生成的 F 以及宽度为 d 的矩阵构成了底层数据结构。当数据流中到达一个项 $e=(A,\ B;\ w)$ 时，它会应用哈希函数 $H(\cdot)$，并将节点映射到一个值域 $[0, D)$ 中（D 的值会影响草图的准确率）。随后，我们分别得到对应于两个顶点的哈希值 $H(A)$ 和 $H(B)$。

指纹技术。GSS 将哈希值拆分 $H(v)$

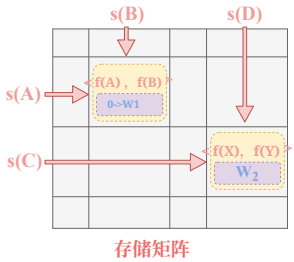


图 2. 存储矩阵的运行示例。项 (A, B) 成功插入一个空单元格，而项 (C, D) 到达一个占用单元格，插入失败。

将顶点 v 转换为地址 $s(v)(0 \le s(v) \le d)$ 和指纹 $f(v)(0 \le f(v) \le F)$，其中 $s(v) = \lfloor \frac{H(v)}{F} \rfloor$，$f(v) = H(v)\%F$，且 F 表示指纹的大小（例如，$F=1024$ 表示 10 位指纹）。同时，它将 D 设为 $d*F$，这是宽度为 d 的矩阵能够准确表示的最大哈希范围。通过地址对（$s(A)$ $s(B)$），GSS 可以定位矩阵中应存储该项的单元格。除了权重 w，它还存储指纹对（$f(A)$ $f(B)$）作为唯一标识符，以防止该项在后续更新中因哈希冲突而被覆盖。

图 2 展示了处理数据项时实际操作的一个示例。如果单元格非空，或两个指纹对不匹配，则该边无法存储在存储矩阵中。对于此类边，GSS 还添加了一种名为缓冲区 的数据结构，详情如下。

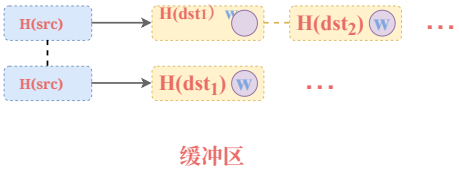


图3. 缓冲区的结构

缓冲区。由于缓冲区旨在存储剩余的边，GSS 直接使用邻接表。如图 3 所示，其中顶点 v 的哈希值 $H(v)$ 用作其标识符。总而言之，GSS 由一个优化的存储矩阵和一个用作缓冲区的邻接表组成。

孪生单元。如前所述，一个矩阵单元只存储指纹相同的边。插入过程中发生指纹不匹配时，该边会被放入缓冲区。为了提高存储矩阵的插入成功率，GSS 引入了孪生单元，在不增加矩阵存储空间的情况下将其容量加倍。

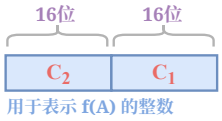


图 4. GSS 中孪生单元的示意图

具体而言，它将单元格中的所有配置（包括一对指纹及其对应的权重）划分为两个孪生段，C_1 和 C_2（即两个子段）。例如，我们可以用 16 位分别表示

C_1 和 C_2的指纹，并将它们拼接到一个 32 位字中。因此，如图 4 所示，每个孪生段 C_1 和C_2 都能够存储一个唯一项。当一条边通过哈希和指纹定位到某个单元格后，GSS 会依次检查较低段 C_1和较高段 C_2 的状态，以进行插入。一旦C_1 已被占用而 C_2 可用，就需要使用位运算将较低段 C_1 与新插入的较高段C_2 合并，然后更新相应的权重。通过这种方式，GSS 能够减少查找桶时的内存访问次数，同时不增加存储开销，从而提升整个草图的插入和查询效率。

平方哈希与采样。为进一步缓解节点度偏斜所造成的矩阵拥塞，GSS 不再坚持将一个顶点映射到固定的行或列。相反，它为每条边生成一系列映射单元格并尝试插入，从而通过扩展顶点的映射选择，实现边在存储矩阵中的去中心化分布。为实现**独立性** [26], [30] 和**可逆性**这两个属性，GSS 采用线性同余法 [37]生成平方哈希。GSS 将 $f(v)$ 设为种子，以生成长度为 r 的随机值序列，可表示为

$$\begin{cases} l_1(v) = (T \times f(v) + I)\%M \\ l_i(v) = (T \times l_{i-1}(v) + I)\%M \quad (2 \le i \le r) \end{cases} \tag{1}$$

其中，乘数 T、增量 I和模数 M是用于确定生成器的整数常量。适当的参数可以生成一个不含重复值的序列，其周期远大于 r。基于该随机值序列，GSS 得到地址候选列表如下:

$$\{s_i(v) | s_i(v) = (s(v) + l_i(v))\%d, 1 \le i \le r\} \tag{2}$$

序列的长度 r 主要根据数据集的偏斜程度设定。如果检查全部 $r \times r$ 个单元格，过程将十分耗时；而在理想情况下，大多数边无需遍历所有这些单元格。因此，GSS 不会检查所有映射单元格，而是再次使用线性同余法，从中选择 s 个桶作为采样单元。它计算一条边两个端点的指纹之和来表示该边，并使用该和值作为种子生成以下序列:

$$\begin{cases} Sp_1(e) = (T \times (f(A) + f(B)) + I)\%M \\ Sp_i(e) = (T \times Sp_{i-1}(e) + I)\%M \quad (2 \le i \le s) \end{cases} \tag{3}$$

序列 Sp中的每个值随后使用与指纹技术相同的方法转换为两个端点的地址下标，从而

$$\begin{cases} A_i = \lfloor \frac{Sp_i(e)}{r} \rfloor \%r \\ B_i = Sp_i(e)\%r \quad (1 \le i \le s) \\ Sampled\,cell_i = (s_{A_i}(A), s_{B_i}(B)) \end{cases} \tag{4}$$

其中， A_i 表示起始顶点的地址候选列表中选定的下标，而 B_i 表示结束顶点对应的下标。

3 LSKETCH 的构建

虽然 GSS 在同构图流上表现良好，但无法支持基于异构图流的查询，而这类查询在实际场景中更为常见。因此，我们提出在 GSS 的基础上构建能够适应异构图流的草图。本节将逐步介绍如何构建 **LSketch** （即带标签的图流草图）。首先，我们将介绍如何存储带标签信息，以及如何应用滑动窗口来着重保留对时间敏感的信息。然后，我们将展示伪代码和运行示例，以更好地说明 LSketch 在实践中的工作方式；最后，我们将介绍 LSketch 的改进方法，以满足不同类型数据集的需求。

3.1 标签启用矩阵

如第 1 节所述，节点和边的标签在分析基于图的应用时发挥着重要作用。现在，我们讨论如何在 GSS 的基础上纳入标签。对于异构图流，边的形式为 $e = (A, B; l_A, l_B, l_e; w)$，其中 l_A 和 l_B 分别是节点 A 和 B 的标签，而 l_e 表示边标签。存储块划分。草图的设计着眼于查询效率和较低的存储空间开销。我们的核心思路是将具有相同顶点标签的节点聚集在一起，从而无需占用额外空间即可对顶点标签进行编码。因此，我们将存储矩阵划分为 $n \times n$ 个块（$n = d/b$），其中 d 表示整个矩阵的宽度，与 GSS 中的定义相同；每个块都是宽度为 b 的子矩阵。对于一个输入项 e，我们首先使用哈希函数 $H(\cdot)$ 将两个顶点标签 l_A 和 l_B 映射到取值范围 $[0, n)$，以定位一个存储块。然后，我们再次使用 $H(\cdot)$ 将 A 和 B 映射到 $[0, b)$ 中的值，以定位所选块内用于存储 e 的单元格 E。

通过上述两级哈希技术，我们不仅可以在不增加存储开销的情况下完成顶点标签的编码，还能确保草图可以非常高效地支持后续基于顶点标签的聚合查询。图 5 展示了存储块的概念。

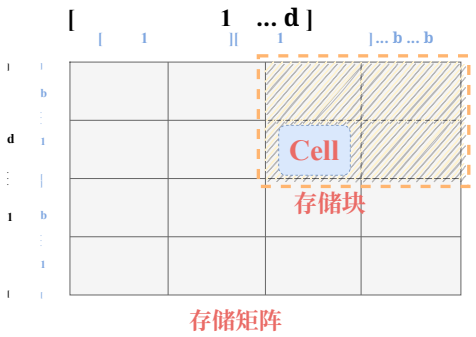


图 5. 存储块示意图

双计数器。如第 2.1 节所述，同一对节点之间可能会多次出现一条边缘，并且在不同时间可能具有不同的权重或标签。图 G 中一条边缘的总权重，是所有具有相同端点对的条目权重之和。在

