Reinforcement Learning Enhanced Weighted Sampling for Accurate Subgraph Counting on Fully Dynamic Graph Streams

Kaixin Wang*, Cheng Long*§, Da Yan†, Jie Zhang*, H. V. Jagadish‡
*School of Computer Science and Engineering, Nanyang Technological University, Singapore
†Department of Computer Science, University of Alabama at Birmingham, United States
‡Computer Science and Engineering, University of Michigan, United States
{kaixin.wang, c.long, zhangj}@ntu.edu.sg, yanda@uab.edu, jag@umich.edu

***Abstract*—As the popularity of graph data increases, there is a growing need to count the occurrences of subgraph patterns of interest, for a variety of applications. Many graphs are massive in scale and also fully dynamic (with insertions and deletions of edges), rendering exact computation of these counts to be infeasible. Common practice is, instead, to use a small set of edges as a sample to estimate the counts. Existing sampling algorithms for fully dynamic graphs sample the edges with uniform probability. In this paper, we show that we can do much better if we sample edges based on their individual properties. Specifically, we propose a weighted sampling algorithm called `WSD` for estimating the subgraph count in a fully dynamic graph stream, which samples the edges based on their weights that indicate their importance and reflect their properties. We determine the weights of edges in a data-driven fashion, using a novel method based on reinforcement learning. We conduct extensive experiments to verify that our technique can produce estimates with smaller errors while often running faster compared with existing algorithms.**

I. Introduction

Graphs have been widely used to represent the data structures in online social networks (e.g., Facebook and Twitter) and Internet applications (e.g., Youtube, blogs, and web pages), where vertices represent individuals or entities and edges represent interactions or connections among them. Counting a certain subgraph pattern (e.g., triangle) on these graphs can help reveal network structure information [1], detect anomalous behavior [2], [3] and identify the interests of users [4], [5]. For example, in social network analysis, a triangle has been proven to be an evidence of the phenomena of homophily [6], [7] (i.e., people tend to make friends with who are similar to them) and transitivity [8], [9] (i.e., people who share common friends become friends). Therefore, many concepts in social network, such as clustering coefficient [10] and transitivity ratio [11], are based on the triangle count. Moreover, [12], [13] show that in web networks, normal nodes usually have similar and mild ratios of the triangle counts to the degrees whereas spammers or harmful accounts are often linked with fewer but remarkably well-connected nodes, resulting in extremely high ratios. Therefore, they propose to detect anomalous nodes and structures in email systems and phone call networks based on the triangle counts and degrees information, and the effectiveness of the proposed method has been verified via case studies [2].

Among the existing studies for estimating the count of a certain subgraph pattern, some focus on insertion-only graph streams [2], [14], [15] and others on fully dynamic graph streams (which involve both insertions and deletions of edges) [16], [3], [17], [18], [19]. In this paper, we study the problem on fully dynamic graph streams since they are more general than insertion-only graph streams and in many of these applications, the graphs are in the form of fully-dynamic graph streams with edges inserted and deleted dynamically. For example, in a social network such as Facebook, connections and disconnections among users would happen as time goes by, which correspond to edge insertions and deletions, respectively.

We observe that the existing studies for fully dynamic graph streams all sample the edges with uniform probabilities, e.g., each edge is treated equally for sampling [3], [16], [17], [18], [19], which would likely result in sub-optimal samples. To illustrate, suppose that we would like to estimate the number of triangles (3-cliques) in a social network like Twitter. There would be more triangles involving two celebrities if they subscribe each other, and thus the edge between them should be sampled with a higher probability than those among the generic public so as to achieve smaller estimation variance.

Motivated by the phenomenon that different edges have different importance and should be assigned with different probabilities for sampling, in this paper, we aim to develop a *weighted* sampling algorithm for estimating the count of a certain subgraph structure on fully dynamic graph streams. In a weighted sampling algorithm, each edge would be assigned with a *weight* indicating the importance of the edge such that edges with higher weights would be sampled with higher probabilities. This immediately gives rise to two problems. First, how can we perform weighted sampling on a fully dynamic graph stream? Second, while we can all intuitively see that more important edges should be weighted higher, what exactly should these weights be? In this paper, we address both

[§]Corresponding author.

these problems in turn.

To address the first problem, we first review an existing weighted sampling framework called `GPS` [14] for insertion-only graph streams. Specifically, `GPS` maintains a reservoir of a fixed size M. Whenever an edge is inserted, it computes a rank of the edge in a probabilistic manner such that it would be more likely that the rank is higher if the edge's weight is higher. It then samples the edge if its rank is among the top-M. However, `GPS` is not capable of dealing with fully-dynamic graph streams since an edge with its rank not among the top-M may be sampled if some space of the reservoir has been released due to some edge deletions that happened earlier. We then present an adaption of `GPS` called `GPS-A`, which works similarly as `GPS` for edge insertions and attaches to each edge to be deleted a binary tag without practically deleting it. `GPS-A` works for fully-dynamic graph streams with both insertions and deletions, but is lazy to clear up the space taken for storing edges that have been deleted in a reservoir, which results in low accuracy. Finally, we introduce a carefully designed weighted sampling framework called *weighted sampling with deletions* (`WSD`) for fully-dynamic graph streams. `WSD` extends `GPS` in a smart and careful way such that it can handle both edge insertions and deletions and avoid the drawback of `GPS-A`, i.e., the edges that have been deleted would not be stored in a reservoir. One intuition behind `WSD` is that it maintains a rank threshold and samples an edge only if its rank exceeds the threshold. The threshold is updated properly such that any two edges with equal weights would be sampled with equal probabilities. We then construct an estimator of the count of a certain subgraph pattern based on the sampled edges in the reservoir by `WSD` and prove the *unbiasedness* of the estimator.

For the second problem, the conventional approach has been to use heuristics to set the weights. For example, in the weighted sampling method `GPS` on insertion-only graph streams, the strategy is to set the weight of an edge to be number of connections between the edge and those in the reservoir or the number of subgraph structures formed by the edge and those in the reservoir. Nevertheless, the heuristic-based methods are non-adaptive to the underlying dynamics. To fully unleash the power of `WSD`, we further propose an adaptive way of setting the weights of edges via reinforcement learning (RL). Specifically, we regard the sampling process for subgraph counting problem as a sequential decision making process, i.e., we make a decision on how to set the weight of each new edge. We then model the sequential decision process as a Markov Decision Process (MDP) [20] and use a policy gradient method to learn a policy for the MDP. We carefully design the MDP including states and rewards such that (1) the states capture both temporal and topological information of the edge dynamics and can be easily computed with the sampled edges, and (2) the objective of maximizing the rewards of the policy is consistent with the goal of minimizing the estimation error.

In summary, the main contributions of this paper are as follows.

- We propose a new fixed-size, weight-sensitive, one-pass sampling method `WSD` to handle fully dynamic graph streams. `WSD` is the first weighted sampling method for fully dynamic graph steams. Based on `WSD`, we construct an estimator of the count of a given subgraph structure and prove the unbiasedness of the estimator. (Section III)
- We develop a reinforcement learning-based method for setting the weights of edges in `WSD` in a data-driven fashion, which is superior over heuristic-based methods. (Section IV)
- We conduct extensive experiments on several real graphs, including community networks, citation graphs, social networks and web graphs to verify that the new sampling framework `WSD` and the RL-based weighting method work better than the state-of-the-art methods. For example, `WSD` improves the effectiveness by 25% - 47% and often runs faster than the state-of-the-art method `WRS` [17], [18]. (Section V)

For the rest of the paper, we present the problem definition in Section II, review the related work in Section VI, and conclude the paper in Section VII.

