Newton sketches: Estimating Node Intimacy in Dynamic Graphs Using Newton's Law of Cooling

Qizhi Chen†, Ke Wang‡, Aoran Li†, Tong Yang†, Bin Cui†

†*National Key Laboratory for Multimedia Information Processing, School of Computer Science, Peking University, Beijing, China*

‡*Yale University, New Haven, CT, USA*

Abstract—**Dynamic graphs are gaining importance in many real-world applications based on different graph queries. Due to the large volume and high dynamicity, people resort to compute approximations to answer graph queries. However, previous work primarily evaluates the relationship between nodes based on frequency, which is not sufficient in many cases. We observe that this relationship varying process is highly similar to the water cooling process in nature. Based on the observation, we formulate a new concept Intimacy with Newton's law of cooling, to illustrate the relationship between nodes. Currently, there is no prior algorithm tailored for Intimacy estimation. Because Intimacy varies in every time unit, the main challenge lies in how to record and update the Intimacy efficiently. In this paper, we propose a novel technique named Newton-Observe to address this challenge. The key idea of Newton-Observe is that we only decay the Intimacy when we observe/query it. Based on Newton-Observe, we develop a series of Newton sketches to answer three fundamental tasks of Intimacy in dynamic graphs. We theoretically prove that the Newton sketch can estimate the Intimacy within an additive constant error to the real Intimacy. Our experiments on real-world datasets and synthetic datasets show that Newton-Observe outperform the strawman solution by up to $570\times$ smaller ARE and improve the throughput by up to $1.62\times$. All source codes are open sourced at Github anonymously.**

I. Introduction

A. Background and Motivations

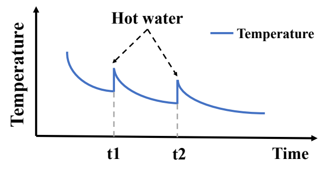
Dynamic graph is omnipresent in social network, recommendation systems, network traffic and so on, representing large-scale entities and their relationship. More and more applications are implemented based on graph queries, such as community detection [8], [16], [37], personalized recommendations [21], network traffic monitoring [12], [20], and so on [6], [24], [25], [29], [32], [33]. In the big data era, dynamic graphs not only grow in scale but also in dynamicity, making it both time and memory consuming to answer these graph queries. A list of pioneering algorithms [19], [26], [34], [45] have been proposed to compute approximations for the challenging dynamic graph queries, such as node query, edge query [19], [34], triangle counting [18] and so on. In this work, we focus on edge query, i.e., measuring the relationship between nodes.

Previous work primarily evaluates the relationship between nodes based on frequency[1], which is not sufficient in many cases. In a social network, it's likely that people are engaged in different communities and have different friends in different life stages. The relationship between people cannot be simply profiled by their interaction frequency, without considering their interaction recency. For example, Alice once had an old friend Bob but lost interaction ten years ago. Now She just makes a new friend Carol and communicates a lot. Though the frequency of interaction with Bob may be much larger than Carol, her current relationship with Carol is much stronger than her relationship with Bob. In a recommendation system, the interest of people may change from time to time. For example, in an E-commerce platform, a new couple may search a lot about baby products. However, when their babies grow up, we should no longer recommend baby products. Recommendation system algorithms inherently require extensive historical data, yet temporal information must also be given significant attention. In network traffic, the network traffic graphs are rapidly and constantly changing. The elephant flows may change from edges to edges, which also can not be simply profiled by frequency. Recent anomalies may indicate the presence of abnormal interactions or information leaks.

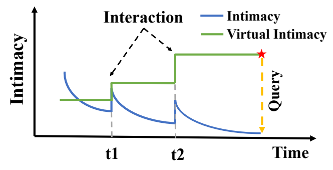
In order to monitor the prompt relationship between nodes, we need to focus more on recent interactions, while older edges are considered of less importance, as the community or network topology formed by them are out-of-date. It remains a problem how to quantify the relationship. One straightforward solution is to adopt sliding windows algorithms[2], which is widely used in streaming graph algorithms and systems [17], [31]. Sliding windows have three major flaws, which our subsequent definition avoids. First, elements outside the window are completely forgotten, which is unrealistic in the real world. Second, it is difficult to distinguish the order of elements within the window; as mentioned earlier, frequency information is not always sufficient. Third, implementation is challenging; due to the dynamic nature of the streaming graph and the inherent complexity of the graph query process, running a sliding window model on a dynamic graph is also computationally ineffective. Another typical metric related to time, namely persistency [43], cannot solve this problem

[1]Frequency in this paper refers to the number of interactions between two nodes

[2]The sliding-window algorithms focus on the data in the most recent T seconds.



(a) Temperature variation



(b) Intimacy variation

Fig. 1: Temperature/Intimacy variation

either. Persistency only counts the number of periods where an element appears but has no sense of recency.

In this paper, we adopt a new way to address this problem. By observing some important regularities of the relationship varying process, we define a new metric, namely Intimacy, to profile this process. On one hand, intimacy should be time-sensitive; past interactions and the most recent ones are not equal, with newer interactions having a greater impact on intimacy. Considering continuous time, a characteristic reflected in intimacy is that it decays over time. The relationship between nodes will fade away with time if there is no more interaction. On the other hand, the relationship between nodes should strengthen upon new interactions. Intimacy should accurately profile both the decay and strengthen process. Our **key observation** is that we find that *the process is highly similar to the water cooling process in nature*: A cup of hot water cools down naturally if there is no hot water coming; Every time we pour some hot water into the cup, the temperature rises.

Based on our above observation, our first contribution is to define Intimacy according to the Newton's law of cooling [3], which the water temperature variation follows. Figure 1 illustrates the variation of temperature and Intimacy, which follows the same trend. Besides the fundamental properties, there are also many detailed similarities between the water cooling process and the decay process of relationship (Intimacy) as shown in Figure 1. Both temperature/Intimacy decrease fast when the temperature/Intimacy is high and decrease slowly when the temperature/Intimacy is low. The temperature of the object will fall to zero eventually if there is no hot water coming in any more. Friends will finally become strangers if there is no interaction any more.

Our second contribution is to address three fundamental tasks in graph queries about Intimacy: edge query (e.g., relationship measurement), global top-k query (e.g., network traffic monitoring), and local top-k query (e.g., personalized recommendations). Edge query is to query the Intimacy of an edge. Global top-k query is to find out the top-k edges with highest Intimacy on the whole graph. Local top-k query is to find out the top-k edges with highest Intimacy incident on a certain node. There are two primary requirements to design a new data structure to answer these three tasks: 1) sub-linear memory cost to fit into the limited-size fast memory [39], [44]; 2) constant update time [36] to match the fast change of dynamic graphs.

B. Our Solution

To address the above three kinds of fundamental tasks, we propose a series of Newton sketches: CM version, CU version, and Space-Saving version. The key challenge of Newton sketches is how to record and update the Intimacy, which is continuously changing with time. The naive solution is to update the Intimacy of all edges in every time unit, where the computation cost is intolerable.

Our first technique is called **Decay operation Per Update (DPU)**, which significantly reduces the computation cost of the naive solution. DPU performs decay operation per update rather than per time unit. DPU records the Intimacy and the most recent timestamp. When a new interaction arrives, DPU only decays the corresponding Intimacy according to the interval time, and updates the timestamp. The DPU solution seems effective but there are imperceptible shortcomings both in memory cost and computation cost. For the computation cost, because we only update its corresponding edge timestamp for every interaction, different edges have different timestamps. When comparing the Intimacy between two edges, it cannot directly compare the values we recorded. Instead, it needs to first align the Intimacy to the same timestamp, and then compare, which incurs extra complexity. It causes extra costs in top-k query, where we need to maintain a heap and perform massive comparisons. For the memory cost, it records an auxiliary data, the most recent timestamp, to compute Intimacy.

To further improve both the time and space efficiency of DPU, we propose the second technique: **Newton-Observe**. The key idea is that we only decay the Intimacy when we query it. In essence, Intimacy states that the older the interaction is, the lower important it is. Instead of maintaining the Intimacy and timestamp in Newton sketches, we propose the *virtual Intimacy* which implies the newer the interaction is, the more important it is, as shown in Figure 1. For every arriving interaction, we increment the virtual Intimacy by a value, which is positively related to the timestamp. Only when we want to query the Intimacy, will we decay the virtual Intimacy to calculate the Intimacy. Therefore, we transform a fluctuating Intimacy into a monotonously increasing virtual Intimacy, as shown in Figure 1(b). By recording the virtual Intimacy, we can well address both shortcomings in DPU. We don't need to record a timestamp but only the virtual Intimacy. To compare the Intimacy of two edges, we can directly compare their virtual Intimacy at any given time.

Based on Newton-Observe, we develop Newton sketches: Count-Min version (Newton-CM) and Conservative-Update version (Newton-CU) to answer the edge query. We combine the technique of Newton-Observe and two-dimensional Count-Min sketch [34] to present Newton-CM. Newton-CU optimizes Newton-CM by exploiting the technique of Conservative-Update [14]. To answer the two top-k queries, we combine the technique of Newton-Observe and Two-dimensional Space-Saving to present Newton sketches: Space-Saving version (Newton-SS). Finally, we also propose the Elastic version

(Newton-Elastic) which can simultaneously address the aforementioned tasks. The experiments show that Newton sketches can efficiently answer the above queries and Newton-Observe outperforms DPU by up to $570\times$ smaller ARE on global top-k query, $6.3\times$ smaller ARE on local top-k query and improve the throughput by up to $1.62\times$.