为了有效保留异构信息，我们的草图必须能够区分边标签，并存储不同边标签各自对应的权重。

我们提出了双计数器机制，通过在矩阵单元 E中维护两个不同的计数器，准确记录不同类型的边权重。第一个计数器 C 记录命中目标单元格的边数。该计数器用于有效回答无标签限制的查询。第二个计数器 P 记录素数的乘积，以便用一个数高效编码一条或多条不同类型的边。为此，我们给出一个长度为 c的素数列表，并将每个不同的边标签映射到唯一的素数。对于一个传入项 e 及其边标签 l_e，我们使用 $H(\cdot)$ 将 l_e 映射到值范围 $[0, c)$，从而获得该特定边标签的素数表示。然后，将选定的素数乘入计数器P，使权重叠加在一起。由于素数乘积分解的唯一性，我们可以通过计数器 P 轻松获得与每个边标签对应的权重。因此，该草图可以以非常节省空间的方式处理多重边。请注意，在未引入滑动窗口时，我们会通过设置阈值，将计数器 P处理为一个大数列表。在实际场景的应用中，在合理的窗口大小下，计数器 P 通常由一个大数存储。双计数器机制适用于无权图流和有权图流（权重必须为整数）。对于有权图流，我们只需根据权重重复上述操作，具体示例将在示例2中说明。

3.2 融入时间戳的矩阵

在实际应用中，图流以很高的速度更新。随着图流不断演变，旧边的存在会对当前时刻的数据分析产生不利影响。滑动窗口是一种常用技术 [38], [39], [40]。因此，我们设计了一种滑动窗口方案来自动处理边的删除，确保图流的时效性，以支持后续的时间敏感型查询。

滑动窗口。假设滑动窗口的大小为 W 时间单位，我们的草图仅维护在 $t-W$之后到达的项目，其中 t 是当前时间，而过旧的项目将被自动移除。现实世界的查询

大多与时间段有关，而非特定时间点。因此，我们根据图流分析的粒度需求，将滑动窗口细分为 k个子窗口。已知 W 和 k后，我们可以轻松计算出每个子窗口的大小为 $\frac{W}{k}$。为了以节省存储空间的方式存储所需信息，无需记录每个项目的时间戳。相反，我们提出存储一个“lastT”时间戳，用来表示最近一个子窗口的开始时间。尽管整个草图中只记录了一个时间戳，但我们仍能根据子窗口的大小和整个窗口的长度推断出整体时间范围。设最近一个子窗口的开始时间为 t_n。每当当前时间 $t \geq t_n + \frac{W}{k}$，我们就开始一个时间为 t 的新子窗口，并移除最旧的子窗口。通过这种方式，我们可以简洁有效地支持具有预定义粒度的滑动窗口。图 6 展示了滑动过程。

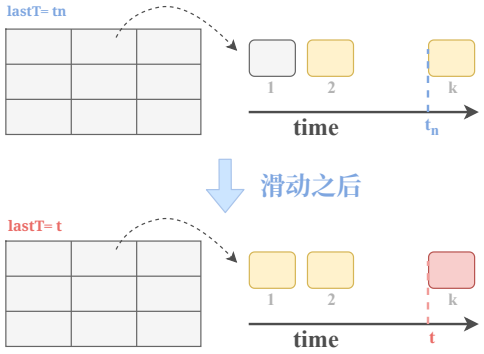


图 6. 存储矩阵的滑动过程

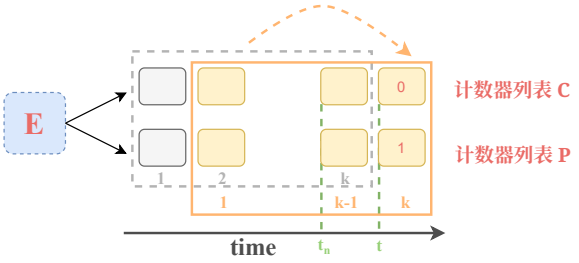


图 7。引入滑动窗口后的矩阵单元数据结构

从图 7 可以看出，双计数器机制可以轻松地与滑动窗口集成。对于每个子窗口，我们分配两个计数器，用于记录单元格的总权重以及不同边标签各自的权重。因此，在每个矩阵单元中维护两个长度为 k 的计数器列表。

3.3 附加池

除存储矩阵外，我们还需要一种名为附加池的结构，用于容纳矩阵中的冲突项。与 GSS 中使用的邻接表缓冲区类似，附加池也应是一种“兜底”结构。具体而言，我们重新设计了附加池的结构，如图 8 所示，其中顶点的哈希值 $H(v)$ 作为其标识符。我们使用邻接表存储顶点之间的指向关系，并通过数组和双端队列存储索引（idx），以维护顶点之间的权重。这样既能快速滑动子窗口，也能确保附加池与存储矩阵同步滑动。

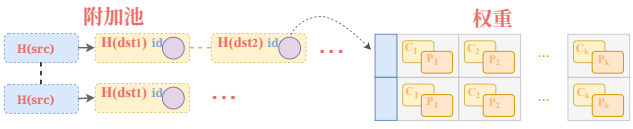


图 8. 附加池的结构

—

3.4 LSkecth：完整框架。

LSketch 的各个关键组件均已就绪，接下来我们将看看它们如何协同工作。受 GSS 的启发，我们将孪生单元和平方哈希技术引入 LSketch，以进一步提升存储矩阵的容量，从而应对节点度数偏斜导致的空间消耗不均问题。如果提高矩阵中条目的插入成功率，LSketch 的整体更新效率和查询效率将显著提升，因为遍历附加池时需要进行列表搜索，效率较低。引入孪生单元策略相对简单。我们只需确保将单元格中的指纹对及其对应的计数器列表划分为两个孪生段，如图 9 所示。接下来，我们将介绍如何应用平方哈希策略。

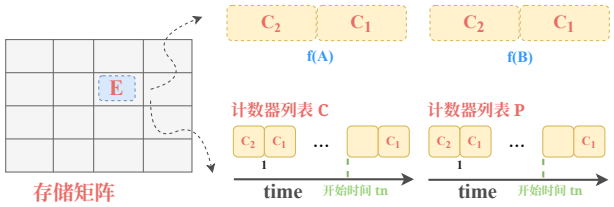


图 9. 矩阵中的孪生单元示意图

当一个项目到达时，我们首先使用第3.1节所述的相同方法，通过两个端点的顶点标签定位某个存储块。所有后续分析均限制在此存储块内。

简而言之，对于一个顶点 v，我们使用哈希函数$H(\cdot)$获得其哈希表示 $H(v) = (s(v)\ f(v))$。然后，参考 GSS 的思想，我们使用线性同余方法 [37] 为 v生成地址候选列表 $\{s_1(v)\ s_2(v)..., s_r(v)\}$，其中 $0 \le s_i(v) < b$。因此，根据两个端点的候选列表，边(A, B)有 $r \times r$ 个候选单元格可插入选定的存储块中，可表示为 $\{(s_i(A)\ s_j(B))|1 \le i \le r, 1 \le j \le r\}$，其中 $s_i(A)$表示行坐标，$s_i(B)$表示列坐标。一旦尝试成功，该边将存储在第一个有效单元格中；否则需要将其放入附加池中。

显然，为了成功执行项标识匹配及后续查询，矩阵单元需要记录地址候选列表的选择，我们将其定义为索引对 (i_r, i_c)。这意味着，对于起点/终点顶点 A/B，选择了第 i_r/第 i_c 个地址，因此该项被存储在存储块的 $(s_{i_r}(A)\ s_{i_c}(B))$ 位置。

需要指出的是，启用孪生单元策略后，每个单元格的检查都涉及对两组索引对和指纹对进行匹配尝试。我们使用图 10 展示包含候选项列表的模型。为清晰起见，我们省略了孪生单元策略，并
此处的滑动窗口模型。选定的存储块如图 10 右侧的绘图所示，其中 r 设置为 2，共得到 4 个候选单元格。

熟悉了所有组成部分后，我们就可以展示完整的算法了。算法 1 展示了

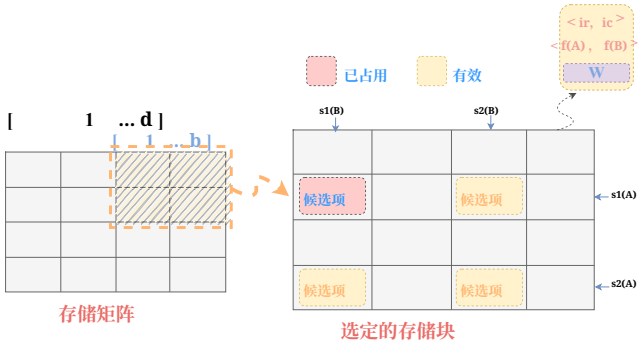


图 10。候选项列表示意图

算法 1： 预计算

输入： 一个顶点 A 及其对应的标签 l_A
输出： 哈希值 $H(A)$，初始地址 $s(A)$，指纹 $f(A)$，以及候选项列表 $l_i(A)(1 \le i \le r)$

1 初始化存储矩阵的宽度 d，存储矩阵的宽度 存储块的宽度 b，指纹长度 F，候选项列表长度 r，以及哈希函数 $H(\cdot)$
2 $m_A = H(l_A)\%\frac{d}{b}$
3 $s(A) = \lfloor \frac{H(A)}{F} \rfloor$
4 $f(A) = H(A)\%F$
5 $l_1(A) = (T \times f(A) + I)\%M$
6 **对于** $i \leftarrow 2 \ldots r$ **执行**
7 $l_i(A) = (T \times l_{i-1}(A) + I)\%M$
8 返回 $H(A)$, $s(A)$, $f(A)$，以及 $l_i(A)(1 \le i \le r)$