II. Preliminaries and Problem Definition

We first review the problem of counting subgraphs in a fully dynamic graph stream [16]. Consider a dynamic graph G, which evolves over time with the edges being inserted and deleted dynamically. Formally, the graph G can be modelled as an edge stream $S = \{s^{(1)}, s^{(2)}, \cdots\}$, where $s^{(t)} = (op, e_t)$ represents an event of inserting (indicated by $op = +$) or deleting (indicated by $op = -$) the edge e_t. We denote by $S^{(t)}$ the sequence of the first t edge events, i.e., $S^{(t)} = \{s^{(1)}, s^{(2)}, \cdots, s^{(t)}\} \subset S$. Let $G^{(t)} = (V^{(t)}, E^{(t)})$ be the induced graph from $S^{(t)}$. We assume that all edge events are feasible, i.e., if $e \in E^{(t)}$ (resp. $e \notin E^{(t)}$), $s^{(t+1)}$ cannot be $(+, e)$ (resp. $(-, e)$).

Consider a certain subgraph pattern H (e.g., a triangle). We use $|H|$ to denote the number of edges in H. Given a graph $G^{(t)} = (V^{(t)}, E^{(t)})$, we use the notation $\mathcal{J}^{(t)}$ to denote the set of all subgraphs which are isomorphic to H in $G^{(t)}$. Each subgraph $J \in \mathcal{J}$ can be uniquely identified by a set of ordered edges $J = \{e_{i_1}, e_{i_2}, \cdots, e_{i_{|H|}}\}$ with $i_1 < i_2 < \cdots < i_{|H|}$ being the arrival order of these edges.

Definition 1 (Subgraph Counting in Fully Dynamic Graph Streams [16], [17]). *Given a fully dynamic graph stream S, the problem is to estimate $|\mathcal{J}^{(t)}|$ for any t accurately with the following constraints.*

- ***No Knowledge**. We have no knowledge about the stream (e.g., the size of stream, the number of vertices and edges, etc.) in advance.*
- ***Limited Memory**. We can store at most M edges in a reservoir, where M is a predefined parameter and independent to the size of the stream.*
- ***Single Pass**. Edge insertions and deletions are processed one by one in their arrival order. Edges cannot be accessed again once they are discarded.*

These three constraints are inherited from existing studies [16], [17], on which we elaborate as follows. First, the "no knowledge" constraint naturally holds in many social network analytics applications, e.g., we normally cannot know how many connections would be formed in a social network in the future. Second, the "limited memory" constraint often holds when (1) we aim to achieve real-time responses (which is only possible when a limited number of edges are stored and used for estimating the properties of a graph) and/or (2) the graph is huge (e.g., it is web-scale and cannot fit in main memory in many cases). Third, the "single pass" constraint holds when we have a very high efficiency requirement since scanning a stream multiple times would be costly.

III. Weighted Sampling Frameworks and Subgraph Count Estimators

In this section, we first review an existing weighted sampling framework called GPS [14], which only works for insertion-only graph streams, in Section III-A. We then present a straightforward adaption of GPS called GPS-A, which works similarly as GPS for edge insertions and attaches to each edge to be deleted a binary tag without practically deleting it, in Section III-B. GPS-A works for fully-dynamic graph streams with both insertions and deletions, but is lazy to clear up the space taken for storing edges that have been deleted in a reservoir, which results in low accuracy. Finally, we introduce a carefully designed weighted sampling framework called *weighted sampling with deletions* (WSD) for fully-dynamic graph streams in Section III-C. WSD extends GPS in a smart and careful way such that it can handle both edge insertions and deletions and avoid the drawback of GPS-A, i.e., the edges that have been deleted would not be stored in a reservoir.

A. GPS Framework

Sampling Process. GPS framework follows the *priority sampling scheme* [21] and aims to sample a fixed-size reservoir of edges via a single pass such that edges that are deemed more important are sampled with higher probabilities. Specifically, when handling an event $(+, e_t)$ at time step t, it involves three steps. First, it assigns the edge e_t an appropriate *weight* denoted by $w(e_t)$, based on the current reservoir $\mathcal{R}^{(t-1)}$. For example, it sets the weight to be the number of subgraph structures (e.g., triangles) that would be newly formed by e_t as an indicator of the importance of e_t - the larger the number is, the higher the weight is [14]. We denote by $W(e, \mathcal{R})$ the function that computes the weight of an edge e based on a reservoir $\mathcal{R}$. Second, it samples a value u from $(0, 1]$ uniformly and computes a *rank* for the edge e_t, denoted by $r(e_t)$, based on $w(e_t)$ and u. For an edge with a higher weight, its rank would be likely higher - the randomness here is due to the sampling process of the value u. We denote by $r = f(w)$ the function that computes the rank of an edge with the weight w. For example, $f(w) = w/u$ is used as the rank function in [14], where u is a random value uniformly sampled in range $(0, 1]$. Third, it includes the edge e_t in the reservoir if either (1) the reservoir is not full or (2) the rank of e_t is larger than the smallest rank of an edge in the reservoir (in this case the edge with the smallest rank in the reservoir would be dropped due to the capacity limit of the reservoir); otherwise, it does not include the edge e_t in the reservoir. For this step, it uses a minimum priority queue with the size equal to M and the keys to be the ranks of edges in the queue.

With GPS, at the end of time step t, for each edge e that has been inserted, the probability that edge e is included in the reservoir, i.e., $\mathcal{R}^{(t)}$, is equal to the probability that e's rank, i.e., $r(e)$, is larger than the $(M+1)^{th}$ largest rank among those of edges that have been inserted (including e), which we denote by r_{M+1}. For $t \le M$, we define r_{M+1} to be equal to 0. Specifically, we have the following equation.

$$\mathbb{P}[e \in \mathcal{R}^{(t)}] = \mathbb{P}[r(e) > r_{M+1}] \quad (1)$$

Note that this probability depends on the rank function. For example in [14], $r = f(w) = w/u$, then $\mathbb{P}[r > r_{M+1}] = \min\{1, w/r_{M+1}\}$. The above equation has been proven in [14]. Here, we provide some intuitive explanations. For $t \le M$, each edge that has been inserted would be included in the reservoir for sure, and thus the equation holds. For $t > M$, the reservoir stores those edges with the M^{th} largest ranks, and thus the probability that an edge to be included in the reservoir should be equal to the probability that its rank is larger than the $(M+1)^{th}$ largest rank of the edges that have been inserted.

Estimator and Analysis. It has been proven in [14] that the probability that a set of edges $E = \{e_1, \cdots, e_{|E|}\}$ $(|E| \le M)$ is included in the reservoir $\mathcal{R}$ at the end of time t is as follows.

$$\mathbb{P}[E \subset \mathcal{R}] = \prod_{e \in E} \mathbb{P}[e \in \mathcal{R}] = \prod_{e \in E} \mathbb{P}[r(e) > r_{M+1}] \quad (2)$$

where r_{M+1} is observed at t. Let $J = \{e_{i_1}, e_{i_2}, \cdots, e_{i_{|H|}}\}$ be a subgraph pattern formed at time $t_a(J)$. Note that $t_a(J)$ is the time at which the last edge of J appears, i.e., $t_a(J) = i_{|H|}$. We define a random variable $X^J_{\texttt{GPS}}$ for J as follows.

$$X^J_{\texttt{GPS}} = \prod_{e \in J \setminus e_{i_{|H|}}} \frac{\mathbb{I}(e \in \mathcal{R})}{\mathbb{P}[r(e) > r_{M+1}]} \quad (3)$$

where $\mathbb{I}(\cdot)$ is an indicator function, and r_{M+1} and $\mathcal{R}$ are the $(M+1)^{th}$ largest rank and the reservoir observed just after time $t_a(J)-1$. We define an estimator of the count of subgraph structures at any time t, denoted by $c^{(t)}_{\texttt{GPS}}$, by the following equation.

$$c^{(t)}_{\texttt{GPS}} = \sum_{J \in \mathcal{A}^{(t)}} X^J_{\texttt{GPS}} \quad (4)$$

where $\mathcal{A}^{(t)}$ is set of subgraphs which are isomorphic to H and have been added to the graph $G^{(t)}$ by time t. In [14], it has been shown that the above estimator $c^{(t)}_{\texttt{GPS}}$ is *unbiased*.