Key Contributions:

- We are the first to propose and strictly define the concept of Intimacy between nodes in a dynamic graph, to profile the recent relationship between different nodes.
- We propose the Newton-Observe technique and develop Newton sketches to answer three typical graph queries, edge query, global top-k query, and local top-k query, in terms of Intimacy.
- We theoretically prove that Newton sketches can estimate the Intimacy within a constant error and extensive experiments confirm that Newton sketches outperforms the baseline solution in all three tasks.

II. RELATED WORK

In this section, we give a brief introduction about the related work. There are two kinds of work for dynamic graphs. The first is sketch synopses such as CM sketches [5], CU sketches [14] and so on [13], [35], [38], [47], which proven to be effective data structures in general data streams [10], [11], [23], [30], [41], [46]. Such sketches also work for the case of graph stream, where they consider edges as items but ignore the connection between different items. These algorithms only support edge weight query but not any query related to topology of the graph. The second work such as TCM [34], gSketches [45], gMatrix [26] and GSS [19], supports multiple queries in the dynamic graph. Due to space limitation, we focus on the second kind of work in this section.

TCM is the state-of-the-art work. TCM compresses a graph stream $G = (V, E)$ into a graph sketch $G_h = (V_h, E_h)$ using a hash function $H(.)$ with the range size M. TCM maps v to the node $H(v)$ in V_h and the edge $e = \overrightarrow{(s, d)}$ to the edge $\overrightarrow{(H(s), H(d))}$ in E_h. The weight of the edge in E_h is the sum of all weights of edges mapping to it. Specifically, to represent the graph sketch, TCM uses a $M \times M$ adjacency matrix where the bucket in row $H(s)$, column $H(d)$ denotes the weight of edge $\overrightarrow{(H(s), H(d))}$ in E_h, that is, each bucket is a counter. gMatrix is a variant of TCM but uses reversible hash functions to generate graph sketches.

Similar to TCM, GSS compresses a graph stream $G = (V, E)$ to a graph sketch $G_h = (V_h, E_h)$. Nodes and edges are aggregated in G_h. The major difference of GSS and TCM is the data structure to represent graph sketch G_h. GSS uses a $M \times M$ matrix and an extra buffer B to store the graph sketch, where each bucket in the matrix consists of a counter and a fingerprint pair. The edge $e = \overrightarrow{(s, d)}$ is mapped to bucket in row $H(s)$, column $H(d)$ with a fingerprint pair. If the bucket is already occupied by other edges, GSS stores the edge in buffer B. In order to restrict the size of buffer, GSS proposes square hashing. In this technique, GSS maps a node to r rows/columns, thus mapping an edge to r^2 buckets and store the edge in the first empty one among these buckets.

All the above works measure the connection of two nodes by frequency. However, measuring the connection using frequency is not sufficient in many cases and it's important to take time into consideration [13], [15], [27]. Significant items [42] combines frequency and persistency to define the significance and put forward an algorithm named LTC to find significant items, which moves a step forward while the significance definition makes it very limited to some certain applications related to persistency. Sliding window model [17], [18] also takes recency into consideration, but due to the dynamic changing nature of the stream graph, the inherent complexity of the graph query process, running sliding window model on dynamic graph is computationally ineffective. Decaying exponentially with time has been studied in data stream [13] but it requires great efforts to apply it to dynamic graphs. To the best of our knowledge, there is no prior work dealing with Intimacy or other metrics decaying by time on dynamic graphs.

III. DEFINITION OF INTIMACY USING NEWTON'S LAW OF COOLING

In this section, we first define graph streams as well as dynamic graphs, and then formulate the Intimacy on dynamic graphs (Section III-A). Then we present three fundamental tasks of the Intimacy on dynamic graphs (Section III-B). Notations used in this section and later are given in Table I.

TABLE I: Notations and Definitions

Notations	**Definitions**
$G = (V, E)$	Dynamic graph
$\alpha(> 0)$	Attenuation coefficient
H	Temperature of the surroundings
t	Current time
t_i	Time of interaction/item i
$T(t)$	Temperature of the object at time t
I_0	Initial Intimacy
$\mathbb{I}(s, d, t)$	Intimacy of the directed edge $\overrightarrow{(s, d)}$ at time t

A. Definition of Intimacy

A graph stream [34] is a sequence of items $S = \{s_0, s_1, ...s_n\}$, where each $s_i = (\overrightarrow{v_{1,i}, v_{2,i}}, t_i)$ indicates a directed edge $e = (\overrightarrow{v_{1,i}, v_{2,i}})$ from node $v_{1,i}$ to node $v_{2,i}$ at the timestamp of t_i, for any $i \neq j$, $t_i \neq t_j$. Therefore, the graph stream forms a dynamic node set V and a dynamic edge set E. We use $G = (V, E)$ to denote the dynamic graph, where G changes with the arrival of items.

To formulate the Intimacy on dynamic graphs, we first observe the Intimacy variation process in real-world dynamic graphs. In a social network, the Intimacy between people will fade away with time if there is no interaction and the Intimacy will increase upon new interactions. Therefore, decaying with time and increasing upon new interactions are two fundamental parts of Intimacy. We observe that this process is highly similar to the water cooling process in nature, which follows Newton's law of cooling. The water cools with time going by and heats upon new hot water coming. Inspired by Newton's law of cooling, we give the formal definition of Intimacy.

Newton's law of cooling states that the rate of heat loss of a body is directly proportional to the difference in the temperatures between the body and its surroundings, which is formalized as:

$$T'(t) = -\alpha(T(t) - H) \tag{1}$$

where α denotes heat attenuation coefficient and H denotes the temperature of the surroundings. Based on Equation (1), we can get the following equations.

$$\begin{aligned}\frac{T'(t)}{T(t)-H} &= -\alpha \\ \int \frac{T'(t)}{T(t)-H}dt &= \int -\alpha dt \\ ln(T(t)-H) &= -\alpha t + c \\ \frac{T(t_1)-H}{T(t_0)-H} &= e^{-\alpha(t_1-t_0)} \\ T(t_1) - H &= (T(t_0)-H)e^{-\alpha(t_1-t_0)}\end{aligned} \tag{2}$$

Without loss of generality, we assume all objects will cool down until the temperature becomes 0, i.e., H equals to 0.

Therefore, the equation can be simplified to:

$$T(t_1) = \frac{T(t_0)}{e^{\alpha(t_1-t_0)}} \tag{3}$$

Given a sequence of items $(\overrightarrow{s,d,t_i}), i = 0, 1, ..., n$ on edge $(\overrightarrow{s,d})$, we define Intimacy of edge $(\overrightarrow{s,d})$ at time t, as the sum of Intimacy of all items on it.

$$\mathbb{I}(s,d,t) = \sum_{i=0}^{n} \frac{I_0}{e^{\alpha(t-t_i)}} \tag{4}$$

The underlying assumption of the definition is that all items share the same initial Intimacy denoted as I_0 and the same attenuation coefficient α, that is, each item comes with the same importance and gets stale at the same rate. Though frequency and recency are not explicitly shown in our definition, Intimacy takes them into consideration in an elegant way: If α is set to 0, then Intimacy is degenerated to frequency; If α is set to a large number, then Intimacy is only related to the several most recent items. Because of the exponentially decaying property, we can derive the upper bound of Intimacy:

$$\mathbb{I}(s,d,t) = \sum_{i=0}^{n} \frac{I_0}{e^{\alpha(t-t_i)}} \leq \sum_{i=0}^{+\infty} \frac{I_0}{e^{\alpha i}} = \frac{I_0 e^{\alpha}}{e^{\alpha}-1} \leq I_0 \cdot (1 + \frac{1}{\alpha}) \tag{5}$$

B. INTIMACY QUERY TASKS

There are three fundamental tasks in dynamic graph queries: edge query, global top-k query, and local top-k query. We formalize these tasks as following.

Given a graph stream S and its dynamic graph $G = (V, E)$.

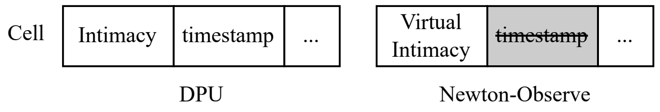


Fig. 2: Basic Unit

- **Edge Query:** Given an edge $(\overrightarrow{s,d})$, return its Intimacy if it exists in E, otherwise return 0.
- **Global Top-K Query:** Return the top-k edges with highest Intimacy on the whole graph. Note that we only need to compare the relative sizes of Intimacy of all the edges on the graph. The initial Intimacy I_0 doesn't impact the results of the global top-k query.
- **Local Top-K Query:** Given a certain node, return the top-k incident edges with highest Intimacy. Similar to global top-k query, we only need to compare the relative sizes of Intimacy of all the incident edges. The initial Intimacy I_0 doesn't impact the results of the local top-k query.
- **Subgraph Query:** Given a subgraph, return the sum of the Intimacy of all edges within it.

IV. THE NEWTON SKETCH

In this section, we first present the basic unit of the Newton sketches. Based on the basic unit, we propose three versions of Newton sketches: Count-Min version (Newton-CM), Conservative-Update version (Newton-CU), Space-Saving version (Newton-SS) and Elastic version (Newton-Elastic) to produce a dynamic graph sketch for three graph query tasks.