如何使用顶点标识符和标签计算地址候选列表。首先，根据顶点标签计算某一维度上存储块的索引（第 2 行），然后计算顶点的初始地址和指纹（第 3–4 行）。利用公式 1 和 2 生成给定顶点的地址候选列表（第 5–7 行）。

算法 2 展示了如何将项插入 LSketch 的完整过程。我们主要展示将项插入存储矩阵的过程，而与附加池相关的插入过程类似，但更简单。我们首先初始化 LSketch（第 1–4 行）。然后，每当有项到达时，我们便处理该项的插入。首先，根据传入项携带的时间戳，我们检查是否需要启动新的子窗口并执行窗口滑动（第 6–9 行）。然后，我们使用其他信息查找该项可能的插入位置。对顶点 A 和 B 调用算法 1，即可得到它们的指纹和候选项列表（第 10 行）。

在此之后，我们根据第 2.2 节前文所述的策略生成采样序列（第 11—15 行）。最后，我们依次检查 s 单元格。如果遇到空单元格，则直接插入；否则，只有当该单元格的索引对和指纹对与待插入项完全匹配时，我们才会更新其值（第 16—23 行）。插入或更新完成后，本轮即告结束。否则，如果上述所有尝试均失败，我们会将该项放入附加池中（第 24—25 行）。

接下来，我们通过一些 LSketch 示例，更好地说明算法的工作原理。

算法 2: LSketch：带标签的 图流草图

输入： 一个图流 G

输出： LSketch S_k，配备存储矩阵 S 和一个额外池 AP

初始化存储矩阵的宽度 d，以及存储块的宽度 b、指纹长度 F，以及候选列表长度 r 及其采样长度 s，并设置窗口大小 W 和子窗口大小 W_s。此外将最新子窗口的开始时间 t_n 设为当前时间 t_c。
$S \leftarrow$ 一个空草图
$P_r \leftarrow$ 一个预先定义的素数列表，长度为 c
$k \leftarrow W/W_s$，子窗口的数量
对于每个到达项 $e=(A,\ B;\ l_A,\ l_B,\ l_e;\ w;\ t)$ **执行**
  如果 $t_n + W_s \le t$ **然后**
    对于 $i \leftarrow 2 \dots k$ **执行**
      $S[i-1] \leftarrow S[i]$
    $t_n = t$
  $Precompute(A, l_A)$, $Precompute(B, l_B)$
  $Sp_0(e) = f(A) + f(B)$
  为 $i \leftarrow 1 \dots s$ **执行**
    $Sp_i(e) = (T_e \times Sp_{i-1}(e) + I)\%M$
    $A_i = \lfloor Sp_i(e)/r \rfloor \%r$, $B_i = Sp_i(e)\%r$
    $p_1 = (s(A) + l_{A_i}(A))\%b$, $p_2 = (s(B) + l_{B_i}(B))\%b$
    $E = S[m_A \times b + p_1][m_B \times b + p_2]$
    依次遍历孪生单元格
    if 存储的索引对和指纹对匹配要插入的那些，或者单元格为空 **那么**
      $P_e = P_r[H(l_e)\%c]$
      对 $i \leftarrow 1 \dots w$ **执行**
        $E[k].C = E[k].C + 1$
        $E[k].P = E[k].P \times P_e$
      $S[m_A \times b + p_1][m_B \times b + p_2] = E$
  如果存储矩阵中的所有插入尝试均失败 **则**
    将 e 插入附加池 AP
返回 S

示例 *1.* 假设一个图流由以下项组成：*(a, b; l_2, l_1, l_{e1}; 3; t1), (a, c; l_2, l_1, l_{e1}; 1; t2), (b, d; l_1, l_2, l_{e2}; 2; t3), (b, e; l_1, l_1, l_{e1}; 1; t4), (c, b; l_1, l_1, l_{e2}; 2; t5)* 和 *(e, c; l_1, l_1, l_{e1}; 1; t6)*。此时我们设置 d =6, $b = 3, F = 8, r = 2$ 和 $s = 2$ 。

首先，我们计算相应的哈希值，并计算每个顶点的最终候选地址，如图 *11* 左侧所示（为清晰起见，此处省略计算地址候选列表的过程）。根据相应顶点标签的哈希值，我们可以确定每个项应存储在哪个块中。例如，项 (b, e) 、 (c, b) 和 (e, c) 应放置在左上方的存储块中。接下来，我们通过插入每个项来更新草图。对于项 $(a, b;\ l_2, l_1, l_{e1};\ 3;\ t1)$ ，将其放置在左下方存储块中索引对为 *(1,1)*、指纹对为 *(1,4)* 的第 *0/1* 个单元格中。对于项 $(a, c;\ l_2, l_1, l_{e1};\ 1;\ t2)$ ，我们首先根据两个端点的候选地址列表检查左下方存储块中的第 *0/1* 个单元格。请注意，为了更好地说明如何处理单元冲突以及如何搜索地址列表，此处省略孪生单元策略。由于该单元格已被其他项占用（因为指纹不匹配），我们检查

成功将其插入左下方存储块的第0个/2个单元格，并赋予索引对(1，2)（注意，在实际操作中，如果没有冲突，第一次尝试插入时就完全可能成功）。该其他项的插入方式类似，最终结果如图 *11* 右图所示。由于本示例旨在说明如何定位项，因此省略了其中权重变化的操作。

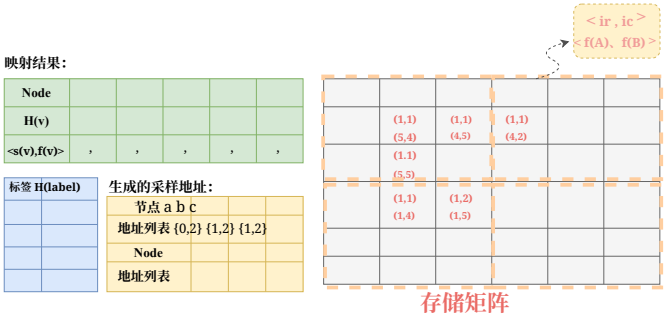


图11。LSketch 的第一个示例

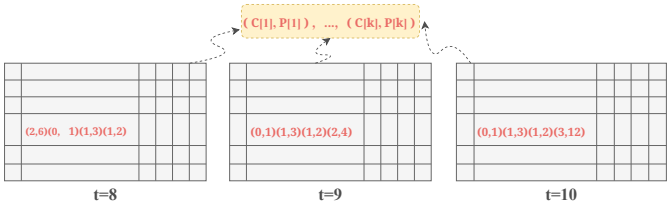


图12。LSketch 的第二个示例

示例 2。 接下来，我们重点说明 *LSketch* 如何在滑动窗口模型下处理权重变化。

我们设定 $W = 8$ 和 $W_s = 2$，因此每个单元格维护两个长度为 *4* 的计数器列表。此外，我们给定预定义的素数列表为 $P_r = [2, 3]$ ，其中 $c = 2$。假设我们已将顶点对 (a, b) 对应的项目定位到左下存储块的第 0 位/1 位（与前一个示例中的设置相同），时间8 的当前状态如图*12*所示。

现在我们继续处理顶点对 *(a, b)* 的项目。在时间 *9*，假设项目 *(a, b; l_2, l_1, l_{e1}; 2; 9)* 到达。由于第一个子窗口过期，我们将其滑动并启动一个新的子窗口。根据哈希结果，其中边标签 l_{e1} 对应素数$P_r[0] = 2$，我们将新的子窗口从 *(0, 1)* 更新为 $(0+1+1, 1*2*2)$。在时间 *10*，假设项目 *(a, b; l_2, l_1, l_{e2}; 1; 10)* 到达，因此根据素数表示 $P_r[1] = 3$，最新的子窗口更新为 $(2+1, 4*3)$。

3.5 进一步改进

到目前为止，LSketch 能够很好地处理在现实场景中高速更新的异构图流。然而，我们观察到，在某些情况下，顶点的标签分布并不均匀。也就是说，图流中的大多数顶点具有相同的标签。如果继续采用之前的存储块划分方法，就会导致某个块严重拥塞，而其他块的存储却较为稀疏。因此，针对这种极端情况，我们提出了另一种称为“偏斜分块”的策略，以改进之前均匀分块的缺陷。

偏斜分块。为了将顶点标签高效编码到草图中并支持后续查询，我们仍采用分块的思路。然而，每个矩阵块的大小

应根据需求设定。简而言之，我们不再对矩阵进行均匀划分，而是根据预定义的场景设置标签的比例分布。

给定一个包含两种顶点标签的数据集，图13 展示了在均匀分块和偏斜分块策略下插入一条边时候选项列表的示例，其中在偏斜分块策略下，两种顶点标签的比例设为 3:7。可以看出，当面对极度不平衡的数据集时，偏斜分块策略能够有效提高将边插入矩阵的成功率。为了获取图流的分布特征，我们可以收集一小段时间内的数据，并根据数据的分布特征选择均匀分块或偏斜分块。