Theorem 1 (Unbiasedness of the GPS estimator [14]). *Given the graph stream S, which only consists of edge insertion events, and $M \ge |H|$, $\forall t$, we have*

$$\mathbb{E}[c^{(t)}_{GPS}] = |\mathcal{A}^{(t)}| = |\mathcal{J}^{(t)}|. \quad (5)$$

Inapplicability of GPS for Fully Dynamic Graph Streams. Unfortunately, GPS cannot be applied to fully dynamic graph streams, which involve both edge insertions and edge deletions. The reason is as follows. The correctness of GPS relies on the fact it guarantees that the edges with equal weights would be included in the reservoir with the equal probabilities (as shown in Eq. (1)). However, this would no longer be guaranteed when GPS is applied to fully dynamic graph streams directly. To illustrate, consider a scenario below.

Example 1. *Consider an edge stream where (1) all edges are assigned with equal weights, (2) at time t ($t > M + 1$) the first event of an edge deletion, i.e., $(-, e_t)$, happens, and (3) at time $t' = t+1$, an event of edge insertion $(+, e_{t'})$ happens. Let p and p' be the probability that an edge is included in the reservoir at time t and t', respectively. Obviously, we have $0 < p' \leq p < 1$ since the probability that an edge is included in the reservoir cannot be increasing. Note that probabilities p and p' are shared by all edges since they have equal weights. Let p'' be the probability that the edge $e_{t'}$ is inserted into the reservoir when the reservoir is full at time t'. Note that p'' is different from p'. We then deduce that for edge $e_{t'}$, the probability that it is included in the reservoir should be equal to $p \cdot 1 + (1-p) \cdot p''$. For the term $p \cdot 1$, it corresponds to the case that e_t has been included in the reservoir and then deleted from the reservoir at time t (and thus the probability for this case is equal to p), and in this case, $e_{t'}$ would be included in the reservoir for sure (i.e., with the probability 1) since (1) the reservoir would be not full when inserting $e_{t'}$, and (2) GPS would unconditionally include an edge when the reservoir is not full. For the term $(1-p) \cdot p''$, it corresponds to the case that e_t has not been included in the reservoir at time t (and thus the probability for this case is equal to $(1-p)$), and in this case, $e_{t'}$ would be included in the reservoir with probability p'' (by definition of p''). In conclusion, the probability that $e_{t'}$ would be included in the reservoir at time t' is larger than that for other edges since $p \cdot 1 + (1-p) \cdot p'' > p \geq p'$ (though all edges are assigned with equal weights), which implies that GPS would fail in this scenario.*

B. GPS-A Framework

Sampling Process. In GPS-A, the sampling process is exactly the same as that of GPS except that when a deletion event happens on an edge in the reservoir, we only attach a "DEL" tag to the edge, but we do not remove the edge from the reservoir. Equivalently speaking, we *ignore* the deletion operations during the sampling process first (by attaching the tags to edges only), and when constructing the estimator, we would *neglect* those edges with the tags. In this way, the probabilities of including the edges with equal weights in the reservoir would be equal, in the same way as GPS. The drawback of GPS-A is that some space of the reservoir, which is taken by the edges with the "DEL" tags and can be used for including other edges otherwise, would be wasted.

Estimator and Analysis. Since GPS-A simply attaches a tag to each of the deleted edges, Eq. (2) also holds for GPS-A. We denote by $\mathcal{R}_{tag}$ the set of edges with the "DEL" tags in the reservoir $\mathcal{R}$. Let $J = \{e_{i_1}, e_{i_2}, \cdots, e_{i_{|H|}}\}$ be a subgraph formed at time $t_a(J)$. Note that $t_a(J)$ is the time at which the last edge of J appears, i.e., $t_a(J) = i_{|H|}$. We define a random variable $X^J_{\texttt{GPS-A}}$ for J as follows.

$$X^J_{\texttt{GPS-A}} = \prod_{e \in J \setminus e_{i_{|H|}}} \frac{\mathbb{I}(e \in \mathcal{R} \setminus \mathcal{R}_{tag})}{\mathbb{P}[r(e) > r_{M+1}]} \tag{6}$$

where $\mathbb{I}(\cdot)$ is an indicator function, $\mathcal{R}$ and r_{M+1} are the reservoir and the $(M+1)^{th}$ largest rank of an edge among those that have appeared, as observed just after time $t_a(J)-1$. Assume that a subgraph J is destroyed at time $t_d(J)$ when a deletion event happens on an edge e_x of J. We define another random variable $Y^J_{\texttt{GPS-A}}$ for J as follows.

$$Y^J_{\texttt{GPS-A}} = \prod_{e \in J \setminus e_x} \frac{\mathbb{I}(e \in \mathcal{R} \setminus \mathcal{R}_{tag})}{\mathbb{P}[r(e) > r_{M+1}]} \tag{7}$$

where $\mathcal{R}$ and r_{M+1} are the reservoir and the $(M+1)^{th}$ largest rank of an edge among those that have appeared, as observed just after time $t_d(J)-1$. Based on these two subgraph estimators, we define an estimator of the count of subgraph structures at any time t, denoted by $c^{(t)}_{\texttt{GPS-A}}$, as follows.

$$c^{(t)}_{\texttt{GPS-A}} = \sum_{J \in \mathcal{A}^{(t)}} X^J_{\texttt{GPS-A}} - \sum_{J \in \mathcal{D}^{(t)}} Y^J_{\texttt{GPS-A}} \tag{8}$$

where $\mathcal{A}^{(t)}$ (resp. $\mathcal{D}^{(t)}$) is set of subgraphs which are isomorphic to H and have been added to (resp. deleted from) the graph $G^{(t)}$ by time t.

Theorem 2 (Unbiasedness of the subgraph count estimator of GPS-A)**.** *Given the graph stream S and $M \geq |H|$, $\forall t$, we have*

$$\mathbb{E}[c^{(t)}_{\texttt{GPS-A}}] = |\mathcal{A}^{(t)}| - |\mathcal{D}^{(t)}| = |\mathcal{J}^{(t)}|. \tag{9}$$

The unbiasedness of $c^{(t)}_{\texttt{GPS-A}}$ can be verified based on Eq. (2). Detailed proof can be found in the technical report [22].

Theorem 3 (Complexities of GPS-A)**.** *The time and space complexity of GPS-A is $O((|A| + |D|) \cdot \log M \cdot \gamma(M))$ and $O(M)$, respectively, where $|A|$ (resp. $|D|$) is the number of insertion (resp. deletion) events in the stream, M is the maximum size of the reservoir, and $\gamma(M)$ is the time complexity of enumerating the subgraphs formed by the sampled edges.*

Proof. The time complexity is as follows. Consider an event of inserting an edge $e = (u, v)$. Finding the subgraphs formed by e and some sampled edges would cost $\gamma(M)$. For example, for triangle counting, $\gamma(M) = O(|\mathcal{N}(u)| + |\mathcal{N}(v)|) = O(M)$ (which corresponds to the cost of computing the intersection between $\mathcal{N}(u)$ and $\mathcal{N}(v)$), where $\mathcal{N}(u)$ (resp. $\mathcal{N}(v)$) is the set of the neighbors of u (resp. v) in the sampled graph. Then, after calculating e's weight and rank, it takes $O(\log M)$ time to add the edge into a priority queue if it is included in the reservoir. Consider an event of edge deletion. Similarly, finding