A. Basic Unit

The basic unit of Newton sketches is called a cell, which must contain a numerical field for Intimacy/virtual Intimacy and may contain some other fields such as ID and timestamp according to the different versions of Newton sketches, as shown in Figure 2. We first show the update operation of a cell when a new item arrives. From the definition, the Intimacy is constantly changing with time. A naive solution is to record the whole sequence of items on it $(\overrightarrow{s,d,t_i}), i = 0, 1, ..., n$ and update the Intimacy of all cells in Newton sketches for every unit time, which is unacceptable both in computation cost and memory cost. We first propose Decay-Per-Update solution to simplify the calculation, which greatly improves the insertion throughput and reduce the memory cost from the whole sequence of items to 16 bytes for each edge. However, there are some imperceptible shortcomings of DPU solution. We further propose Newton-Observe to improve the seemingly effective DPU solution.

1) Decay-Per-Update Solution: In the DPU solution, a cell must contain a field for the Intimacy and a field for the timestamp. For each item, it only update the corresponding unit and leave all other units unchanged. Only when we query a certain edge, is the accurate Intimacy calculated.

Formally, suppose there is a sequence of items $(\overrightarrow{s,d},t_i), i = 0,1,...,n-1$ on edge $(\overrightarrow{s,d})$, and a new item $(\overrightarrow{s,d},t_n)$ comes, we can calculate the Intimacy in the following way.

$$\begin{aligned}\mathbb{I}(s,d,t_n) &= \sum_{i=0}^{n-1} \frac{I_0}{e^{\alpha(t_{n-1}-t_i)}} \times e^{-\alpha(t_n-t_{n-1})} + I_0 \\ &= \frac{\mathbb{I}(s,d,t_{n-1})}{e^{\alpha(t_n-t_{n-1})}} + I_0\end{aligned} \tag{6}$$

According to Equation (6), to calculate the current Intimacy $\mathbb{I}(s,d,t_n)$, we only need to record the previous Intimacy $\mathbb{I}(s,d,t_{n-1})$, and the most recent timestamp t_{n-1}. Every time a new item comes, we can decay the previous Intimacy according to the interval time $t_n - t_{n-1}$, and add I_0 to get the current Intimacy. At the first glance, there is no memory and computational redundancy in this solution, but actually both the time and space cost can be significantly reduced by our following solution.

2) Newton-Observe Solution: In Newton-Observe, a cell contain a field for the virtual Intimacy but don't need the timestamp. Different from Decay-Per-Update solution decaying Intimacy for every coming item, Newton-Observe solution performs the decay operation only when we indeed query the intimacy. Essentially, Equation (6) states the principle that the earlier the item comes, the smaller the Intimacy is and the newly arrived item has the Intimacy I_0. In Newton-Observe solution, instead of decaying the previous Intimacy when inserting a new item, we assign a larger virtual Intimacy for the newly coming item. We can directly obtain the following equation from Equation (6).

$$\mathbb{I}(s,d,t_n)e^{\alpha t_n} = \mathbb{I}(s,d,t_{n-1})e^{\alpha t_{n-1}} + I_0 e^{\alpha t_n} \tag{7}$$

Though Equation (7) seems to have no great difference with Equation (6), it helps to further improve both memory and computation efficiency. With Equation (7), we name $\mathbb{I}(s,d,t_n)e^{\alpha t_n}$ as virtual Intimacy, denoted as v. Every time a new item$(\overrightarrow{s,d},t)$ comes, we add $I_0 e^{\alpha t}$ to the virtual Intimacy.

$$v := v + I_0 e^{\alpha t} \tag{8}$$

There is no need to store the most recent timestamp. It's also easy to calculate the Intimacy from the virtual Intimacy given any timestamp t.

$$\mathbb{I}(s,d,t) = \frac{v}{e^{\alpha t}} \tag{9}$$

Figure 1 illustrates the differences between Intimacy and virtual Intimacy, i.e., Intimacy changes over time while virtual Intimacy only changes at every insertion. In the example, we only need to perform one decay when we query the Intimacy using Newton-Observe while it takes four decay using DPU. Therefore, maintaining virtual Intimacy is much easier than Intimacy.

Compared with DPU, the technique of Newton-Observe not only saves the memory, but also greatly simplifies the computation. When performing comparison in DPU, the Intimacy of the two edges needs to be aligned to the same timestamp while in Newton-Observe, we can directly compare the virtual Intimacy. Therefore, Newton-Observe is especially beneficial in the top-k query which needs multiple comparison for every insertion. Based on the basic unit, we develop several sketches to address the three fundamental tasks illustrated in Section III-B.

B. Newton Sketches: CM Version

First we propose Newton sketches: Count-Min version (Newton-CM). Newton-CM compresses a graph stream $G = (V,E)$ into a graph sketch $G_h = (V_h, E_h)$ using a hash function $H(.)$ with the range size M. Without any auxiliary data structure, Newton-CM can only support edge query but not top-k query because it doesn't maintain the information of the source and the destination node of the edge. To support global top-k query, we need an extra min-heap which maintains the source node, destination node, and virtual Intimacy v of the current top-k edges on the graph. However, with the extra min-heap, it still does not support local top-k query.

As shown in Figure 3, Newton-CM with parameters I_0, α is represented by a two-dimensional array with width M and depth M: $cell[0][0],...,cell[M-1][M-1]$. $cell[i][j]$ denotes the virtual Intimacy of the directed edge $(\overrightarrow{i,j})$ in E_h.

Initially, we choose l hash functions $H_i(.), i = 1,2,...,l$, whose values are ranged from $[0,M)$. Then we can generate the graph sketch with the following operations.

Initialization: Each field of the array is initialized to zero.

Edge Insertion: When a graph stream item$(\overrightarrow{s,d},t)$ comes, we first compute l hash functions for both nodes s and d: $H_1(s),...,H_l(s),H_1(d),...,H_l(d)$. In another word, we map node s to l different nodes in V_h and node d to l different nodes in V_h, thus mapping edge $(\overrightarrow{s,d})$ to the l^2 edges in E_h. Then we update these l^2 edges using Newton-Observe, that is, we add the $I_0 e^t$ to the $cell[H_1(s)][H_1(d)],...,cell[H_l(s)][H_l(d)]$ respectively.

Edge Query: When querying the Intimacy of a given edge $(\overrightarrow{s,d})$ at the timestamp of t, we find out the minimum Intimacy of the l^2 edges in E_h, i.e., the minimum virtual Intimacy v_m of $cell[H_1(s)][H_1(d)],...,cell[H_l(s)][H_l(d)]$ and return $\frac{v_m}{e^{\alpha t}}$.

Global Top-K Query: To perform global top-k query, besides an extra min-heap, we need to add an additional step for the edge insertion. For each incoming item $s = (\overrightarrow{v_1,v_2},t)$, we first insert s to Newton-CM, and then query the virtual Intimacy: v_s. If s is in the heap, we add $I_0 e^t$ to the virtual Intimacy of s in the heap. If v_s is not in the heap but it is larger than the virtual Intimacy of the minimum element of the heap, the minimum element is evicted from the heap and s is inserted into the heap. To report the global top-k query, we report the top-k items with the highest virtual Intimacy in the heap. Maintaining the virtual Intimacy v_s rather than the Intimacy (DPU solution) in the heap achieves much higher throughput. Note that the Intimacy of the item changes with time. When using DPU solution, if we want to compare with the minimum element in the heap, we need to recalculate the

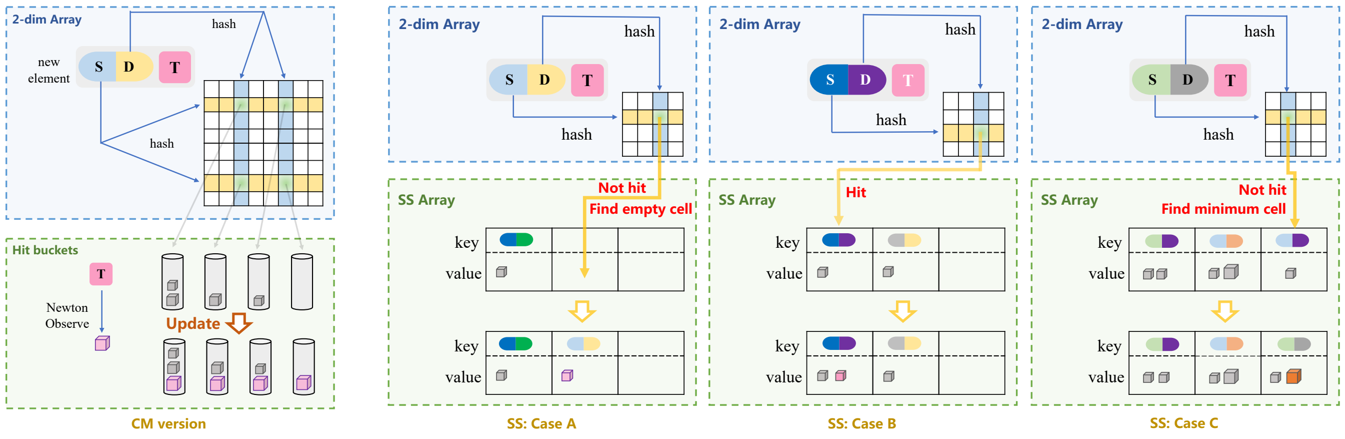


Fig. 3: Newton sketches

decayed Intimacy of the minimum element. If the replacement happens, i.e., to insert an item into the min-heap, we need to recalculate multiple Intimacy of the items stored in the heap. When using Newton-Observe, since the virtual Intimacy in the heap remains unchanged with time, we can directly compare the virtual Intimacy to decide their relative magnitude.