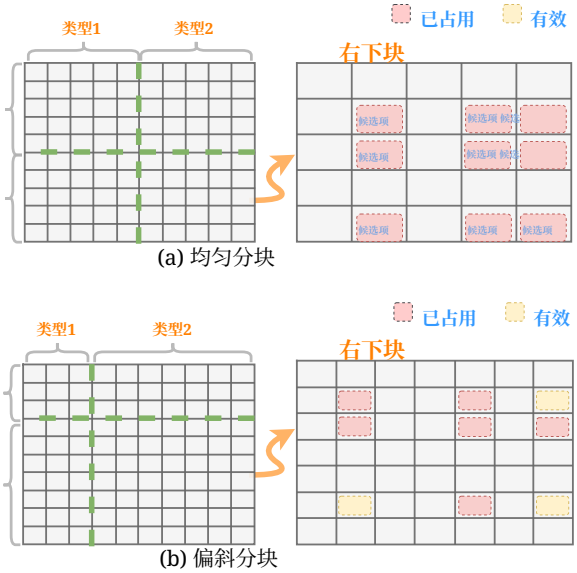


图 13. 展示均匀分块与偏斜分块的比较

3.6 存储与时间复杂度分析

在实际应用中，我们通过指针实现存储矩阵，因此只有现有的边会占用存储空间。结合附加池，LSketch 会将多条边（这些边的端点相同，但边类型可能不同，且到达时间戳可能不同）存储在同一个存储单元中，因此所需的存储空间至多为$k \times |E_s|$，其中 $|E_s|$ 是在滑动窗口内不同边的数量（端点相同的边只计一次）， k 是子窗口的数量。预计这远小于实际边数 $|E|$。因此，LSketch 在存储方面非常高效，尤其适用于输入速率高且重复边较多的图流。在时间开销方面，LSketch 需要 $O(1)$ 时间来定位存储块；将边插入存储块时，我们最多检查 s 个采样单元格，耗时为 $O(s)$ 时间。尽管向附加池插入需要线性时间，但通过我们采用的各种避免插入附加池的技术，实际需要插入其中的概率几乎可以忽略。由于 s 通常是一个较小的常数，插入的时间复杂度为 $O(1)$。

算法 3： GetWeightsInM

输入： 所选矩阵单元 E 及其子选项 $shifter$，边标签 l_e（可选）
输出： w：一个子节点中边的总权重 E、 w_l的片段：具有.....的边的权重 在 E的一个子段中标记 l_e

1 $P_e = P_r[H(l_e)\%c]$//获取素数 如果提供了 l_e 的表示
2 $w = 0$， $w_l = 0$
3 **对** $i \leftarrow 1 \dots k$ **执行**
4 $\quad w = w + E[shifter][i].C$
5 $\quad temp_p = E[shifter][i].P$
6 $\quad$ **当** $temp_p \% P_e = 0$ **时执行**
7 $\quad\quad w_l = w_l + 1$
8 $\quad\quad temp_p = temp_p / P_e$
9 返回 w， w_l

4 LSKETCH 驱动的图查询

由于 LSketch保留了节点之间的连接，因此能够处理所有基于图结构的查询。在本节中，我们将展示其中一些查询，以说明采用均匀分块策略的LSketch如何支持各种查询，包括基于图结构的查询和对时间敏感的查询。采用偏斜分块策略的LSketch 上的查询方法可以轻松扩展，因此为节省篇幅我们将其省略。首先，我们介绍基础算法 GETWEIGHTSINM，它是后续查询的基础。

算法 GETWEIGHTSINM 用于在给定指定存储单元中的特定子段时，计算总权重以及具有特定边标签的权重。实际应用中，我们使用位运算来获取单元格中某个子段的权重。但为方便起见，我们在伪代码中将其视为一个列表。因此，通过遍历 k 个子窗口并依次处理计数器（第 3–8 行），我们可以轻松获得所需的权重。

为便于对本节后续查询进行准确性分析，我们首先讨论一条边 e 发生边碰撞的概率。

定理 1. 一条边发生边碰撞的概率 $\tilde{P}$ 为 $1-e^{-\frac{(q_1+Lq_2+L^2q_3)(|E|-d_v)+(DLh_1+DL^2h_2)d_v}{D^2L^2}}$，其中$|E|$是边的数量， d_v 是顶点 v 的出度（其他参数的含义将在证明中说明，随后还会给出一个示例进行解释。）。

证明。 根据 LSketch 的构造方法，我们知道，边缘冲突只可能发生在初始哈希阶段，因为之后所有的索引对和指纹对都能确保哈希值不同的边不会占用相同的存储空间。如前所述，顶点的哈希范围为 $[0, D)$， 顶点标签的哈希范围为 $[0, L)$。

定位一条边缘需要对顶点标识符和顶点标签进行哈希。对于一条与给定边缘 e 没有共同端点的边缘，只有当起始顶点、终止顶点以及两个顶点标签都与 e 中对应的对象发生碰撞时，它才会与 e 发生碰撞。顶点标签发生碰撞有三种情况：a) 对应的起始顶点标签和终止顶点标签均不同，但它们的哈希值相同。b) 与起始

顶点或终止顶点对应的顶点标签不同，但其哈希值相同。c) 对应的起始顶点和终止顶点的顶点标签相同。因此，碰撞概率可表示为:

$$p_1 = \frac{1}{D^2} \times (\frac{1}{L^2} \times q_1 + \frac{1}{L} \times q_2 + q_3) \tag{5}$$

，其中 q_1、 q_2 和 q_3 分别是上述三种情况的概率，且 $q_1 + q_2 + q_3 = 1$。其他 $|E| - d_v$ 条边均未与 e 冲突的概率为:

$$P_1 = (1 - p_1)^{|E|-d_v} \tag{6}$$

对于与 e共享一个公共端点的 d_v 边，顶点标签与另一个顶点发生碰撞有两种情况。a)非共享顶点的顶点标签不同，而它们的哈希值相同。b) 非共享顶点的顶点标签确实相同。因此，那些边与 e 发生碰撞的概率为:

$$p_2 = \frac{1}{D} \times (\frac{1}{L} \times h_1 + h_2) \tag{7}$$

，其中 h_1 和 h_2 分别是上述两种情况的概率，且 $h_1 + h_2 = 1$。

而 d_v 边均不与 e 发生冲突的概率为:

$$P_2 = (1 - p_2)^{d_v} \tag{8}$$

总而言之， $|E|$ 条边中没有一条与 e 发生碰撞的概率是:

$$\begin{aligned} P =& P_1 \times P_2 = (1 - p_1)^{|E|-d_v} \times (1 - p_2)^{d_v} \\ &= e^{-p_1 \times (|E|-d_v)} \times e^{-p_2 \times d_v} \\ &= e^{-\frac{q_1+Lq_2+L^2q_3}{D^2L^2} \times (|E|-d_v)} \times e^{-\frac{h_1+Lh_2}{DL} \times d_v} \\ &= e^{-\frac{(q_1+Lq_2+L^2q_3)(|E|-d_v)+(DLh_1+DL^2h_2)d_v}{D^2L^2}} \end{aligned} \tag{9}$$

~，一条边遭遇边碰撞的概率为 $1 - P$，即 P 。

□

为便于分析，我们假设节点标签服从均匀分布，这将导致

$$\begin{cases} q_1 = \frac{(l-1)^2}{l^2} \\ q_2 = \frac{2(l-1)}{l^2} \\ q_3 = \frac{1}{l^2} \\ h_1 = \frac{l-1}{l} \\ h_2 = \frac{1}{l} \end{cases} \tag{10}$$

，其中 l 是节点标签的数量。因此,

$$P = e^{-(\frac{L+l-1}{DLl})^2(|E|-d_v)-\frac{L+l-1}{DLl}d_v} \tag{11}$$

在 LSketch中，假设我们有 $D = d \times F$ 和 $L = t \times F$，其中 d 是矩阵的宽度， F 是指纹的范围。假设我们设置 $F = 256$ （使用8 位指纹）和 $d = 1000$。当处理一个具有 $|E| = 5 \times 10^5$ 和 $t = 2$ 的数据集时，我们可以通过 $d_v = 200$ 得到概率 $P = 0.9996$ 。换句话说，一条边遇到边碰撞的概率仅为 0.0004，这很好地满足了实际应用中的准确性要求。

4.1 顶点查询

顶点查询包括顶点 v的出边权重查询和入边权重查询，用于估计其出权重或入权重（当我们将所有权重设置为 1时，这等价于出度和入度查询）。此外，顶点查询还可以按边标签 l_e进行限制，因此查询结果表示具有某一边标签、由顶点v发送/接收的权重。这里我们仅讨论顶点 v 的出边权重查询，因为入边权重查询与之类似。