Algorithm 1: `WSD` Framework

```
Input: An edge event stream $S = \{s^{(1)}, s^{(2)}, \cdots\}$.
Output: A reservoir $\mathcal{R}$ of sampled items.
Let $\mathcal{R}$ be a min-priority queue with the maximum size $M$
$\mathcal{R} \leftarrow \Phi$, $\tau_p \leftarrow 0$, $\tau_q \leftarrow 0$
foreach $(op, e_t) \in S$ do
    if $op = +$ then
        insert($e_t$)
    else
        delete($e_t$)
function insert($e$)                    // Case 1 & Case 2
    $w(e) \leftarrow W(e, \mathcal{R})$
    $r(e) \leftarrow f(w(e))$
    if $|\mathcal{R}| < M$ then
        if $r(e) > \tau_p$ then                // Case 1.1
            $\mathcal{R} \leftarrow \mathcal{R} \cup e$
    else
        $e_m \leftarrow$ the edge with the minimum rank in $\mathcal{R}$
        $\tau_p \leftarrow r(e_m)$
        if $r(e) > \tau_p$ then                // Case 2.1
            $\mathcal{R} \leftarrow (\mathcal{R} \setminus e_m) \cup e$
            $\tau_q \leftarrow \tau_p$
        else if $r(e) > \tau_q$ then           // Case 2.2
            $\tau_q \leftarrow r(e)$
function delete($e$)                           // Case 3
    if $e \in \mathcal{R}$ then
        $\mathcal{R} \leftarrow \mathcal{R} \setminus e$
```

the subgraphs that are destroyed with this edge deletion would cost $\gamma(M)$. Then, we need to identify the position of the edge in the reservoir and assign it the tag if it is sampled, which costs $O(\log M)$. Thus, the total time cost is $O((|A| + |D|) \cdot \log M \cdot \gamma(M))$. The space complexity is $O(M)$ since we can store at most M edges in the reservoir. □

Drawbacks of `GPS-A`. As mentioned before, `GPS-A` has an intrinsic drawback, i.e., those edges that have been included in the reservoir and then deleted would occupy some of storage of the reservoir without any benefits (for estimating the count of the subgraph structures). As the sampling process goes on, the reservoir would become smaller and smaller practically, which would result in low accuracy. Furthermore, it requires some extra space for storing the "DEL" tags.

C. `WSD` *Framework*

Sampling Process. In `WSD`, we also use a min-priority queue with a fixed size of M for storing the sampled edges. To avoid the drawbacks of `GPS-A`, when the deletion of an edge happens, we directly remove the edge from the reservoir if it exists in the reservoir. To ensure the correctness, we maintain two variables, namely τ_p and τ_q, throughout the sampling process.

- **τ_p.** We use it as a rank threshold such that for each insertion event $(+, e_t)$ at time t, we sample edge e_t only if e_t's rank is larger than τ_p.
- **τ_q.** It is a rank value for computing the probability with which an edge is sampled to the reservoir at the end of time t.

We maintain τ_q and τ_p appropriately such that at the end of time t, for each edge e that has been inserted by time t (inclusively) and not deleted yet, the probability e is sampled in the reservoir is equal to the probability that e's rank is larger than τ_q, i.e.,

$$\mathbb{P}[e \in \mathcal{R}^{(t)}] = \mathbb{P}[r(e) > \tau_q] \tag{10}$$

Similarly, this probability depends on the rank function. If we adopt $r = f(w) = w/u$, then $\mathbb{P}[r > \tau_q] = \min\{1, w/\tau_q\}$. We will prove Eq. (10) formally later in Lemma 1. We note that τ_p (resp. τ_q) is not always equal to the M^{th} (resp. $(M+1)^{th}$) rank of those edges that have been inserted, which reflects one of the differences between `WSD` and `GPS-A`.

Specifically, we present the sampling process of `WSD` as follows. First, we initialize the queue to be empty and both variables τ_p and τ_q to be 0. Then, we present the sampling process in the following cases.

- **Case 1: For an event $(+, e_t)$ with a *non-full* reservoir.** We (1) assign e_t a weight, $w(e_t)$, based on the reservoir and (2) compute the rank of edge e_t, i.e., $r(e_t)$, based on $w(e_t)$ and a random value u randomly sampled in $(0, 1]$.
  - **Case 1.1:** $r(e_t) > \tau_p$**.** We include edge e_t in the reservoir.
  - **Case 1.2:** $r(e_t) \leq \tau_p$**.** We discard edge e_t.
- **Case 2: For an event $(+, e_t)$ with a *full* reservoir.** We do (1) and (2) as in Case 1, and (3) update τ_p to be minimum rank of the edges in the reservoir.
  - **Case 2.1:** $r(e_t) > \tau_p$**.** We exclude the edge with the minimum rank, include edge e_t in the reservoir, and update τ_q to be τ_p.
  - **Case 2.2:** $\tau_q < r(e_t) \leq \tau_p$**.** We discard edge e_t, and update τ_q to be $r(e_t)$.
  - **Case 2.3:** $r(e_t) \leq \tau_q$**.** We discard the edge e_t.
- **Case 3: For an event $(-, e_t)$.** We drop e_t from the reservoir if e_t has been sampled before and do nothing otherwise.

The pseudo-code of `WSD` is presented in Algorithm 1. We explain some of the intuitions behind the sampling process of `WSD`. First, in Case 1, we do not update τ_p or τ_q, which we explain as follows. In this case, the reservoir is not full, and thus no edges that have been included in the reservoir would be dropped. This implies that the probabilities of including these edges in the reservoir would not be changed, which further implies that τ_q should be retained since we aim to use τ_q for computing these probabilities, as shown in Eq. (10). Correspondingly, the rank threshold τ_p which has been used for sampling previous edges should be retained.

To see this, we consider the same scenario described in Section III-A. Let p be the probability that an edge is included in the reservoir at time t. Let τ be the value of τ_p observed at $t-1$. Let τ' (resp. τ'') be the value of τ_p observed at t' if the reservoir is non-full (resp. full). Let $p(\tau)$, $p(\tau')$ and $p(\tau'')$ be the probabilities that an edge's rank is no less than τ, τ' and τ'', respectively. Consider an edge e before $e_{t'}$. The probability that e is included in the reservoir at t' is $p \cdot p(\tau) + (1-p) \cdot p(\tau'')$. For the term $p \cdot p(\tau)$, it corresponds to case that e_t has been included in the reservoir at t (which has the probability of p), and $p(\tau)$ corresponds to the conditional probability that e is in the reservoir at $t-1$ (in this case, e would be in the reservoir at t' for sure since the reservoir is not full at t'). For the term $(1-p) \cdot p(\tau'')$, it corresponds the case that e_t has not been included in the reservoir at t (which has the probability of $(1-p)$, and in this case the reservoir is full at the beginning of t' and $p(\tau'')$ corresponds to the conditional probability that e is in the reservoir at t'. Consider the edge $e_{t'}$. The probability that $e_{t'}$ is included in the reservoir at t' is $p \cdot p(\tau') + (1-p) \cdot p(\tau'')$, which can be verified similarly. In order to guarantee that the probabilities for e and $e_{t'}$ to be included are equal, it is necessary to have $\tau' = \tau$ (i.e., τ_p is retained).

Second, in Case 2, we update τ_p to be the minimum rank of edges in the reservoir since the reservoir is full and for any edge e_t to be included in the reservoir, its rank $r(e_t)$ should be larger than the minimum rank of existing edges in the reservoir. In addition, depending on the rank of e_t, we update the reservoir and τ_p correspondingly. In particular, in Case 2.1 and Case 2.2, we update τ_p appropriately so as to make sure Eq. (10) holds.

Third, in Case 3, we do not update τ_p or τ_q since after an edge is deleted, the probability that a remaining edge that has been inserted is included in the reservoir would not be affected.

Lemma 1. *In WSD, at the end of a time t, the probability that an edge e is sampled in the reservoir, i.e., $\mathbb{P}[e \in \mathcal{R}^{(t)}]$, is equal to $\mathbb{P}[r(e) > \tau_q]$, where τ_q is observed at time t.*

Proof. We prove this lemma by deduction. When $t \leq M$, τ_q is always 0. Each edge would be included into reservoir for sure, which implies that Eq. (10) holds. Assume that Eq. (10) holds for $t = k$ $(k \geq M)$, i.e., $\mathbb{P}[e \in \mathcal{R}^{(k)}] = \mathbb{P}[r(e) > \tau_q^{(k)}]$, where $\mathcal{R}^{(k)}$ and $\tau_q^{(k)}$ correspond to $\mathcal{R}$ and τ_q as observed at time k, respectively. Consider $t = k+1$. There are three possible cases of how the reservoir may change (cases as defined before).