Subgraph Query: We can return the sum of the intimacy of all edges within a subgraph, which is meaningful. For example, communities in social networks are such subgraphs, and our algorithm can estimate the activity level of these communities.

Other Queries: Based on our data structure, we can support a wide range of graph queries. For instance, node queries, where the intimacy of a row or a column can serve as an estimate for all out-degrees or in-degrees of a node, with multiple hashing reducing the error of this task. For path queries, depending on the task, we can aggregate the intimacy of all edges on the path by summing them up or taking the minimum value to address the path query. However, not all graph algorithms can be integrated with the concept of intimacy, which opens up many questions for exploration.

The overall memory cost of Newton-CM is $O(|E|)$. We usually set $M = \beta \times \sqrt{|E|}$, where β is a constant, approximate to 1. To achieve better performance, we can set a large β (proved in Section V-A). When the memory is not sufficient, we can also use smaller M with the sacrifice of accuracy. For edge insertion, the time cost $O(l^2)$, where l is a constant. f we want to support global top-K query, the insertion time cost is $O(l^2 + log_2 K)$, where both l and k are constants. Overall, Newton-CM can achieve constant update speed.

C. Newton Sketches: CU Version

To reduce the over-estimation in Newton-CM, we propose Newton sketches: Conservative-Update version (Newton-CU). Newton-CU is the same as Newton-CM except for edge insertion. Instead of updating all the hashed edges in E_h, Newton-CU only updates some edges. When a graph stream item $(\overrightarrow{s,d},t)$ arrives, we first find out the minimum virtual Intimacy v_{min} of the l^2 edges, i.e., $v_{min} = min\{cell[H_1(s)][H_1(d)], ..., cell[H_k(s)][H_k(d)]\}$. Then for all the l^2 edges whose virtual Intimacy is less than $v_{min}+I_0e^t$, we set the virtual Intimacy of these edges to $v_{min} + I_0e^t$, i.e., $cell[H_i(s)][H_j(d)] = max\{cell[H_i(s)][H_j(d)], v_{min} + I_0e^t\}$, for all the $i, j = 1, 2, ..., l$. Newton-CU achieves the same memory and time cost as Newton-CM.

D. Newton Sketches: Space-Saving Version

Both Newton-CM and Newton-CU are primarily designed for edge query and cannot support local top-k query. We further propose Newton sketches: Space-Saving version (Newton-SS), which can support all three query tasks. As shown in Figure 3, a Newton-SS with parameters I_0, α is represented by a two-dimensional array with width M and depth M. We use *bucket* to denote the array: $bucket[0][0], ..., bucket[M-1][M-1]$. Each *bucket* has m cells. Each cell consists of two fields: $key, value$, where key consists of the nodes of the edge $(\overrightarrow{s,d})$ in G (not hashed edge in G_h) and $value$ records virtual Intimacy of the edge.

We choose one hash function $H(.)$ whose value is ranged from $[0, M)$. Then we can generate the graph sketch with the following operations.

Initialization: Each field of the array is initialized to zero.

Edge Insertion: When a graph stream item $(\overrightarrow{s,d},t)$ comes, we first map the node s to the node $H(s)$ in V_h and the node d to the node $H(d)$ in V_h, thus mapping the edge $(\overrightarrow{s,d})$ to the edge $(\overrightarrow{H(s),H(d)})$ in E_h. Then we scan the $bucket[H(s)][H(d)]$.

1) As shown in Case A, if $(\overrightarrow{s,d})$ is not in the $bucket[H(s)][H(d)]$ and there are empty cells. We randomly choose one empty cell and set the cell's key to $(\overrightarrow{s,d})$, $value$ to $I_0e^{\alpha t}$.

2) As shown in Case B, if $(\overrightarrow{s,d})$ is in the $bucket[H(s)][H(d)]$, we add $I_0e^{\alpha t}$ to the $value$ of the cell.

3) As shown in Case C, if $(\overrightarrow{s,d})$ is not in the $bucket[H(s)][H(d)]$ and there is no empty cell. We find out

the cell with the smallest $value$. Then we add $I_0 e^{\alpha t}$ to the $value$ and replace the key with $\overrightarrow{(s,d)}$.

Edge Query: When querying the Intimacy of a given edge $\overrightarrow{(s,d)}$ at the timestamp t, we compare $\overrightarrow{(s,d)}$ with the keys in the $bucket[H(s)][H(d)]$. If $\overrightarrow{(s,d)}$ is in the bucket, we get the corresponding $value$ v and return $\frac{v}{e^{\alpha t}}$. Otherwise, we return 0 which means we don't know its Intimacy.

Global Top-K Query: To perform global top-k query, we traverse all the cells in all the buckets and report the edges with the top-k highest $value$, i.e., virtual Intimacy.

Local Top-K Query: To perform local top-k query of node s, we traverse all the cells incident to node s, i.e, all the cells in the $bucket[H(s)][j], j = 0, 1, ..., M-1$ and $bucket[i][H(s)], i = 0, 1, ..., M-1$, and report the top-$k$ edges incident to s. More specifically, for every cell in the $bucket[H(s)][j], j = 0, 1, ..., M-1$, only those whose source node in key is exactly s are regarded as edges incident to node s. For every cell in the $bucket[i][H(s)], i = 0, 1, ..., M-1$, only those whose destination node in key is exactly s are regarded as edges incident to node s

The overall memory cost of Newton-SS is also $O(|E|)$, and we usually set $M = \beta \times \sqrt{\frac{|E|}{m}}$, where β is a constant, approximate to 1. When the memory is fixed, there is a trade off between M and m, i.e., the number of buckets and the number of cells in a bucket. We experimentally study the trade off and find setting $m = 8$ achieves a balance. For edge insertion, the time cost of Newton-SS is $O(m)$, where m is a constant, so Newton-SS achieves constant update speed.

E. Newton Sketches: Elastic Version

Newton-Elastic can simultaneously solve edge queries, global top-k queries, and local top-k queries. Inspired by the Elastic Sketch algorithm [40] for general data streams, we extended it to address dynamic graph problems. It consists of two parts: the heavy part and the light part. The heavy part of Newton-Elastic stores the global Top-k. Unlike a typical Elastic Sketch, we employed a structure similar to Newton-SS, enabling Newton-Elastic to address local Top-k issues through querying the rows and columns obtained from hashes. The replacement strategy for the heavy part adapts Elastic Sketch to Newton-Observe, using votes against Virtual Intimacy to manage replacements. The light part of Newton-Elastic is Newton-CU, which allows it to handle edge queries. While Newton-SS may miss edges, Newton-Elastic can always return results from the light part.

Initialization: Each field of the array is initialized to zero.

Edge Insertion: The first two insertion scenarios of Newton-Elastic are consistent with those of Newton-SS. H owever, when there is no empty cell for the third scenario, the replacement strategy differs. Drawing from Elastic Sketch, Newton-Elastic records how many times a "vote against" has occurred for each bucket, using our Newton-Observe method. When the virtual intimacy of the "votes against" is eight times greater than the current bucket's minimum cell, the minimum cell is moved to the light part, and the "votes against" are cleared (the factor of eight is a hyperparameter introduced by Elastic Sketch, which has shown good performance in our experiments).

Query: Global Top-K Query and Local Top-K Query in Newton-Elastic are similar to those in Newton-SS. The difference lies in that, during each query, the items in the heavy part are not pure values and require querying in the light part's Newton-CU to compensate for the error. In the Edge Query task, if the target is not found in the heavy part, Newton-SS can only return 0, whereas Newton-Elastic can query from the light part and return the result.

V. Error Bounds of Newton sketches

In this section, we analyse the error bounds of the Newton sketches. Due to the limitation of the space, we only present the proofs for Newton-CM and Newton-SS. The proof for Newton-CU are similar to the proof of Newton-CM.

A. Error Bound of Newton-CM

In this section, we theoretically prove that Newton-CM can estimate the Intimacy within an additive **constant error** to the real Intimacy in Theorem 1. For sketches on frequency, which increases monotonously, the estimation error always increases monotonously with items coming. Because the stale information is automatically discarded in Intimacy, Intimacy promise the potential to be estimated within a constant error. We first give two lemmas. Based on two lemmas, we prove the Theorem 1.