算法 4： AggregateVertexQueries

输入： 一个顶点 A、一个顶点标签 l_A、一个边标签 l_e（可选）

输出： w：顶点 A的出权重， w_l：该顶点 A带标签 l_e 的出权重; sum：具有的所有顶点的出权重 标签 l_A, sum_l：带标签的出权重 l_e 具有标签的所有顶点中的 l_A

```
1  w = 0, w_l = 0, sum = 0, sum_l = 0
2  Precompute（A， l_A）
3 对于 i ← 1...r 执行
4      p_1 = (s(A) + l_i(A))%b
5      针对 j ← 1...d 执行
6          针对 shifter ← 1...2 执行
7              如果 存储的索引等于 i 且存储的指纹匹配 f（A）则
8                  w+ = GetWeightsInM(S[m × b + p1][j]， shifter)
9                  w_l+ = GetWeightsInM(S[m × b + p1][j]， shifter， l_e)
10 对于 i ← m × b...m × b + b 执行
11     对于 j ← 1...d 执行
12         对于 shifter ← 1...2 执行
13             sum+ = GetWeightsInM（S[i][j]， shifter)
14             sum_l+ = GetWeightsInM(S[i][j], shifter, l_e)
15 w = w + weight 在附加池中获得
16 w_l = w_l + weight 在附加池中获得
17 sum = sum + weight 在附加池中获得
18 sum_l = sum_l + weight 在附加池中获得
19 返回 w, w_l, sum, sum_l
```

详细算法如算法4所示。对于存储矩阵，我们首先计算顶点 v 的可能行（第 2-4 行），然后遍历所有列。如果索引和指纹匹配成功，我们使用GETWEIGHTSINM获取满足条件的所有单元格的权重（第 5-9 行）。我们还遍历附加池以查找剩余项。回顾图 8 中描述的数据结构——附加池中的元素通过源节点链接，因此我们需要检查源节点列表。如果链表头的哈希值与所查询顶点的哈希值相同，则找到了匹配的源节点，并且应将该列表中的总权重添加到 w。带有标签限制的权重也会被获取并添加到 w_l （第 15-16 行）。

此外，由于我们的草图会将具有相同顶点标签（类型）的顶点放在一起，因此我们可以针对某一类型的顶点进行权重查询。我们使用顶点标签获取存储块的起始行 $m \times b$（第 2 行），因此具有特定标签的顶点的存储位置为 $[m \times b,\ m \times b + b)$。对于这些 b 行，我们遍历所有列，并使用 GETWEIGHTSINM 对权重求和（第 10-14 行）。

类似地，我们遍历附加池以获得最终结果（第 17–18 行）。

根据定理1，对于顶点查询，不带标签限制的出边权重查询的准确率由 $P^{|V|-d_v}$保证，其中 $|V|$ 表示 G 中顶点的数量，而 d_v 是被查询顶点的出度。带标签限制的出边权重查询涉及范围为 $[0, c)$的边标签哈希，因此其准确率由 $P^{|V|-d_v} \times (1-\frac{1}{c})^{l-1}$ 保证，其中 l 是边标签的数量。

对于查询时间，我们需要遍历被查询顶点的所有可能r行，将所有匹配单元格的k子窗口中的权重求和，最后补充附加池中的权重。因此，时间复杂度为$O(d \times r \times k + |AP|)$。

4.2 边查询

给定一条边 e，其端点为 A和 B，对应的顶点标签为 l_A 和 l_B，边查询用于估算 e的边权重。具体而言，边查询可以通过边标签 l_e进行限制，旨在获取端点 A 和 B之间类型为 l_e 的边的权重。与顶点查询类似，LSketch 可以处理顶点具有特定类型的情况，例如查询一个顶点与某一特定类型顶点之间的边，或两种特定类型顶点之间的边。

算法 5： AggregateEdgeQueries

输入： 一条边 $e=(A, B)$ 及其对应的顶点标签 l_A 和 l_B，一条边标签 l_e（可选）
输出： w_{vg}：顶点 A 与标签为 l_B、w_{vg_l}的顶点：标签为 l_e 的权重（顶点 A 与标签为 l_B的顶点之间）

```
1  w_vg = 0, w_vg_l = 0
2  Precompute(A, l_A), Precompute(B, l_B)
3  对 i ← 1...r 执行
4      p_1 = (s(A) + l_i(A))%b
5      对于 j ← m_B × b ... m_B × b + b 执行
6          对于 shifter ← 1...2 执行
7              E = S[m × b + p_1][j]
8              如果 已存储的索引等于 i 且已存储的指纹匹配 f（A）然后
9                  w_vg += GetWeightsInM(E, shifter)
10                 w_vg_l += GetWeightsInM(E, shifter, l_e)
11 w_vg = w_vg + weight 在附加池中获得
12 w_vg_l = w_vg_l + weight 在附加池中获得
13 返回 w_vg, w_vg_l
```

详细算法如算法 5 所示。针对特定边的边查询与插入过程类似，因此我们只展示顶点与特定类型顶点之间的边查询。执行顶点 A 与标签为l_B的顶点之间的边查询时，我们首先遍历候选项列表技术生成的 A可能位置的 r 行。然后，我们计算标签为 l_B的存储块中的起始列，并从 $m_B \times b$ 遍历到 $m_B \times b + b$列。如果索引和指纹都匹配，我们就对权重求和（第 7–9 行）。最后，我们遍历附加池以获得最终结果（第 10–11 行）。

不带标签限制的边查询的准确率由定理 1 中计算出的精确值 P 保证，而带边标签限制的边查询的准确率上界为 $P \times (1-\frac{1}{c})^{l-1}$，其中 l 是边标签的数量，$\bar{c}$ 是它们的哈希范围。此外，边查询的时间复杂度与插入 LSketch 的时间复杂度相同，即为 $O(1)$。

4.3 路径查询

给定两个顶点 A 和 B，路径查询用于确定是否存在一条从 A 到 B的可达路径。具体而言，可以通过指定边标签来限制路径查询，以查找从 A 到 B是否存在特定类型的路径。

基于 LSketch维护的结构信息，我们的算法可以作为黑盒应用于任何现有的路径可达性算法（或任何其他基于结构的图查询）。我们以BFS 为例说明该查询，不带边标签限制的详细算法如算法 6 所示。

算法 6： 路径可达性

输入： 顶点 A 和 B，以及其对应的顶点标签 l_A 和 l_B
输出： A 到 B之间是否存在路径

```
1  q = {A} // 队列
2  checked = {A}
3  当 q 不为空则
4      src = q.头部
5      Precompute(src, l_src), Precompute(B, l_B)
6      对于 i ← 1...r，执行
7          p_1 = (s(src) + l_i(src))%b
8          对于 j ← 1...r，执行
9              p_2 = (s(B) + l_B_j(B))%b
10             E = S[m_src × b + p_1][m_B × b + p_2]
11             对于 shifter ← 1...2 执行
12                 如果存储的索引对和指纹对匹配查询顶点的那些然后
13                     如果 GetWeightsInM(E, shifter) > 0 then
14                         return true
15     将附加池中 src 的所有后继节点添加到 q 和 checked
16     对于 i ← 1...r，执行
17         p_1 = (s(src) + l_i(src))%b
18         对于 j ← 1...d，执行
19             E = S[m_src × b + p_1][j]
20             对于 shifter ← 1...2，执行
21                 如果 存储的索引等于 i 且存储的指纹匹配 f(src) 然后
22                     H(temp) = s(temp) × F + f(temp)
23                     q.push(H^-1(temp))
24                     checked.add(H^-1(temp))
```

我们定义一个队列 q 和一个已检查列表 $checked$ 来实现 BFS。将 A 添加到 q 和 $checked$（第 1–2 行）后，我们开始遍历 q。当 q 为空时，循环停止，并返回 false。如果提前找到可达路径，则返回 true。

在每次循环中，我们首先分别计算所提取的临时起始顶点src和 B可能对应的行/列（第4-5行）。然后，我们依次遍历可能的孪生段，以检查索引对和指纹对是否都匹配（第6-11行）。如果匹配，则我们已经找到了一条从 A 到 B 的可达路径，并且

返回 true（第 12-14 行）；否则，我们将附加池和主矩阵中 src 的所有后继节点添加到q 和 $checked$ （第 15-24 行）。请注意，我们需要分配存储空间来存储每个节点v的初始地址 $s(v)$ 。函数 H^{-1} 接收一个哈希值，并返回其对应的顶点标识符。

无标签限制的路径查询的准确率由 $P^{|V|-d_a}$ 保证，其中 P 在定理 1 中计算， $|V|$ 和 d_a 分别表示 G 中顶点的总数和平均出度。带边标签限制的路径查询的准确率由 $P^{|V|-d_a} \times (1-\frac{1}{c})^{l-1}$ 界定，其中 l 是边标签的数量， c 是这些标签的哈希范围。此外，查询的时间复杂度由所采用的路径可达性算法决定。

4.4 近似子图查询

LSketch 保留了图的连通性，因此边的聚合结果实际上可以表示图的连通性。因此，为了提高查询效率，我们使用近似子图查询，以一定的准确率表示子图查询结果。给定一个由一组边 $G_q = \{e_1, ..., e_n\}$ 组成的子图，子图查询会估算与之匹配的子图模式数量。在我们的模式中，一条边 e_i 表示为 $(x_i, l_{x_i}, y_i, l_{y_i})$，对应于它的两个顶点及相关的顶点标签。简而言之，我们只需对每条边执行一次边查询以获取其匹配权重，然后取所有权重中的最小值作为匹配数量。详细算法见算法 6。此外，子图查询可以轻松扩展到带边标签约束的情况。