- **Case 1.** Consider an edge e (before e_{k+1}). Since (1) no edges that have been included in the reservoir would be dropped, which implies that the probabilities of including these edges in the reservoir would not be changed, and (2) we hold τ_q, we have $\mathbb{P}[e \in \mathcal{R}^{(k+1)}] = \mathbb{P}[e \in \mathcal{R}^{(k)}] = \mathbb{P}[r(e) > \tau_q^{(k)}] = \mathbb{P}[r(e) > \tau_q^{(k+1)}]$. Consider e_{k+1}. Since we hold τ_p, we would sample e_{k+1} with the exactly the same probability as we sample those edges before e_{k+1} with equal weights. Therefore, Eq. (10) holds.
- **Case 2.** Consider e_{k+1}. We compare $r(e_{k+1})$ with τ_p, which corresponds to the minimum rank of the edges in $\mathcal{R}^{(k)}$. In case that e_{k+1} is included into the reservoir, we update $\tau_q^{(k+1)}$ to τ_p. Therefore, at the end of time $t = k+1$, $\mathbb{P}[e_{k+1} \in \mathcal{R}^{(k+1)}] = \mathbb{P}[r(e_{k+1}) > \tau_p] = \mathbb{P}[r(e_{k+1}) > \tau_q^{(k+1)}]$. Consider an edge e (before e_{k+1}). It can be verified that at the end of time t, e is included in the reservoir only if its rank is larger than τ_q, i.e., $\mathbb{P}[e \in \mathcal{R}^{(k+1)}] = \mathbb{P}[r(e) > \tau_q]$. Thus, Eq. (10) holds.
- **Case 3.** Since (1) the deletion of e_{k+1} would not affect other edges' probabilities, and (2) we do not update τ_q. Consider an edge e (before e_{k+1}). We have $\mathbb{P}[e \in \mathcal{R}^{(k+1)}] = \mathbb{P}[e \in \mathcal{R}^{(k)}] = \mathbb{P}[r(e) > \tau_q^{(k)}] = \mathbb{P}[r(e) > \tau_q^{(k+1)}]$. That is, Eq. (10) holds.

Therefore, $\forall t$, Eq (10) holds. □

Estimator and Analysis. Let $J = \{e_{i_1}, e_{i_2}, \cdots, e_{i_{|H|}}\}$ be a subgraph formed at time $t_a(J)$. Note that $t_a(J)$ is the time at which the last edge of J appears, i.e., $t_a(J) = i_{|H|}$. We define a random variable $X^J_{\texttt{WSD}}$ for J as follows.

$$X^J_{\texttt{WSD}} = \prod_{e \in J \setminus e_{i_{|H|}}} \frac{\mathbb{I}(e \in \mathcal{R})}{\mathbb{P}[r(e) > \tau_q]} \tag{11}$$

where $\mathcal{R}$ and τ_q are the reservoir and the rank value, as observed just after time $t_a(J) - 1$. Assume that a subgraph J is destroyed at time $t_d(J)$ when a deletion event happens on an edge e_x of J. We define another random variable $Y^J_{\texttt{WSD}}$ for J as follows.

$$Y^J_{\texttt{WSD}} = \prod_{e \in J \setminus e_x} \frac{\mathbb{I}(e \in \mathcal{R})}{\mathbb{P}[r(e) > \tau_q]} \tag{12}$$

where $\mathcal{R}$ and τ_q are the reservoir and the rank value, as observed just after time $t_d(J) - 1$. Based on these two subgraph estimators, we define an estimator of the count of subgraph structures at any time t as follows.

$$c^{(t)}_{\texttt{WSD}} = \sum_{J \in \mathcal{A}^{(t)}} X^J_{\texttt{WSD}} - \sum_{J \in \mathcal{D}^{(t)}} Y^J_{\texttt{WSD}} \tag{13}$$

where $\mathcal{A}^{(t)}$ (resp. $\mathcal{D}^{(t)}$) is set of subgraphs which are isomorphic to H and have been added to (resp. deleted from) the graph $G^{(t)}$ by time t.

Theorem 4 (Unbiasedness of the subgraph count estimator of WSD)**.** *Given the graph stream S and $M \geq |H|$, $\forall t$, we have*

$$\mathbb{E}[c^{(t)}_{WSD}] = |\mathcal{J}^{(t)}|. \tag{14}$$

We provide a sketch of the proof here and put the details in the technical report [22]. We first prove by deduction that at the end of time t for all cases, the probability that a set of edges $E = \{e_1, \cdots, e_{|E|}\}$ $(|E| \leq M)$ is included in $\mathcal{R}^{(t)}$ is

$$\mathbb{P}[E \subset \mathcal{R}^{(t)}] = \prod_{e \in E} \mathbb{P}[e \in \mathcal{R}^{(t)}] = \prod_{e \in E} \mathbb{P}[r(e) > \tau_q], \tag{15}$$

where τ_q is observed at time t. Then, we prove the following two statements based on Eq (15).

$$\mathbb{E}[X^J_{\texttt{WSD}}] = 1, \quad \mathbb{E}[Y^J_{\texttt{WSD}}] = 1. \tag{16}$$

Algorithm 2: Subgraph Counting with `WSD`

```
Input: An edge event stream $S$ and a subgraph pattern $H$.
Output: Estimated subgraph count $c$.
1  Let $\mathcal{R}$ be a priority queue with the maximum size $M$
2  $\mathcal{R} \leftarrow \Phi$, $\tau_p \leftarrow 0$, $\tau_q \leftarrow 0$, $c \leftarrow 0$
3  foreach $(op, e_t) \in S$ do
4      $\mathcal{H} \leftarrow \{J \subset (\mathcal{R} \cup e_t) \mid e_t \in J, J \cong H\}$
5      foreach $J \in \mathcal{H}$ do
6          if $op = +$ then
7              $c \leftarrow c + \prod_{e \in J \setminus e_t} \frac{1}{\mathbb{P}[r(e) > \tau_q]}$
8          else
9              $c \leftarrow c - \prod_{e \in J \setminus e_t} \frac{1}{\mathbb{P}[r(e) > \tau_q]}$
10     if $op = +$ then
11         insert($e_t$)
12     else
13         delete($e_t$)
14 return $c$
```

Finally, based on the linearity of expectation, we have

$$\begin{aligned}\mathbb{E}[c_{\texttt{WSD}}^{(t)}] &= \sum_{J \in \mathcal{A}^{(t)}} \mathbb{E}[X_{\texttt{WSD}}^{J}] - \sum_{J \in \mathcal{D}^{(t)}} \mathbb{E}[Y_{\texttt{WSD}}^{J}] \\ &= \sum_{J \in \mathcal{A}^{(t)}} 1 - \sum_{J \in \mathcal{D}^{(t)}} 1 = |\mathcal{A}^{(t)}| - |\mathcal{D}^{(t)}| = |\mathcal{J}^{(t)}|.\end{aligned} \tag{17}$$

Implementation of the estimator. The subgraph counting on fully dynamic graph streams with `WSD` is shown in Algorithm 2. We initialize the counter c as 0 at the beginning. When an edge insertion (resp. deletion) event happens, we check whether the edge forms some subgraph structures with the sampled edges. If yes, we add (resp. subtract) the product of the inverse probabilities of these sampled edges, as defined in Eq. (11) (resp. Eq. (12)), to (resp. from) the counter. Then we update the reservoir based on `insert` (resp. `delete`) function defined in Algorithm 1.

Theorem 5 (Complexities of `WSD`)**.** *The time and space complexity of* `WSD` *is* $O((|A|+|D|)\cdot \log M \cdot \gamma(M))$ *and* $O(M)$*, respectively, where all notations have the same meanings as those in Theorem 3.*

Proof. They can be verified similarly as those of `GPS-A`. □

Remarks. (1) `WSD` v.s. `GPS-A`. While they have the same time complexity, `WSD` would run slightly faster than `GPS-A` in practice. This is because the reservoir of `GPS-A` is always full when $t \geq M$ while the reservoir of `WSD` could be smaller than M after some deletion events happen. For subsequent insertion events, it would take more time to add an edge to the reservoir of `GPS-A` than to that of `WSD`. (2) `WSD` v.s. Existing algorithms `Triest` [16], `ThinkD` [19] and `WRS` [17]. Existing algorithms all have a time complexity of $O((|A|+|D|\cdot M)\cdot \gamma(M))$. Thus, when the number of deletions satisfies

$$|D| > \frac{\log M - 1}{M - \log M} \cdot |A| \approx \frac{\log M}{M} \cdot |A|, \tag{18}$$

`WSD` would run faster than existing algorithms.