Lemma 1. *For any edge $a = \overrightarrow{(s,d)}$, at any time t, the estimated Intimacy $\hat{\mathbb{I}}_a$ is equal or larger than $\mathbb{I}_a$: $\hat{\mathbb{I}}_a \geq \mathbb{I}_a$.*

Proof. For any edge, we can find l^2 counters to estimate its Intimacy. Consider edge a and let $X_a(i,j)$ $(i,j = 1, 2, ..., l)$ to denote the contribution from other edges to $counter[H_i(s)][H_j(d)]$. We can easily get $X_a(i,j)$ is always equal or larger than zero as virtual Intimacy is always larger than zero.

$$counter[H_i(s)][H_j(d)] = \mathbb{I}_a e^{\alpha t} + X_a(i,j) \geq \mathbb{I}_a e^{\alpha t} \tag{10}$$

Since all the l^2 counters are equal or larger than $\mathbb{I}_a e^{\alpha n}$, we can conclude that:

$$\min_{i,j}(counter[H_i(s)][H_j(d)]) \geq \mathbb{I}_a e^{\alpha t} \tag{11}$$

Thus, we get:

$$\hat{\mathbb{I}}_a = \frac{\min_{i,j}(counter[H_i(s)][H_j(d))}{e^{\alpha t}} \geq \mathbb{I}_a \tag{12}$$

□

Lemma 2. *At time t, for any edge $a = \overrightarrow{(s,d)}$, $\mathbb{E}(X_a(i,j)) < \frac{I_0 l^2 e^{\alpha(t+1)}}{M^2 \alpha}$.*

Proof. Each edge are mapped to l^2 cells, so edge $a^{'} (a^{'} \neq a)$ will be mapped to $counter[H_i(s)][H_j(d)]$ with the probability of $\frac{l^2}{M^2}$. At time t, the upper bound of virtual Intimacy is $\sum_{k=0}^{t} I_0 e^{\alpha k}$. We can derive the expectation of $X_a(i,j)$ as follows:

$$\begin{aligned}\mathbb{E}[X_a(i,j)] &\leq \frac{l^2}{M^2}\sum_{k=0}^{t} I_0 e^{\alpha k} \\ &= \frac{I_0 l^2 (e^{\alpha(t+1)}-1)}{M^2(e^\alpha - 1)} \\ &< \frac{I_0 l^2 e^{\alpha(t+1)}}{M^2(e^\alpha - 1)} \\ &< \frac{I_0 l^2 e^{\alpha(t+1)}}{M^2 \alpha}\end{aligned} \tag{13}$$

□

Theorem 1. *For any edge $a = (\overrightarrow{s,d})$, the estimated Intimacy $\hat{\mathbb{I}}_a$ has the following guarantee: with the probability of at least $1-\delta$, $\hat{\mathbb{I}}_a < \mathbb{I}_a + \frac{I_0 l^2 e^{1+\alpha}}{M^2\alpha}$, where $\delta = e^{-l^2}$.*

Proof. At any time t, use Markov's inequality and we can get:

$$\begin{aligned}Pr(X_a(i,j) \geq e \cdot \frac{I_0 l^2 e^{\alpha(t+1)}}{M^2\alpha}) &\leq \frac{\mathbb{E}(X_a(i,j))}{e \cdot \frac{I_0 l^2 e^{\alpha(t+1)}}{M^2\alpha}} \\ &< \frac{\frac{I_0 l^2 e^{\alpha(t+1)}}{M^2\alpha}}{e \cdot \frac{I_0 l^2 e^{\alpha(t+1)}}{M^2\alpha}} \quad < \frac{1}{e}\end{aligned} \tag{14}$$

For different i, j, $X_a(i,j)$ is independent to each other. With multiplication theorem of probability, we conclude that:

$$Pr(\forall i,j \ X_a(i,j) \geq e \cdot \frac{I_0 l^2 e^{\alpha(n+1)}}{M^2\alpha}) < (\frac{1}{e})^{l^2} \quad = e^{-l^2} = \delta \tag{15}$$

Let $\delta = e^{-l^2}$, with the probability of at least $1-\delta$:

$$\begin{aligned}\hat{\mathbb{I}}_a &= \frac{\min_{i,j}(counter[H_i(s)][H_j(d)])}{e^{\alpha n}} \\ &= \frac{\min_{i,j}(\mathbb{I}_a e^{\alpha n} + X_a(i,j))}{e^{\alpha n}} \\ &< \mathbb{I}_a + e \cdot \frac{I_0 l^2 e^{\alpha(n+1)}}{M^2 \alpha e^{\alpha n}} \\ &= \mathbb{I}_a + \frac{I_0 l^2 e^{1+\alpha}}{M^2\alpha}\end{aligned} \tag{16}$$

□

Note that the error bound is a constant given all fixed parameters I_0, l, M, α. To achieve a smaller error bound, the most straightforward and efficient way is increasing M, i.e., increasing the memory usage. Because finding the minimum counter can be done in linear time, the time to produce the estimation is $O(l^2) = O(ln(1/\delta))$.

B. Error Bound of Newton-SS

In this section, we derive the error bound for global top-k queries in Newton-SS. Suppose there are N different edges, with Intimacy $\mathbb{I}_1$, $\mathbb{I}_2$, ..., $\mathbb{I}_N$ and M^2 buckets. Each bucket has m cells. We prove that at least $(1-\delta)k$ global top-k edges will retain in the buckets with the fixed probability (Theorem 5).

Lemma 3. *Suppose we want to put X identical balls into Y different boxes and make sure there are at most Z balls in each box. There are $\sum_{r=0}^{\lfloor \frac{X}{Z+1} \rfloor}(-1)^r C_Y^r C_{X+Y-(Z+1)r-1}^{X-(Z+1)r}$ combinations to satisfy the conditions.*

Proof. We use generation function $(1+u+u^2+...+u^Z)^Y$ to solve the problem. Each box can contain zero to D balls and there are B boxes. There are X balls in total, so we can get the number of cases by calculating the coefficient of u^X.

$$\begin{aligned}(1+u+...+u^Z)^Y &= (\frac{1-u^{Z+1}}{1-u})^Y \\ &= (1-u^{Z+1})^Y(1-u)^{-Y} \\ &= \sum_{r_1=0}^{+\infty}(-1)^{r_1} C_Y^{r_1} u^{(Z+1)r_1} \sum_{r_2=0}^{+\infty} C_{Y+r_2-1}^{r_2} u^{r_2}\end{aligned} \tag{17}$$

The coefficient of u^X, i.e., the number of combinations satisfying the conditions, is $\sum_{r=0}^{\lfloor \frac{X}{Z+1} \rfloor}(-1)^r C_Y^r C_{X+Y-(Z+1)r-1}^{X-(Z+1)r}$. □

Theorem 2. *The probability of a bucket being mapped by at most n different edges is $P_1 = M^{-2N} \cdot \sum_{r=0}^{\lfloor \frac{N}{n+1} \rfloor}(-1)^r C_{M^2}^r C_{N+M^2-(n+1)r-1}^{N-(n+1)r}$.*

Proof. Edges are independent to each other and will be mapped each bucket with the same probability. We have N different edges and M^2 buckets, so the number of all the cases is M^{2N}. Using Lemma 3, we can calculate the number of cases that satisfy our assumption is $\sum_{r=0}^{\lfloor \frac{N}{n+1} \rfloor}(-1)^r C_{M^2}^r C_{N+M^2-(n+1)r-1}^{N-(n+1)r}$. Then we can get the conclusion. □

Lemma 4. *The sum of values stored in the cells of a bucket, denoted as S, is equal to the sum of all the Intimacy of edges mapped to this bucket.*

Lemma 5. *The value of minimum cell (min_val) is no greater than $\lfloor \frac{S}{m} \rfloor$.*

Lemma 6. *Any edge, which is mapped to the bucket and has Intimacy larger than the min_val of the bucket, must retain in the bucket.*

Theorem 3. *Suppose the Intimacy satisfies zipfian distribution with parameter α_z and each bucket is mapped by at most n different edges. If $m > 1 + \frac{1}{1-\alpha_z}[(k+n-1)(\frac{k}{k+n-1})^{\alpha_z} - k]$ and the bucket is accessed by only one global top-k edge, the global top-k edge must retain in this bucket.*

Proof. From Lemma 4, 5, 6, if the Intimacy of a global top-k edge is larger than min_val of the mapped bucket, it must retain in the bucket. We consider the lowest Intimacy, denoted as $\mathbb{I}_k$, among all the global top-k edges $\mathbb{I}_i, i = 1,2,...,k$. Because each bucket is accessed by only one global top-k edge and at most n different edges, the min_val of the bucket mapped by edge $\mathbb{I}_k$ is less than $\frac{\sum_{i=k}^{k+n-1}\mathbb{I}_i}{m}$.

With the assumption that $m > 1 + \frac{1}{1-\alpha_z}[(k+n-1)(\frac{k}{k+n-1})^{\alpha_z} - k]$, i.e., each bucket has relatively sufficient cells to store edges mapped to it, we can derive the following in-equations:

$$\begin{aligned}\frac{m-1}{k^{\alpha_z}} &> \frac{1}{(1-\alpha_z)}[(k+n-1)^{1-\alpha_z} - k^{1-\alpha_z}] \\ &= \int_k^{k+n-1} \frac{1}{x^{\alpha_z}} dx \\ &> \sum_{i=k+1}^{k+n-1} \frac{1}{i^{\alpha_z}}\end{aligned} \tag{18}$$

Because Intimacy follows the zipfian distribution, Equation (18) essentially states

$$\mathbb{I}_k > \frac{\sum_{i=k+1}^{k+n-1} \mathbb{I}_i}{m-1} > \frac{\sum_{i=k}^{k+n-1} \mathbb{I}_i}{m} > min\_val \tag{19}$$

Therefore, we prove that the Intimacy of $\mathbb{I}_k$ is greater than the *min_val*. As other global top-k edges have larger Intimacy, they will retain in the mapped buckets because of the same reason. □

Theorem 4. *The probability of more than $(1-\delta)k$ buckets are mapped by only one global top-k edge is greater than $\frac{V!U^{\delta k}}{V^k U!}$ if each bucket is mapped by no more than n different edges, where $V = \frac{N}{n}$, $U = V - (1-\delta)k$.*