算法 7： 近似子图查询

输入： 一个子图 G_q 由一组边组成 $\{e_1, ..., e_v\}$, where $e_i = (x_i, l_{x_i}, y_i, l_{y_i})$
输出： 查询的匹配数量 子图 G_q

```
1 res = MAX
2 对于 i ← 1...v 执行
3     w_i = AggregateEdgeQueries(x_i, l_{x_i}, y_i, l_{y_i})
4     如果 w_i = 0，则
5         返回 0
6     如果 w_i < res，则
7         res = w_i
8 返回 res
```

近似子图查询的准确率与边查询相似，对于不带/带边标签限制的情况，分别由 P^v 和 $P^v \times (1-\frac{1}{c})^{l-1}$ 保证，其中 l 表示边标签的数量， c 表示它们的哈希范围， v 表示查询子图的大小。此外，子图查询的时间复杂度为 $O(v)$。

5 实验评估

5.1 实验设置

数据集。我们使用了表 2 中展示的四个真实世界数据集。

表 2 数据集统计信息

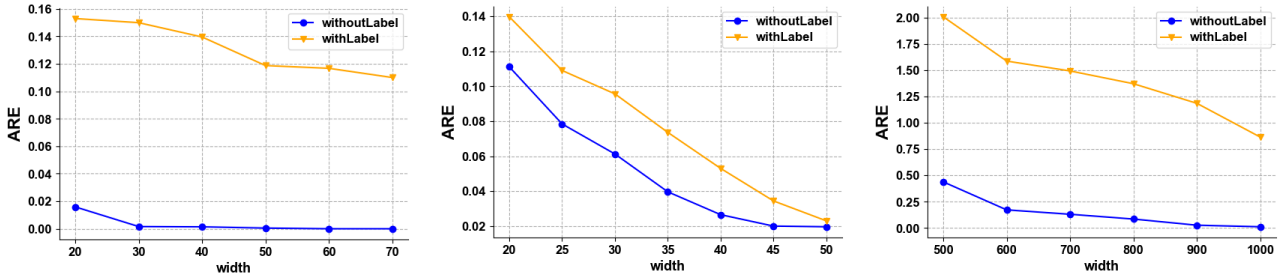
数据集	Phone	Road	Enron	com-FS
边数	60,765	870,757	2,064,442	1,806,067,135
顶点标签数量	2	–	11	20
边标签数量	4	6	35455	100
窗口大小	1 周	1 天	1 周	1 天
子窗口大小	1 h	5 分钟	1 h	10 分钟

1)Phone数据：MIT Reality 数据集[2]。该项目记录了 2004 年至2005年间 94 名受试者的通信、接近、位置和活动信息。图流中的每一项都包括呼叫 ID、接收 ID、呼叫时间、呼叫类型和呼叫持续时间。我们对图流进行预处理，将顶点分为两组：研究对象和其他对象，并根据呼叫的类型和持续时间分配九种边标签。该数据集共有 60,765 条边。**2)Road数据**：香港道路的实时交通速度地图[3]。我们从下载的 XML 文件中提取每条观测记录中道路的两个端点ID、区域、道路类型、道路饱和度和交通速度，并根据时间戳将其建模为图流。其中，我们不考虑顶点标签，仅根据道路类型和饱和度分配六种边标签来描述道路状况。该数据集共有 870,757 条边。**3) Email 网络数据**：一个包含 2,064,442 条边的电子邮件交互数据集[4]。我们使用员工的职位作为顶点标签，共得到十一种类型。每项数据的边标签是电子邮件的主题，该主题经过哈希处理并以唯一编号表示。 **4) Social 网络数据**：来自 SNAP 的Friendster 社交网络 [41]。该数据集是一个包含 1,806,067,135 条边的静态网络。我们生成一个大小为20 的顶点标签集合和一个大小为100 的边标签集合，并为每项数据随机分配标签和时间戳。该数据集的最终大小为32.9 GB。我们使用这个半合成数据集来检验所提出方法的可扩展性。

竞争方法。 我们选择 GSS [33]和 LGS [31]作为竞争方法。尽管 GSS 无法处理异构图，也无法区分在不同时间戳到达的项目，但它是与我们的方法最相关且准确率较高的最新方法。LGS 能够处理标签和时间戳，是目前适用于异构图流的最先进方法。我们通过实验评估 LSketch 相较于 GSS 和 LGS 的效率与准确率。其中，在无标签限制的查询中，我们将 LSketch 与 GSS 和 LGS 进行比较；在有标签限制的查询中，则仅与 LGS 进行比较。

参数设置。为进行公平比较，我们将所有数据集的项目权重设置为 1，以便 GSS 和 LGS 处理这些图流，并且在 GSS 中不考虑标签和时间戳。对于 LGS，我们使用 6 个图草图副本来提高其准确率。对于这三种方法， d设置为相同值，因此与 GSS 和 LSketch 相比，LGS 将使用六倍的存储空间。具体而言，我们将在后文讨论如何为不同数据集设置 d 的大小。此外， b 的大小（对于 LGS 和 LSketch）是

2 20050701 3.https://crawdad.org/mit/reality/ . https://data.gov.hk/sc-data/dataset/hk-td-sm 1-交通-速度-地图 4. http://www.ahschulz.de/enron-email-data



(a) Phone (b) Road (c) Enron

图 14. 矩阵的顶点查询随 d 变化

根据 d 和数据集中顶点标签的数量进行设置，指纹大小范围为 8–16 位。我们将候选项列表的长度设为 16，采样列表的长度 s 也设为 16。根据图流的实际应用及其总体时间跨度，各数据集的窗口大小和子窗口大小如表 2 所示。**指标**。我们使用**平均相对误差（ARE）**来描述顶点查询、边查询和子图查询的准确率。将查询的估计结果和真实结果分别记为 res_s 和 res，相对误差计算为 $\frac{res_s - res}{res}$，因为由于可能发生哈希冲突，res_s 不会小于 res。对于每种查询，我们随机选择查询集，并通过 500 次重复运行得到 ARE。对于路径查询，仅当真实结果为 false 而草图返回 true 时才会出现误差。因此，我们使用**准确率**（误报数/查询数）来衡量性能。

设置。我们在一台配备 16GB 内存、Intel Core i7 处理器（频率为 3.40GHz）的台式机上进行了所有实验。所有算法，包括 LSketch、GSS 和 LGS，均使用 C++ 实现。

5.2 矩阵宽度评估 d

本节将通过展示随 d 变化的评估结果，讨论如何设置最重要的参数 d——矩阵宽度。确定 d 后，我们只需根据数据集中的顶点标签数量计算块的大小（使用均匀分块策略时）。时间相关参数，例如子窗口和窗口大小，则根据实际应用需求进行设置。对于其他参数，我们遵循 GSS 的建议。

一般来说，矩阵是存储边的主要部分，其容量为 $d * d * 2$。因此，该容量应与数据集中不同边的数量相当。以 Phone数据集为例，其中有 4952 条不同的边，因此最合适的 d 应为 50 左右。然而，由于我们的方法涉及由顶点标签编码导致的块划分，通常需要设置更大的 d，以获得更高的准确率。按照上述指导，我们设置不同的 d 来生成草图，并使用有/无边标签约束的顶点查询结果衡量其性能，如图 14(a) 所示。请注意，为了更清楚地展示草图的性能差异，我们设置了较短的指纹长度。可以看到，随着增长。当 d 增大时，查询的平均相对误差会降低。当 d 达到 60 时，不再出现错误。

同样，计算得出 Road 和 Enron 推荐的 d 分别为 40 和 600。实验结果如图 14(b) 和图 14(c) 所示。平均相对误差的变化趋势与我们的推断一致，推荐宽度附近的误差在可接受范围内。对于 GSS 和 LGS，作者也分别在论文中讨论了矩阵宽度 d，其实验结果与我们的结论一致。对于所有草图，查询的平均相对误差会随着矩阵宽度的增加而降低，并最终趋于饱和。

在实践中，对于支持流处理的场景，可根据一个窗口内边的到达速率设置 d。在后续实验中，我们使用每个数据集推荐的 d 来构建草图。

5.3 系统吞吐量

本节通过考察草图的系统吞吐量，评估其可扩展性。由于 GSS 无法处理时间戳，我们使用不带滑动窗口的 LGS 和 LSketch 展示整体对比结果。表 3 显示了每种方法插入一条数据所需的平均时间，以及在每个数据集上的总插入时间。可以看出，在前三个数据集上，所有方法的平均插入时间都在 μs 级别。其中，LSketch 保留的信息远多于 GSS，且准确率远高于 LGS（见第 5.4 节），因此加载时间可能会稍长。在超大型数据集上，三种方法完成一次插入所需的平均时间都在 ms 级别。这是因为即使附加池中只存储了约 1% 的边，由于邻接表效率低下，插入和更新这些边仍然代价高昂。