IV. Reinforcement Learning based Weight Function

Recall that in the `WSD` algorithm, one step is to set the weight of an edge e when it is inserted given the current reservoir $\mathcal{R}$. This step is modelled as a weight function $W(e, \mathcal{R})$. Due to the online nature and the one-pass sampling setting, the weights cannot be set optimally for optimizing some objective (e.g., minimizing the variance of the estimator). A common practice is to set the weights using some heuristics. For example, one heuristic is to use the number of subgraph structures that would be newly formed by the new edge as an indicator of the importance of the edge - the larger the number is, the higher the weight is [14]. While this heuristic is intuitive enough, it cannot be adaptive to different datasets and/or the underlying dynamics of edges. According to our experimental results, this heuristic based cannot work stably when the order of edge insertions/deletions is changed.

We observe that whenever an edge is inserted, `WSD` needs to decide the weight of the edge. In addition, the weights decided for different edges would collectively affect the sampling process (i.e., the probabilities that the edges are sampled). This naturally triggers us to think of the reinforcement learning (RL) for this task of deciding the weights during the sampling process. Note that RL is well known for effectively making sequential decisions so as to optimize an objective that is collectively affected by the decisions made at each step [20]. Next, we present our RL based method for this task by formalizing it as an Markov decision process (MDP) [20] in Section IV-A and then presenting the algorithm for learning the policy based on the MDP (which would be then used for deciding the weights of edges during the sampling process) in Section IV-B.

A. Weight Function Modeled as an MDP

Let $t_1, t_2, \cdots$ be the time steps, at which the edge insertion events happen. We define an MDP, which consists of (1) states, (2) actions, (3) transitions and (4) rewards, as follows.

(1) States. We denote the state at time t_k by s_k. Intuitively, the state s_k should capture essential information of the edge in the whole graph $G^{(t_k)}$. Since we have no access to the edges which have not been sampled in the reservoir, we rely on the edges in the reservoir $\mathcal{R}$ and the new edge e. We identify two types of information, namely the topological information and the temporal information, for defining the state s_k.

To capture the topological information of the new edge $e = (u, v)$, we take the number of subgraph patterns which are newly formed by e and some edges sampled in the reservoir, denoted by $|\mathcal{H}_k|$ (which is computed in line 4 of Algorithm 2). Besides, we take the number of the neighbors of u (resp. v)

in the sampled graph, denoted by $|\mathcal{N}_k(u)|$ (resp. $|\mathcal{N}_k(v)|$). We concatenate these values together and form a vector, denoted by s_k^g, that is,

$$s_k^g = [|\mathcal{H}_k|, |\mathcal{N}_k(u)|, |\mathcal{N}_k(v)|]. \tag{19}$$

The rationale of s_k^g is as follows. The number of the subgraphs $|\mathcal{H}_k|$ indicates the importance of the edge e to some extent - if the edge e can form more subgraphs in the sampled graph, it has a larger probability to form more subgraphs in the complete graph. The numbers of neighbors ($|\mathcal{N}_k(u)|$ and $|\mathcal{N}_k(v)|$) indicate the potential of the edge e for forming subgraphs in the future. For example, in the example discussed in Section I, famous people can form more triangles than the general public since they have more connections with others. For other subgraphs, similarly, the edges that are adjacent to more others tend to have higher chance to form subgraphs.

To capture the temporal information of the new edge $e = (u, v)$, we consider all subgraphs that are formed by the edge e, i.e., $\mathcal{H}_k$. For each such subgraph, it is associated with an ordered set of edges, i.e., $J = \{e_{i_1}, e_{i_2}, \cdots, e_{i_{|H|}}\}$, where $i_1 < i_2 < \cdots < i_{|H|} = t_k$. Note that $e_{i_{|H|}} = e$. Then for each entry $j \in [1, |H|]$, we take the maximum value i_j among these $|\mathcal{H}_k|$ subgraphs and denote it by v_j, that is,

$$v_j = \max\{i_j \mid e_{i_j} \in J, J \in \mathcal{H}_k\}. \tag{20}$$

Finally, we concatenate these values v_j for $j \in [1, |H|]$ and form a vector, denoted by s_k^v, that is

$$s_k^v = [v_1, v_2, \cdots, v_{|H|}]. \tag{21}$$

The rationale of s_k^v is that it provides the temporal information related to edge e. For example, if the elements of s_k^v are large, we know that the subgraphs are more likely to be formed with recent edges, and thus sampling edge e with a larger probability may help form more subgraphs in the near future.

In summary, we define the state s_k to be a $(|H| + 3)$-dimension vector, which captures both topological and temporal information, as follows.

$$s_k = [s_k^g, s_k^v] \in \mathbb{R}^{|H|+3}. \tag{22}$$

(2) Actions. We denote the action at time t_k by a_k. Recall that at time t_k, the state is s_k, and we need to assign a weight for the edge e to represent its importance. Formally, we define a_k as follows.

$$a_k = w \quad (w > 0), \tag{23}$$

which means we assign a positive real weight w to the edge, i.e., $W(e, \mathcal{R}) = w$.

(3) Transitions. Let s_k be a state and suppose that we take an action $a_k = w$, which assigns the edge e a weight w. Then the `WSD` algorithm would proceed with the weight of e until a new edge insertion arrives. Then, we arrive a new state and compute the state s_{k+1} accordingly.

(4) Rewards. We first define the error between the estimation and the ground truth at time t, denoted by $\epsilon(t)$, as follows.

$$\epsilon(t) = |c^{(t)} - |\mathcal{J}^{(t)}||, \tag{24}$$

where $c^{(t)}$ is the value of the subgraph estimation and $|\mathcal{J}^{(t)}|$ is the value of the ground truth of the subgraph counts. Consider that we take action a_k at a state s_k and then we arrive at a new state s_{k+1}. We define the reward associated with this transition from s_k to s_{k+1}, denoted by r_k, as follows,

$$r_k = \epsilon(t_k) - \epsilon(t_{k+1}), \tag{25}$$

The intuition is that if the error between the estimation and the ground truth resulted from the reservoir after processing the edges between t_k and t_{k+1}, i.e., $\epsilon(t_{k+1})$, is smaller, then the reward is larger. It is worthy of noting that with the reward defined as above, the objective of the MDP, which is to maximize the accumulative rewards, would be equivalent to that of the subgraph counting problem, which is to minimize the estimation error. To see this, suppose that we go through a sequence of states $s_1, s_2, \cdots, s_N$ and correspondingly we receive a sequence of rewards $r_1, r_2, \cdots, r_{N-1}$. In the case that the future rewards are not discounted, we have

$$\sum_{i=1}^{N-1} r_i = \sum_{i=1}^{N-1} (\epsilon(t_i) - \epsilon(t_{i+1})) = \epsilon(t_1) - \epsilon(t_N) = -\epsilon(t_N) \tag{26}$$

Note that $\epsilon(t_1) = 0$ because at the very beginning, the estimation and the ground truth are both 0, and $\epsilon(t_N)$ corresponds to the error of the subgraph counts at time t_N.

B. Policy learning

The core problem of an MDP is to find an optimal policy for the agent, which corresponds to a function that specifies the action that the agent should choose in a specific state so as to maximize the accumulative rewards. In our MDP, the states are high dimensional vectors, and the actions are in a continuous domain. Therefore, we adopt DDPG [23], an actor-critic method, to solve our MDP as it targets the same setting. Specifically, DDPG maintains two main networks. One is an actor network $\mu(s; \theta)$, parameterized by θ, which specifies the policy by deterministically mapping a state to a specific action,

$$a = \mu(s; \theta) = \sigma(\boldsymbol{W}s + \boldsymbol{b}) \tag{27}$$

where θ denotes the parameters $\{\boldsymbol{W}, \boldsymbol{b}\}$, and σ denotes the activation function. The other is a critic network $Q(s, a; \phi)$, parameterized by ϕ, which approximates the expected accumulative rewards the agent would receive by following any policy after seeing state s and taking action a.