Proof. We consider the worst case that there are V buckets, each of them is accessed by n different edges where different global top-k edges are most likely to meet each other (mapped to the same bucket). Every global top-k edge can access at most V different buckets. Therefore, the number of different permutations of global top-k edges is no more than V^k. We split the global top-k edges into global top-$(1-\delta)k$ edges and the left δk edges. The number of cases that the global top-$(1-\delta)k$ edges do not meet other global top-k edges is no smaller than $\mathrm{A}_V^{(1-\delta)k}(V-(1-\delta)k)^{\delta k}$. The probability of at least $(1-\delta)k$ buckets being mapped by only one global top-k edge (P_2) is at least:

$$\begin{aligned}P_2 &\geq \frac{\mathrm{A}_V^{(1-\delta)k} \cdot (V-(1-\delta)k)^{\delta k}}{V^k} \\ &= \frac{\binom{V}{(1-\delta)k} \cdot [(1-\delta)k]! \cdot (V-(1-\delta)k)^{\delta k}}{V^k} \\ &= \frac{V!U^{\delta k}}{V^k U!} \; (U = V-(1-\delta)k)\end{aligned} \tag{20}$$

□

Theorem 5. *Suppose the Intimacy satisfies zipfian distribution with parameter α_z and $m > 1+\frac{1}{1-\alpha_z}[(k+n-1)(\frac{k}{k+n-1})^{\alpha_z} - k]$. With probability of at least $\frac{V!U^{\delta k}P_1}{V^k U!}$, at least $(1-\delta)k$ global top-k edges will retain in the buckets.*

Proof. Combining Lemma 2, 3, 4, we can draw the conclusion. □

VI. Experimental Evaluation

In this section, we show the efficiency of Newton sketches by comparing the Newton-Observe with DPU solution. We use DPU-CM, DPU-CU, and DPU-SS for the DPU based Newton sketches and Newton-CM, Newton-CU, and Newton-SS for the Newton-Observe based Newton sketches. Newton-Elastic is based on Newton-Observe. We start by explaining the experimental environments (Section VI-A). We evaluate the performance of Newton sketches over two real-world data sets and one synthetic data set. The results show that Newton sketches can sufficiently achieve considerable performance on all the dynamic graph query tasks (Section VI-B). We set $l=2$ and $m=8$ as our default setting according to the experimental results. We set $I_0 = 1, \alpha = 0.000001$, which means every time an edge comes, the Intimacy of the edge increase by $I_0 = 1$ and after $\frac{1}{\alpha} = 1000000$ units of time, the Intimacy of the edge decays by e times.

A. Experimental Setup

1) Data Sets and Setup:

1) CAIDA: This data set comes from CAIDA Anonymized Internet Trace 2016 [4], which consists of 10M IP packets. We regard each IP packet as an item of a graph stream, that is, we regard the source IP address as the source node, the destination IP address as the destination node and the index as the timestamp. This graph contains 411509 vertices and 440449 static directed edges.

2) Social: This data set is a temporal network of interactions on the stack exchange web site [28], which consists of 10M items. Each item is represented by a directed edge (u, v, t) which means user u answers user v's question at the timestamp of t. This graph contains 1256732 vertices and 9107956 static directed edges.

3) MovieLens: The well-known MovieLens dataset [22], which includes numerous user ratings of movies, is used to test the performance of our algorithm on bipartite graphs like those found in recommendation system networks. The original dataset does not contain duplicate edges; we randomly generated arrival times and occurrence frequencies for each edge, constructing a dataset containing 100M items.

4) Friendster: Another social network dataset collected by Stanford [28], which includes community information, is utilized to test the performance of our algorithm on subgraph query tasks. The original dataset does not contain duplicate edges; we randomly generated arrival times and occurrence frequencies for each edge, constructing a dataset containing 100M items. Additionally, this dataset provides high quality community information, which we utilize to test the reliability of our algorithm in subgraph query tasks.

5) Synthetic: We generate 5 data sets using the well-known graph generator GTGraph [7], each has 100M items. We first use the R-MAT model [9] to generate a large network with the power-low degree distribution. Then we set the times of appearance of each edge using Zipfian distribution and randomly scatter these edges. We vary the skewness of Zipfian

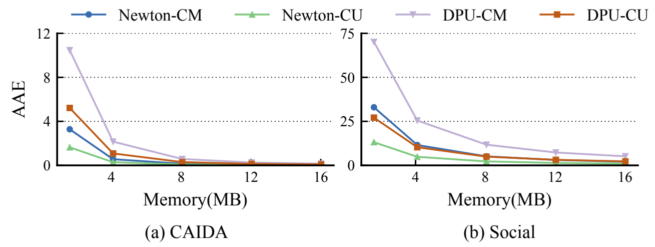


Fig. 4: Eage Query AAE vs. memory size

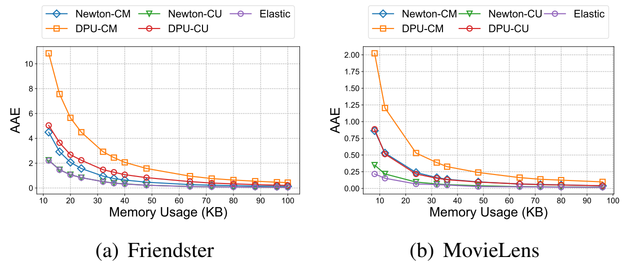


Fig. 5: Eage Query AAE vs. memory size

distribution from 1.4 to 3.0 with a step of 0.4 to generate 5 data sets. The generated graph contains 10000 vertices and about 1 million static directed edges.

Implementation: We implement all versions of Newton sketches in C++. We use Bob Hash [1] as our hash function. We run all the programs on a server with 18-core CPUS (36 threads, Intel CPU i9-10980XE @3.00 GHz) with 32 kB/32 kB instruction/data cache, 1 MB L2 cache, and 24.75 MB L3 cache, and 128 GB total system memory. Codes of Newton sketches are open-sourced at GitHub anonymously [2].

2) Metrics:

AAE / ARE: AAE/ARE measures the absolute/relative accuracy of the reported Newton sketches in edge queries. Given a query q, the absolute error(AE) and the relative error(RE) are formalized as:

$$\mathbb{AE}(q) = \left|\hat{f}(q) - f(q)\right|, \mathbb{RE}(q) \quad = \frac{\left|\hat{f}(q) - f(q)\right|}{f(q)} \qquad (21)$$

where $\hat{f}(q)$ and $f(q)$ are the estimated value of q and the real value of q respectively. Given a query set, the average absolute error(AAE)/average relative error(ARE) is the average of the AE/RE over all queries on it. Note that AAE is influenced by I_0 and α much greater than ARE, we use ARE as our metric in most of our experiments and only use AAE as our metric in the experiments Section VI-B1.

Precision: Precision measures the ratio of correctly reported answers to the number of reported answers. Suppose the correct top-k set is ψ, the estimated top-k set is ϕ, the precision is defined as $\frac{|\phi\cap\psi|}{|\phi|}$.

Throughput: Throughput evaluates the insertion speed of Newton sketches. We use Millions operations per second (Mops) to denote the throughput.

B. Experimental Results

1) Experiments on edge query:

In this section, we evaluate the performance of Newton-CM, Newton-CU and Newton-Elastic on edge query. The performance of Newton-SS is not showed because it cannot answer the edge query on items which are not stored in it. First, we study the performance on different memory sizes, i.e., with different M. Second, we study the performance on the synthetic data sets with different skewness. Third, we study the performance varying the number of hash functions. In this series of experiments, we query all the edges in the network where edges with low Intimacy are of little importance but greatly influence ARE, so we compare the AAE rather than ARE in this section.

1) AAE vs. memory size. In this experiment, we vary the memory size from 1MB to 16MB for the CAIDA and Social datasets, while for the Friendster and MovieLens datasets, we use from 10KB to 100KB. Although these two datasets involve processing more edges, the graphs after deduplication are not large, requiring less memory than CAIDA and Social. As shown in Figure 4 and Figure 5, with the memory size increasing, the AAE decreases fast and converges to zero. The AAE in the social data set is greater than the AAE in the CAIDA data set with the same memory size because the social data set is much more sparse than CAIDA data set and the Intimacy on it is much more evenly distributed. Newton-Observe improves the AAE up to $4.1\times$ comparing with DPU solution. When the memory size is small, Newton-Observe performs much better than the DPU solution and when the memory size increases, the gap gradually narrows. CU always outperforms CM, and Elastic always performs slightly better than CU.

2) AAE vs. skewness. In this experiment, we set the memory to 16MB. From Figure 6, it is observable that the relative order of the five algorithms remains consistent. The relationship between AAE and Skewness does not appear to be significant, as our algorithm is always related to the number of edges, and the number of edges after deduplication in these generated datasets is similar. This also proves that our algorithm can operate across various graph structures.