此外，为了更详细地比较 LSketch 和LGS，我们还在采用滑动窗口的Phone 和 Road数据集上进行了相同的实验。表 4中的结果表明，与LGS 相比，LSketch 在时间效率方面更适合处理异构图流。LSketch 的平均插入时间比 LGS 小一个数量级。

我们的方法是一种通用存储结构，支持广泛的图查询。因此，与针对特定图查询进行优化的结构相比，它具有更强大的功能。尤其是，引入滑动窗口后，我们能够支持各种

表 3 平均插入时间（微秒/边缘）和总插入时间（毫秒）

Time	方法	数据集			
		Phone	Road	Enron	com-FS
平均值（微秒/边缘）	GSS	1.22	1.01	2.77	5.89 毫秒
	LGS	1.56	1.22	11.43	6.77 毫秒
	LSketch	2.73	1.83	7.27	8.82 毫秒
总计 (ms)	GSS	74.3	875.3	5724.5	3210000 秒
	LGS	94.9	1062.6	23593.5	3670000 秒
	LSketch	166.1	1591.2	14998.5	4780000 s

表 4 使用滑动窗口时的插入时间（每条边/总体，毫秒）

Time	方法	数据集	
		Phone	Road
平均值（毫秒/边缘）	LGS	1.36	0.03
	LSketch	0.43	2.45 微秒
总计 (ms)	LGS	82739.40	23468.90
	LSketch	26429.70	2137.80

时间区域约束下的结构化查询类型，这具有重要的实际意义。

5.4 查询回答评估

1）时间效率。草图在支持查询方面的时间效率显著优于原始数据查询，快了数个数量级。表5展示了顶点查询和边查询的结果。查询过程中，LSketch 需要进一步处理两个顶点标签，因此时间消耗略高于 GSS。不过，这种差异几乎可以忽略不计，因为使用草图时，所有查询都能在 μs 级别内完成。其他类型查询的时间消耗也大致相同，为清晰起见，此处不再赘述。

表5 顶点查询和边查询的响应时间

查询	方法	数据集		
		Phone	Road	Enron
顶点查询	原始数据	73.90 毫秒	934.58 毫秒	8388.80 ms
	GSS	4.18 微秒	5.47 微秒	72.87 微秒
	LGS	2.21 微秒	3.98 微秒	68.88 微秒
	LSketch	4.67 微秒	5.13 微秒	76.87 微秒
Edge 查询	原始数据	332.59 毫秒	3611.56 毫秒	32208.23 毫秒
	GSS	1.07 微秒	1.19 微秒	6.30 微秒
	LGS	1.60 微秒	1.38 微秒	6.43 微秒
	LSketch	3.04 微秒	1.05 微秒	10.77 微秒

2）忽略时间戳时查询的准确率。

顶点查询。给定查询的顶点，我们对所有数据集的三个草图执行出度查询，结果如图15(a)所示。可以看出，我们的该方法在所有数据集上的表现都远优于 LGS。由于我们在不增加额外存储空间的情况下保留了图流的更多信息，因此在Phone数据集上，其准确率略低于GSS。在其他数据集上，LSketch 达到了与GSS 相同的准确率，这表明我们的方法在顶点标签保留方面既高效又准确。

我们还计算了所有数据集的错误率保证。Phone数据集的概率 P 为0.999352，顶点查询的错误率保证低于0.219，这远高于实际运行值。在其他数据集上，结果类似。边不发生碰撞的概率约为0.9，计算得到的错误率保证非常低。

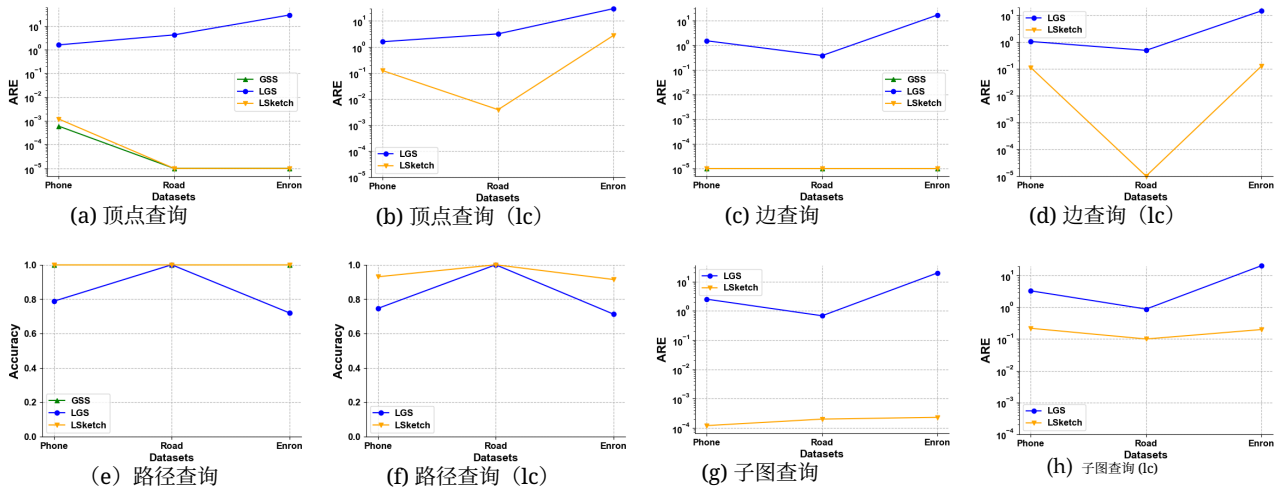
此外，为了展示LSketch在保留异构信息方面的优势，我们执行了带有边标签限制的顶点查询，结果如图15(b)所示。我们只展示LSketch和LGS的结果，因为GSS不保留标签信息，因此无法支持标签约束查询。性能良好，这是因为我们为所有数据集设置了更大的 c ，其中 c 是预定义素数列表的长度。c越大，边标签碰撞的概率就越小，从而降低错误率。可以看出，在相同参数设置下，LSketch的表现明显优于LGS。此外，LSketch无需使用多个草图来提高查询准确率，这展示了LSketch在信息保留、节省空间和查询效率方面的能力。

边查询。 接下来，我们展示三种草图上的边查询结果。从图15(c)中可以看出，三种方法在边查询方面都表现良好。特别是，LSketch 和 GSS 在所有数据集上都几乎达到100% 的准确率。LGS 不具备区分存储在同一位置的不同项的能力，因此其错误率略高。错误率保证等于 $1-P$，即一条边与其他边发生冲突的概率，并且在所有数据集上都可维持在0。1 以下。带边标签限制的边查询结果类似，如图15(d)所示。

路径查询。现在，我们通过图15(e)和15(f)评估 LSketch 对可达性查询的支持性能。由于 Roaddataset 是连通图，其准确率始终为 1。对于其他数据集，LSketch 的准确率与 GSS 相当，且远高于 LGS。正如上文所分析，路径查询的准确率保证与顶点查询相同。

近似子图查询。由于 GSS 的基本版本不支持子图查询，我们只将结果与 LGS 进行比较。我们实现的近似匹配查询是重复执行边查询，因此子图查询的结果与边查询的结果非常相似，如图15(g)和15(h)所示。

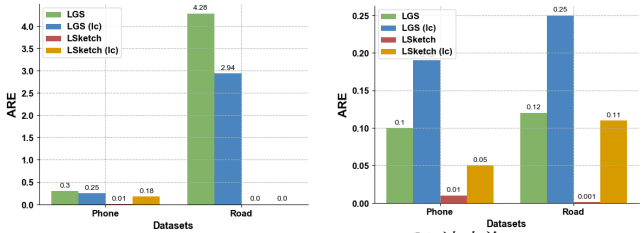
3) 引入滑动窗口后的查询准确率。图16展示了 LSketch 和 LGS 的顶点查询与边查询结果，其中“lc”表示查询受标签约束。可以看出，引入滑动窗口后，LSketch 的平均相对误差进一步降低，远优于 LGS。由于篇幅限制，其他结果不再赘述。



(a) 顶点查询 (b) 顶点查询（lc） (c) 边查询 (d) 边查询（lc）

（e）路径查询 (f) 路径查询（lc） (g) 子图查询 (h) 子图查询 (lc)

图 15. 三种不含滑动窗口的草图的查询性能



(a) 顶点查询 (b) 边查询

图 16. 三种采用滑动窗口的草图的性能

6 结论

本文提出了一种用于图流摘要的新型结构 LSketch。该结构采用滑动窗口机制，适用于异构图流，更符合实际应用的需求。LS-ketch 仅需亚线性存储空间和 $O(1)$ 更新开销。它保留了图流的底层结构和标签信息，使其能够支持多种基于结构的查询。实验结果表明，与 LGS 相比，LSketch 的查询准确率和响应速度均有显著提升。此外，与适用于同构图流的当前最先进方法 GSS 相比，我们提出的方法以略高的时间开销进一步保留了标签和时间戳的存储。上述实验结果和理论分析充分证明了 LSketch 的优越性。