For the training process, DDPG initializes two main networks $\mu(s; \theta)$ and $Q(s, a; \phi)$. It also involves two target networks $\mu'(s; \theta')$ and $Q'(s, a; \phi')$, which are used for calculating the losses for training the main networks. Besides, it maintains a replay memory, which contains the transitions that are used for training the network. The training process is as follows. Consider N experiences sampled uniformly from the replay memory, i.e., (s_i, a_i, r_i, s_{i+1}) for $i = 1, 2, \cdots, N$. For the critic network $Q(s, a; \phi)$, it computes the loss using the Bellman equation [24], i.e.,

$$L(\phi) = \frac{1}{N} \sum_{i=1}^{N} (y_i - Q(s_i, a_i; \phi))^2 \tag{28}$$

TABLE I
DATASET STATISTICS.

Category	Graph (Train)	$\|E\|$	Graph (Test)	$\|E\|$
Citation	cit-HepTH (HE)	2.67M	cit-patent (PT)	16.5M
Community	com-DBLP (DB)	1.04M	com-youtube (YT)	2.99M
Social	soc-Texas84 (TX)	1.59M	soc-twitter (TW)	265M
Web	web-Stanford (SF)	2.31M	web-google (GL)	5.10M

$$y_i = r_i + \gamma \cdot Q'(s_{i+1}, \mu'(s_{i+1};\theta');\phi') \tag{29}$$

where γ is a discount factor. For the actor network $\mu(s;\theta)$, it computes a loss as the negation of the expected return, i.e.,

$$L(\theta) = -\frac{1}{N}\sum_{i=1}^{N} Q(s_i, \mu(s_i;\theta);\phi) \tag{30}$$

Finally, it updates the parameters θ and ϕ by gradient descent.

V. EXPERIMENTS

A. Experimental Setup

Datasets. We use both real datasets and synthetic datasets in our experiments. We use four types of real graphs, including citation graphs, web graphs, community networks and online social networks, which are collected from an open-source network repository [25], [26]. For each graph in a dataset, we ignore the directions, weights and self-loops (if any). There are two ways to generate deletions for constructing fully dynamic graph streams, namely (1) massive deletion [16] and (2) light deletion [17], respectively. We generate fully dynamic graph streams under the massive deletion scenario as follows. All edges are inserted in their natural order but each edge insertion is followed with probability α by a massive deletion event where each edge currently in the graph is deleted with probability β_m independently. We generate fully dynamic graph streams under the light deletion scenario as follows. All edges are inserted in their natural order and each edge has probability β_l to be deleted, where the deletion is located at a random position after the corresponding edge insertion. Following the existing study [17], we generate some synthetic datasets by Forest Fire (FF) [27] model $G(n, p)$, in which n represents the number of vertices and p controls the density of the graph. Specifically, vertices arrive one at a time and form edges with some subset of the earlier vertices. Since the graph G generated by FF maintains the structural properties (heavy-tailed degree distributions, communities, etc.) and temporal properties (densification, etc.) of the real graphs, we transform G into an edge stream in their natural order. To ensure the synthetic datasets have similar scales as the real graphs, we set $n = 2$M and $p = 0.5$ based on empirical results.

Baselines and Metrics. We consider two instances of the `WSD` algorithm, which adopt two different weight functions. The first one uses the learned policy via reinforcement learning, which we denote by `WSD-L`. The second one uses a weight function defined with an existing heuristic [14], which we denote by `WSD-H`. Specifically, in `WSD-H`, $W(e, \mathcal{R}) = 9 \cdot |\mathcal{H}(e)| + 1$ [14], where $|\mathcal{H}(e)|$ is the number of subgraphs completed by edge e and some other edges in $\mathcal{R}$. We compare our `WSD` algorithms with three existing algorithms, namely `Triest` [16], `ThinkD` [19] and `WRS` [17]. We also compare our algorithm with the adapted framework `GPS-A`. All algorithms are single-pass algorithms that estimate the triangle counts with a fixed budget for storing sampled edges. For the subgraph patterns, we first choose the triangle (3-clique) and the wedge (length-2 paths) by following the existing study [14] since these are the two mostly used subgraph structures. In addition, we also consider a dense subgraph pattern, i.e., 4-clique, to verify that our algorithm can also work well on some large subgraph structures. We compare different algorithms by using the following metrics.

- **Absolute Relative Error (ARE)**. Let X be the ground truth of the number of subgraphs at the end of the input streams and $\widehat{X}$ be an estimated value of X. The absolute relative error is $\frac{|\widehat{X} - X|}{X} \times 100\%$.
- **Mean Absolute Relative Error (MARE)**. Let X_t be the ground truth of the number of subgraphs by t and $\widehat{X}_t$ be an estimated value of X_t. The mean absolute relative error is $\frac{1}{T}\sum_{i=1}^{T}\frac{|\widehat{X}_i - X_i|}{X_i} \times 100\%$.

Due to the random nature of all algorithms, for all experiments, given a setting, we report the mean over 100 times of sampling.

Policy Learning. There are two neural networks used in `WSD-L` method. The actor network $\mu(s;\theta)$ involves one input layer and one output layer, and uses ReLU function as the activation function. We add one to the output to avoid assigning zero weights. The critic network $Q(s, a;\phi)$ involves one input layer, one hidden layer and one output layer, where the hidden layer involves 10 neurons and uses ReLU as the activation function. To avoid data scale issues, batch normalization is employed before the activation. For each graph in a real dataset, we use another real graph under the same type for training, as specified in Table I, since they may have some similar structural and temporal properties. For each graph in the synthetic dataset, we generate a graph $G(n = 2\text{M}, p = 0.5)$ by FF model for training. Under massive (resp. light) deletion scenario, we generate 10 different edge event streams with the same parameters α and β_m (resp. β_l) and use these generated graphs for training based on empirical findings. Basically, using fewer streams would suffer from the over-fitting problem and using more would incur larger training cost without improving the quality of the model much. We set the maximum size of the replay buffer as 10,000 and the parameter N as 128. We train the networks for 1,000 iterations. In addition, we use the Adam stochastic gradient descent with learning rate of 0.001. For the reward discount γ, we set it as 0.99.

Evaluation Platform. All methods for comparison are implemented in C++. Note that we first implement and train `WSD-L` using Pytorch (Python 3.6). Then we hardcode the parameters $\theta = \{\boldsymbol{W}, \boldsymbol{b}\}$ in C++ to improve the efficiency. All experiments are conducted on a machine with Intel Core i9-10940X CPU and a single Nvidia GeForce 2080Ti GPU.

TABLE II

Results of counting wedges under the massive deletion scenario, where we use the default values of the parameters, $M = 200,000$, $\alpha = 3,000,000^{-1}$ and $\beta_m = 0.8$.

Graph	WSD-L	WSD-H	GPS-A	Triest	ThinkD	WRS
	Absolute Relative Error (%)					
cit-PT	**0.046**	0.051	0.058	0.077	0.071	0.062
com-YT	**0.011**	0.013	0.061	0.125	0.104	0.092
soc-TW	**0.243**	0.411	0.434	0.627	0.572	0.483
web-GL	**0.041**	0.044	0.117	0.815	0.670	0.366
synthetic	**0.107**	0.148	0.192	0.564	0.324	0.231
	Mean Absolute Relative Error (%)					
cit-PT	**0.036**	0.039	0.072	0.104	0.067	0.054
com-YT	**0.007**	0.008	0.057	0.061	0.054	0.044
soc-TW	**0.306**	0.451	0.688	1.310	1.003	0.894
web-GL	**0.039**	0.040	0.078	0.771	0.253	0.101
synthetic	**0.213**	0.312	0.401	0.646	0.769	0.443
	Running Time (s)					
cit-PT	65.9	**62.4**	67.1	186.7	188.0	191.9
com-YT	9.6	9.3	9.9	**7.8**	7.9	8.1
soc-TW	1,940.7	**1,832.2**	2,981.9	4,392.3	4,417.3	4,592.1
web-GL	13.7	**13.2**	14.3	14.1	14.6	14.8
synthetic	43.3	**42.9**	45.2	262.7	263.2	267.9

TABLE III

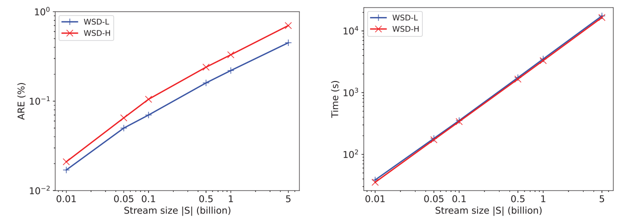
Results of counting triangles under the massive deletion scenario, where we use the default values of the parameters, $M = 200,000$, $\alpha = 3,000,000^{-1}$ and $\beta_m = 0.8$.