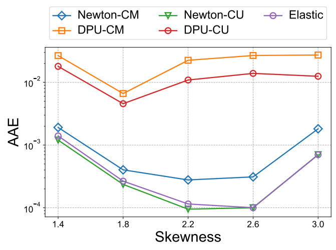


Fig. 6: Eage Query AAE vs. skewness

2) Experiments on global top-k query:

In this section, we evaluate the performance of all algorithms on global top-k query. First, we study the global top-k query performance on different memory sizes and different k. Then, we study the influence of the number of hash functions to the Newton-CM and Newton-CU and the influence of the

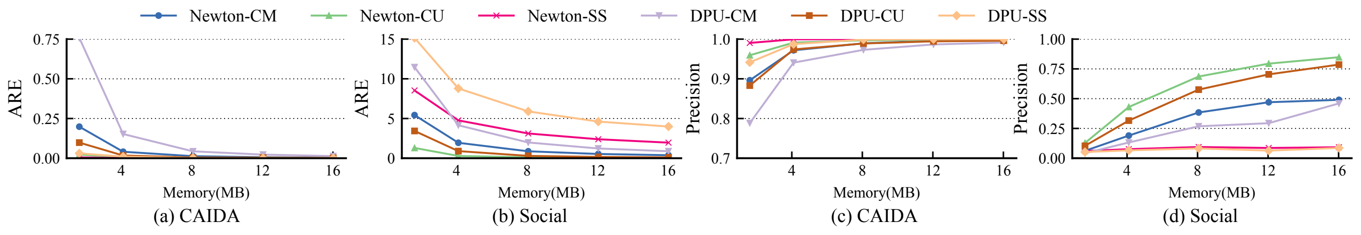


Fig. 7: Global Top-k ARE, Precision vs. memory size

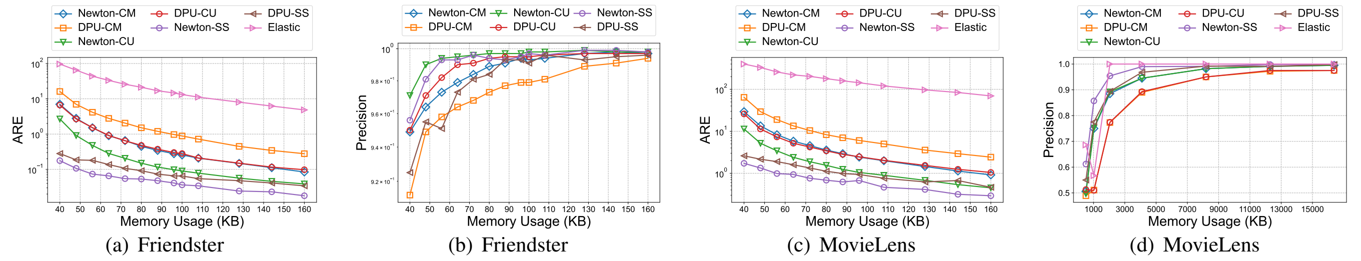


Fig. 8: Global Top-k ARE, Precision vs. memory size

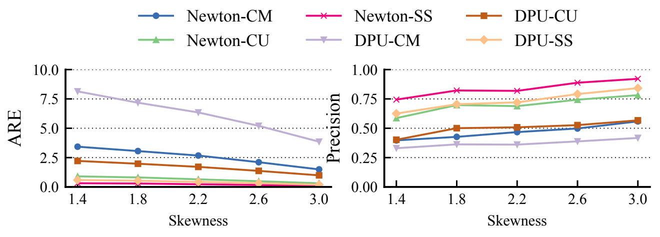


Fig. 9: Global Top-k ARE, Precision vs. skewness

cell number to the Newton-SS. At last, we study the influence of the data set skewness on global top-k query.

1) ARE, Precision vs. memory size. In this experiment, we set $k = 10000$ and vary the memory size from 1MB to 16MB for the CAIDA and Social datasets, while for the Friendster and MovieLens datasets, we use from 10KB to 200KB. Figure 7 shows the results. With the memory increases, the ARE decreases and precision increases. Newton-Observe improves the precision by up to $1.45\times$ and achieves the smaller ARE by up to $570\times$ compared with DPU. The performance of Newton-Elastic is not very good. We believe this is reasonable. Because, in fact, the other algorithms are specifically designed for this task: for example, Newton-CM requires the addition of a heap. Whereas, Newton-Elastic is a single algorithm that addresses all tasks simultaneously and has already shown excellent performance in edge query tasks. Therefore, while its accuracy in global Top-k tasks is not the highest, it is still acceptable.

2) ARE, Precision vs. k. In this experiment, we set memory size to $640KB$ for CAIDA data set and $16MB$ for social data set and vary the k from 100 to 10000. As shown in Figure 10, with the k increasing, the ARE of all algorithms increases and the precision decreases. Both ARE and precision of Newton-SS and DPU-SS in the social data set increase with k increasing, which seems counter-intuitive. The reason is that in the social data set, there is little difference in Intimacy of edges. In this experiment, though we do accurately store the real top-k edges in Newton-SS, we don't report most of them as top-k edges when k is small because they are easily overwhelmed by other edges. When k increases, then these real top-k edges are reported, thus increasing the precision. Newton-Observe improves the precision by up to $2.25\times$ and achieves the smaller ARE by up to $7.14\times$ compared with DPU.

3) ARE, Precision vs. skewness. In this experiment, we set memory size to $128KB$ and $k = 1500$ and study the performance of Newton sketches on the synthetic data sets. As shown in Figure 9 Newton-Observe improves the precision by up to $1.46\times$ and improves the ARE by up to $3.21\times$ compared with DPU. With the skewness increasing, the ARE of all algorithms decreases and the precision increases.

3) Experiments on local top-k query:

Because Newton-CM and Newton-CU don't support local top-k query, we only evaluate the efficiency of Newton-SS and Newton-Elastic in this section. Because not every node connects more than k other nodes and typically one node has several to tens of nodes with high Intimacy, we only query the nodes which connect more than k nodes. First, we study the impact of the memory size to local top-k query. Then, we study the performance of Newton-SS local top-k query with different k. At last, we study the influence of cell numbers on local top-k query.

1) ARE, Precision vs. memory size. In this experiment, we set $k = 5$ and vary the memory size from $128KB$ to $640KB$ for Friendster dataset. It can be seen from in Figure 11, that

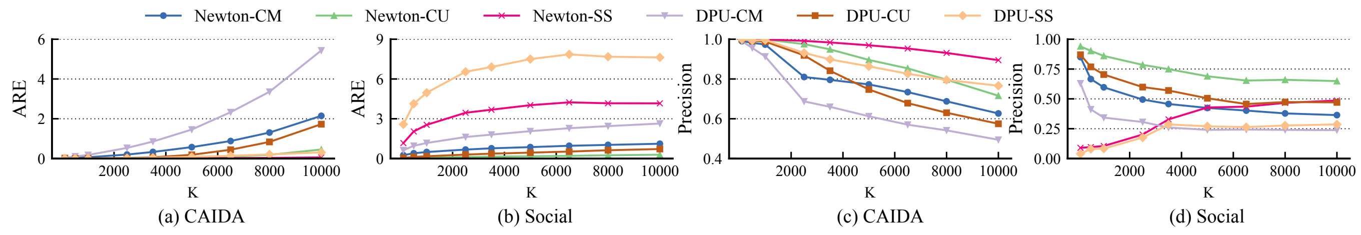


Fig. 10: Global Top-k ARE, Precision vs. k

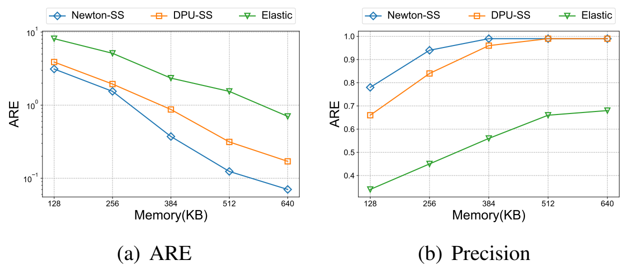


Fig. 11: Lobal Top-k ARE, Precision vs. memory size

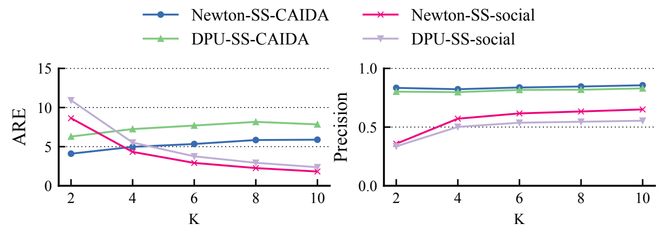


Fig. 12: Lobal Top-k ARE, Precision vs. k

Newton-Observe has a clear optimization effect. However, because Elastic aims to accomplish various tasks with limited memory, its performance in local Top-k tasks is not as good as that of SS, which is specifically designed for this task.

2) ARE, Precision vs. k. In this experiment, we set memory size to $8MB$ and vary the k from 2 to 10. As shown in Figure 12, Newton-Observe improves the precision by up to $1.17\times$ and achieves the smaller ARE by up to $1.45\times$. With the k increasing, the ARE and precision remains almost the same in the CAIDA data set while in the social data set, the ARE decreases and the precision increases. This is because the Intimacy in the social data set is more evenly distributed.

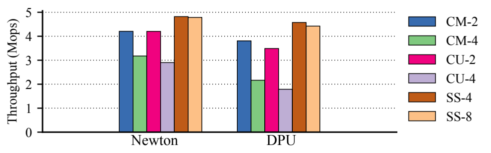


Fig. 13: Throughput vs. # cells or # hash functions

4) Experiments on subgraph query:

In this section, we use the community information provided by the Friendster dataset to test the performance of various algorithms on the subgraph intimacy task. Newton-CM, Newton-CU, and Newton-Elastic all support this task. We set the memory from 32KB to 100KB and measured five communities, taking the average value of the AAE. It can be observed from Figure 14 that the overall trend is consistent with the edge query task.