致谢

本研究部分由国家自然科学基金项目资助，项目编号为 62172237、61772289、U1836109、U1936206、U1936105 和 62077031；同时获得美国国家科学基金会项目 IIS-1633271 以及新英格兰交通联盟项目 20-2 的资助。

参考文献

[1] C. C. Aggarwal，“社交网络数据分析导论”，载于 社交网络数据分析。Springer，2011 年，第 1–15 页。

[2] C. C. Aggarwal, H. Wang 等人，《图数据管理与挖掘》。Springer，2010年，第40卷。[3] I. Tsalouchidou、F. Bonchi、G. D. F. Morales 和 R. Baeza-Yates，“可扩展的动态图摘要”，《IEEE知识与数据工程汇刊》，第32卷，第2期，第360–373页，2018年。[4] G. Cormode 和 S. Muthukrishnan，“多重图流的空间高效挖掘”，载于《第二十四届ACM SIGMOD-SIGACT-SIGART数据库系统原理研讨会论文集》，2005年，第271–282页。[5] J. Feigenbaum、S. Kannan、A. McGregor、S. Suri 和 J. Zhang，“半流式模型中的图问题”，《理论计算机科学》，第348卷，第2–3期，第207–216页，2005年。[6] S. Ganguly 和 B. Saha，“估算流式图中的路径聚合量”，载于《国际算法与计算研讨会》。Springer，2006年，第163–172页。[7] A. D. Sarma、S. Gollapudi 和 R. Panigrahy，“估算图流中的PageRank”，《ACM杂志（JACM）》，第58卷，第3期，第1–19页，2011年。[8] C. C. Aggarwal、Y. Li、P. S. Yu 和 R. Jin，“图流中的稠密模式挖掘”，《VLDB学报》，第3卷，第1–2期，第975–984页，2010年。[9] C. C. Aggarwal，《数据流：模型与算法》。Springer，2007年，第31卷。[10] N. Alon、Y. Matias 和 M. Szegedy，“近似频率矩的空间复杂度”，《计算机与系统科学杂志》，第58卷，第1期，第137–147页，1999年。[11] L. S. Buriol、G. Frahling、S. Leonardi、A. Marchetti-Spaccamela 和 C. Sohler，“数据流中的三角形计数”，载于《第二十五届ACM SIGMOD-SIGACT-SIGART数据库系统原理研讨会论文集》，2006年，第253–262页。[12] Z. Bar-Yossef、R. Kumar 和 D. Sivakumar，“流式算法中的归约及其在图中三角形计数上的应用”，载于《SODA》，第2卷，2002年，第623–632页。[13] C. Estan 和 G. Varghese，“流量测量与记账的新方向：关注大象，忽略老鼠”，《ACM计算机系统汇刊（TOCS）》，第21卷，第3期，第270–313页，2003年。[14] P. Roy、A. Khan 和 G. Alonso，“增强型草图：更快、更准确的流处理”，载于《2016年国际数据管理会议论文集》，2016年，第1449–1463页。[15] T. Yang、Y. Zhou、H. Jin、S. Chen 和 X. Li，“金字塔草图：用于数据流频率估算的草图框架”，《VLDB学报》，第10卷，1453，2017年11月，第1442页起。[16] J. H. Chang 和 W. S. Lee，“在线数据流中自适应地查找近期频繁项集”，载于《第九届ACM SIGKDD知识发现与数据挖掘国际会议论文集》，2003年，第487–492页。[17] C. Jin、K. Yi、L. Chen、J. X. Yu 和 X. Lin，“不确定数据流上的滑动窗口Top-k查询”，《VLDB学报》，第1卷，第1期，第301–312页，2008年。

[18] K. Mirylenka、G. Cormode、T. Palpanas 和 D. Srivastava，“条件重频项：检测数据流中有意义的相关性”，《VLDB期刊》，第24卷，第3期，第395–414页，2015年。[19] R. Ben-Basat、G. Einziger、R. Friedman 和 Y. Kassner，“数据流和滑动窗口中的重频项”，载于《IEEE INFOCOM 2016——第35届IEEE国际计算机通信会议》。IEEE，2016年，第1–9页。[20] G. S. Manku 和 R. Motwani，“数据流中的近似频率计数”，载于《VLDB’02：第二十八届超大型数据库国际会议论文集》。Elsevier，2002年，第346–357页。[21] Y. Zhou、Y. Zhang、C. Ma、S. Chen 和 O. O. Odegbile，“用于网络流量测量的广义草图族”，《ACM计算系统测量与分析汇刊》，第3卷，第3期，第1–34页，2019年。[22] A. Pavan、K. Tangwongsan、S. Tirthapura 和 K.-L. Wu，“图流中的三角形计数与采样”，《VLDB学报》，第6卷，第14期，第1870–1881页，2013年。[23] K. Shin，“WRS：用于实时图流中精确三角形计数的候诊室采样”，载于《2017年IEEE国际数据挖掘会议（ICDM）》。IEEE，2017年，第1087–1092页。[24] A. McGregor，“图流算法：综述”，《ACM SIGMOD记录》，第43卷，第1期，第9–20页，2014年。[25] M. Charikar、K. Chen 和 M. Farach-Colton，“在数据流中查找频繁项”，《理论计算机科学》，第312卷，第1期，第3–15页，2004年。[26] G. Cormode 和 S. Muthukrishnan，“改进的数据流摘要：计数最小草图及其应用”，《算法杂志》，第55卷，第58–75页，2005年。[27] P. Zhao、C. Aggarwal 和 M. Wang，“gSketch：图流中的查询估算”，《VLDB论文集》，第5卷，204，2011年，第193页起。[28] A. Shrivastava、A. C. Konig 和 M. Bilenko，“用于数据流摘要的时间自适应草图（ada-sketches）”，载于《2016年国际数据管理会议论文集》，2016年，第1417–1432页。[29] N. Tang、Q. Chen 和 P. Mitra，“图流摘要：从大爆炸到大收缩”，《2016年国际数据管理会议论文集》，2016年。[30] A. Khan 和 C. Aggarwal，“支持查询的图流压缩”，载于《2016年IEEE/ACM社交网络分析与挖掘进展国际会议（ASONAM）》。IEEE，2016年，第130–137页。[31] C. Song、T. Ge、Y. Ge、H. Zhang 和 X. Yuan，“带标签图草图：跟上实时图流”，《信息科学》，第503卷，第469–492页，2019年。[32] X. Gou、L. Zou、C. Zhao 和 T. Yang，“快速准确的图流摘要”，《2019年IEEE第35届数据工程国际会议（ICDE）》，第1118–1129页，2019年。[33] X. Gou、L. Zou、C. Zhao 和 T. Yang，“图流草图：快速准确地概括图流”，《IEEE知识与数据工程汇刊》，2022年。[34] R. Y. Hung 和 H.-F. Ting，“加权数据流滑动窗口中的重频项查找”，载于《拉丁美洲理论信息学研讨会》。Springer，2008年，第699–710页。[35] O. Papapetrou、M. Garofalakis 和 A. Deligiannakis，“分布式滑动窗口数据流的基于草图的查询”，《VLDB学报》，第5卷，第10期，2012年。[36] M. S. Hassan、B. Ribeiro 和 W. G. Aref，“SBG-Sketch：用于带标签图流摘要的自平衡草图”，载于《第三十届科学与统计数据库管理国际会议论文集》，2018年，第1–12页。[37] P. L’ecuyer，“不同规模线性同余生成器表及其良好的格结构”，《计算数学》，第68卷，第225期，第249–260页，1999年。[38] M. S. Crouch、A. McGregor 和 D. Stubbs，“滑动窗口模型中的动态图”，载于《欧洲算法研讨会》。Springer，2013年，第337–348页。[39] M. Datar、A. Gionis、P. Indyk 和 R. Motwani，“维护滑动窗口上的流统计量”，《SIAM计算杂志》，第31卷，第6期，第1794–1813页，2002年。[40] R. Kumar、T. Calders、A. Gionis 和 N. Tatti，“维护交互网络滑动窗口中的邻域概况”，载于《欧洲机器学习与数据库知识发现联合会议》。Springer，2015年，第719–735页。[41] J. Leskovec 和 A. Krevl，“SNAP数据集：斯坦福大型网络数据集集合”，http://snap.stanford.edu/data，2014年6月。



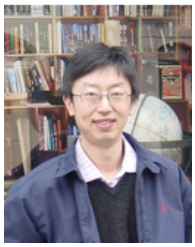
Yiling Zeng 于2020年获得中国南开大学学士学位。目前，她是南开大学硕士研究生。她的研究兴趣包括图摘要以及图流上的图草图。



Chunyao Song 是南开大学计算机科学系副教授。她的研究兴趣包括图数据处理与分析，以及流数据处理与分析。



Yuhan Li 于2020年获得中国东北林业大学学士学位。目前，他是南开大学硕士研究生。他的研究兴趣包括知识图谱、实体链接和数据挖掘。



Tingjian Ge 是马萨诸塞大学洛厄尔分校计算机科学系教授。他的研究兴趣包括数据流、图和图流、噪声与不确定数据、生物医学数据分析，以及数据安全与隐私。