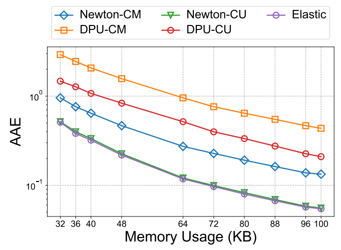


Fig. 14: Subgraph Query AAE vs. memory size

5) Experiments on throughput:

In this section, we evaluate the insertion speed of Newton sketches on the CAIDA data set. Note that the number of cells in the bucket influences the throughput of Newton-SS and the number of hash functions influences the throughput of Newton-CM and Newton-CU, we set $m = 4, 8$ for Space-Saving version (denoted as SS-4, SS-8) and set $l = 2, 4$ for CM version and CU version (denoted as CM-2, CM-4, CU-2 and CU-4) to compare the throughput. As shown in Figure 13, Newton-Observe improves the throughput up to 62% compared with DPU solution. With the number of hash functions increasing, the throughput of Newton-CM and Newton-CU drops by 25% and 31%. With the number of cells in the bucket increasing, the throughput of Newton-SS remains almost the same because either 4 cells or 8 cells can be fit into one cache line.

VII. CONCLUSION

In this paper, we observe that the relationship varying process in dynamic graphs is highly similar to the water cooling process in nature. Based on the observation, we define a new concept, namely Intimacy, using Newton's law of cooling. We propose the technique named Newton-Observe and present Newton sketches using Newton-Observe to solve the typical tasks in dynamic graph queries. The key idea of Newton-Observe is that we can only decay the Intimacy when we actually query it. Experiments show that Newton sketches can achieve well performance with limited memory usage in three typical tasks.

Acknowledgement

We thank all anonymous reviewers for their help in improving this paper. This work is supported by National Key R&D Program of China (No. 2022YFB2901504), and National Natural Science Foundation of China (NSFC) (No. U20A20179, 62372009).

References

[1] Bob Hash source codes. http://burtleburtle.net/bob/hash/evahash.html.

[2] Newton sketches. https://github.com/NewtonSketch/NewtonSketch.

[3] Newton's law of cooling. https://en.wikipedia.org/wiki/Newton%27s_law_of_cooling.

[4] The caida anonymized 2016 internet traces. http://www.caida.org/data/overview/.

[5] An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[6] Discovering hierarchical subgraphs of k-core-truss. *Data Science and Engineering*, 3(2):136–149, June 2018.

[7] D. Bader and K. Madduri. Gtgraph: A synthetic graph generator suite. 2006.

[8] J. W. Berry, B. Hendrickson, R. A. LaViolette, et al. Tolerating the community detection resolution limit with edge weighting. *Phys. Rev. E*, 83:056119, May 2011.

[9] D. Chakrabarti, Y. Zhan, and C. Faloutsos. *R-MAT: A Recursive Model for Graph Mining*, pages 442–446. 2004.

[10] P. Chen, D. Chen, L. Zheng, J. Li, and T. Yang. Out of many we are one: Measuring item batch with clock-sketch. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2021.

[11] F. M. Choudhury, Z. Bao, J. S. Culpepper, and T. Sellis. Monitoring the top-m rank aggregation of spatial objects in streaming queries. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 585–596, 2017.

[12] G. Cormode, T. Johnson, F. Korn, et al. Holistic udafs at streaming speeds. In *SIGMOD*, page 35–46, 2004.

[13] G. Cormode, F. Korn, and S. Tirthapura. Exponentially decayed aggregates on data streams. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, page 1379–1381, USA, 2008. IEEE Computer Society.

[14] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *TOCS*, 21(3):270–313, 2003.

[15] A. Feldmann, A. C. Gilbert, W. Willinger, and T. G. Kurtz. The changing nature of network traffic: Scaling phenomena. *SIGCOMM CCR*, 28(2):5–29, apr 1998.

[16] J. Gao, C. Zhou, J. Zhou, and J. X. Yu. Continuous pattern detection over billion-edge graph using distributed framework. In *2014 IEEE 30th International Conference on Data Engineering*, pages 556–567, 2014.

[17] X. Gou, L. He, Y. Zhang, et al. Sliding sketches: A framework using time zones for data stream processing in sliding windows. In *SIGKDD*, KDD '20, page 1015–1025, New York, NY, USA, 2020. Association for Computing Machinery.

[18] X. Gou and L. Zou. Sliding window-based approximate triangle counting over streaming graphs with duplicate edges. In *SIDMOD*, SIGMOD/PODS '21, page 645–657, New York, NY, USA, 2021. Association for Computing Machinery.

[19] X. Gou, L. Zou, C. Zhao, and T. Yang. Fast and accurate graph stream summarization. In *ICDE*, pages 1118–1129, 2019.

[20] S. Guha and A. McGregor. Graph synopses, sketches, and streams: A survey. *VLDB Endowment*, 5(12):2030–2031, 2012.

[21] Q. Guo, F. Zhuang, C. Qin, et al. A survey on knowledge graph-based recommender systems, 2020.

[22] F. M. Harper and J. A. Konstan. The movielens datasets. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2015.

[23] Y. Izenov, A. Datta, F. Rusu, and J. Shin. Online sketch-based query optimization. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2021.

[24] P. Jia, P. Wang, J. Tao, and X. Guan. A fast sketch method for mining user similarities over fully dynamic graph streams. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1682–1685, 2019.

[25] U. Kang, B. Meeder, E. E. Papalexakis, and C. Faloutsos. Heigen: Spectral analysis for billion-scale graphs. *TKDE*, 26(2):350–362, 2014.

[26] A. Khan and C. Aggarwal. Query-friendly compression of graph streams. In *ASONAM*, page 130–137, 2016.

[27] T. Q. Lee, Y. Park, and Y.-T. Park. A time-based approach to effective recommender systems using implicit feedback. *Expert Systems with Applications*, 34(4):3055–3062, 2008.

[28] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data.

[29] Y. Li, L. Zou, M. T. Özsu, and D. Zhao. Time constrained continuous subgraph search over streaming graphs. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1082–1093, 2019.

[30] M. H. Namaki, K. Sasani, Y. Wu, and T. Ge. Beams: Bounded event detection in graph streams. *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 1387–1388, 2017.

[31] O. Papapetrou, M. Garofalakis, and A. Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *VLDB*, 5(10):992–1003, jun 2012.

[32] S. Raghavan and H. Garcia-Molina. Representing web graphs. In *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*, pages 405–416, 2003.

[33] C. Song and T. Ge. Labeled graph sketches. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1312–1315, 2018.

[34] N. Tang, Q. Chen, and P. Mitra. Graph stream summarization: From big bang to big crunch. In *SIGMOD*, page 1481–1496, 2016.

[35] D. Thomas, R. Bordawekar, C. C. Aggarwal, and P. S. Yu. On efficient query processing of stream counts on the cell processor. In *Proceedings of the International Conference on Data Engineering (ICDE)*, page 748–759, 2009.

[36] D. Ting, J. Malkin, and L. Rhodes. Data sketching for real time analytics: Theory and practice. In *SIGKDD*, 2020.

[37] C. Wang and L. Chen. Continuous subgraph pattern search over graph streams. In *2009 IEEE 25th International Conference on Data Engineering*, pages 393–404, 2009.

[38] P. Wang, P. Jia, X. Zhang, J. Tao, X. Guan, and D. Towsley. Utilizing dynamic properties of sharing bits and registers to estimate user cardinalities over time. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1094–1105, 2019.

[39] P. Wang, Y. Qi, Y. Zhang, and etal. A memory-efficient sketch method for estimating high similarities in streaming sets. In *SIGKDD*, 2019.

[40] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig. Elastic sketch: adaptive and fast network-wide measurements. In *SIGCOMM*, pages 561–575. ACM, 2018.

[41] T. Yang, L. Liu, Y. Yan, M. Shahzad, Y. Shen, X. Li, B. Cui, and G. Xie. Sf-sketch: A fast, accurate, and memory efficient data structure to store frequencies of data items. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 103–106, 2017.

[42] T. Yang, H. Zhang, D. Yang, Y. Huang, and X. Li. Finding significant items in data streams. In *ICDE*, pages 1394–1405, 2019.

[43] Y. Zhang, J. Li, Y. Lei, et al. On-off sketch: A fast and accurate sketch on persistence. *Proc. VLDB Endow.*, 14(2):128–140, oct 2020.

[44] B. Zhao, X. Li, B. Tian, and etal. Dhs: Adaptive memory layout organization of sketch slots for fast and accurate data stream processing. In *SIGKDD*, 2021.

[45] P. Zhao, C. C. Aggarwal, and M. Wang. Gsketch: On query estimation in graph streams. *Proc. VLDB Endow.*, 5(3):193–204, Nov. 2011.

[46] Z. Zhong, S. Yan, Z. Li, D. Tan, T. Yang, and B. Cui. Burstsketch: Finding bursts in data streams. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 2375–2383, 2021.

[47] Y. Zhou, T. Yang, J. Jiang, B. Cui, M. Yu, X. Li, and S. Uhlig. Cold filter: A meta-framework for faster and more accurate stream processing. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, page 741–756, 2018.