Graph	WSD-L	WSD-H	GPS-A	Triest	ThinkD	WRS
	Absolute Relative Error (%)					
cit-PT	**0.075**	0.083	0.106	0.175	0.143	0.142
com-YT	**0.048**	0.053	0.073	0.188	0.109	0.067
soc-TW	**0.404**	0.712	0.893	1.214	1.056	0.952
web-GL	**0.031**	0.037	0.734	0.197	0.195	0.136
synthetic	**2.507**	3.124	3.612	4.293	3.318	3.143
	Mean Absolute Relative Error (%)					
cit-PT	**0.059**	0.065	0.083	0.113	0.089	0.079
com-YT	**0.052**	0.064	0.141	0.236	0.144	0.204
soc-TW	**0.513**	0.748	0.946	1.433	1.341	1.005
web-GL	**0.054**	0.088	0.207	1.024	0.809	0.319
synthetic	**3.614**	4.278	4.662	6.691	5.643	4.817
	Running Time (s)					
cit-PT	70.4	**66.7**	71.5	189.5	193.1	197.4
com-YT	10.2	9.9	10.6	**8.0**	8.2	8.3
soc-TW	2,144.4	**2,091.9**	3,329.3	4,810.8	4,839.3	5,023.5
web-GL	16.1	**14.9**	17.3	16.9	17.5	18.4
synthetic	49.3	**48.1**	53.7	260.2	262.1	265.0

Implementation codes and datasets can be found via this link https://github.com/wangkaixin219/WSD/.

B. *Experimental Results*

(1) Evaluation on real and synthetic datasets. Table II and Table III show the results of counting triangles and wedges under massive deletion scenario. The result of counting 4-cliques and those under the light deletion show similar clues and are thus presented in the technical report [22]. Consider the effectiveness. The results meet our expectations that weighted sampling works better than uniform sampling on all datasets and under two metrics. In addition, because `WSD-L` uses RL to adaptively capture the importance of the edges, it further improves the effectiveness of `WSD-H` which applies a heuristic based weight function. Consider the efficiency. As analyzed in Section III-C, for all datasets except com-YT,



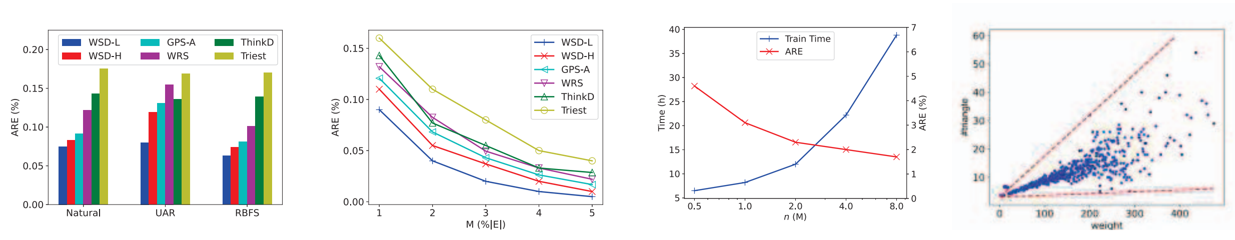
(a) ARE (massive deletion scenario) (b) Time (massive deletion scenario)

Fig. 1. Results on scalability test of counting triangles, reporting the ARE and running time of `WSD-L` and `WSD-H` with different stream sizes (synthetic).

we observe that the number of generated deletions satisfies $|D| > (\log M)/M \cdot |A|$, where $(\log M)/M \approx 3 \times 10^{-5}$, and thus our proposed algorithms run much faster than existing algorithms. In addition, because `WSD-L` needs to calculate a state when an insertion event comes, it runs slightly slower than `WSD-H`. Compared among different subgraph patterns, the ARE and MARE become larger as the size of the subgraph pattern increases. The reasons are as follows. First, when the size of the subgraph pattern increases, it becomes more difficult for a new edge to form some subgraph structures with the sampled edges. For example, when an edge event comes, a 4-clique could be formed only if the other 5 edges have been sampled while a wedge would be formed if the other edge has been sampled. Also, the inclusion probability of a large subgraph pattern is usually much smaller than that of a small subgraph pattern, which means a higher variance would be produced when counting larger subgraph structures.

(2) Scalability test. To study the scalability of `WSD-L` and `WSD-H`, we generate a series of graphs with different sizes. To be specific, we first use FF model $G(n = 1\text{B}, p = 0.5)$ to generate the whole graph with slightly more than 5B edges (5.07 billion edges). We then create the edge event stream by adding deletions under massive and light deletion scenarios. Finally, we generate graphs with different sizes by picking the first 10M, 50M, 100M, 500M, 1B and 5B edge events. We set M as 1M following [14]. The results under massive deletion scenario are shown in Figure 1. Consider the effectiveness. We observe that as the size of the stream increases, the ARE results also increases. The reason is that for all sizes of the streams, we use 1M edges as the sample. Therefore, the estimation results on a stream with more edge events would be less accurate. Still, we can provide a relatively accurate estimation ($< 1\%$ error) when we sample only 0.02% edges. Consider the efficiency. The running time of `WSD` is linear to the size of the stream, which is consistent with our theoretical analysis of the time complexity of `WSD`. Moreover, we also collect the results of the average time of updating the reservoir when an edge event occurs. The average processing time of each edge is around 3.2μs.

(3) Impact of the ordering of the stream. We follow [16] to generate different orderings of the stream. The default setting is the natural ordering of a graph. We also consider the uniform-at-random (UAR) ordering and the random BFS (RBFS) ordering. The UAR ordering is to add edges in the



(a) Ordering of the stream (cit-PT) (b) Max. reservoir size M (cit-PT) (c) Training size (synthetic) (d) Relationship (cit-PT)

Fig. 2. Results of counting triangles under massive deletion scenario: (a) the ARE under different orderings of the stream; (b) the ARE under different M's; (c) the training time and ARE under different sizes of training graphs; (d) the relationship between the weights and numbers of triangles.

order of a permutation of the natural order while the RBFS ordering is to start from a random vertex and add edges in the order of a BFS exploration of the graph. For example, in a social network, when a celebrity registers an account in a new platform, his/her followers would likely create connections (i.e., edge insertions) to him/her in a very short time, as BFS does. Figure 2(a) shows the ARE results of counting triangles on cit-PT under massive deletion scenario. We observe that WSD−L algorithm provides estimations with the smallest errors across different settings of ordering. These results show the robustness of our WSD−L algorithm which is due to its data-driven nature and adaptability to various underlying dynamics.

(4) Effects of M. We change the maximum reservoir size from $1\%\times|E|$ to $5\%\times|E|$. Figure 2(b) shows the results of counting triangles on cit-PT under the massive deletion scenario. The MARE results and the results of counting triangles on other datasets show the similar trends, thus omitted. Under both scenarios, we set the probability parameters same as those used in previous experiments. Our proposed algorithms consistently outperform other existing algorithms. Our algorithm can provide an accurate estimation ($< 0.01\%$ error) when we sample only 4% edges.

TABLE IV
TRAINING TIME (HOURS) OF COUNTING TRIANGLES ($\triangle$) AND WEDGES ($\wedge$) ON FOUR REAL DATASETS UNDER MASSIVE DELETION SCENARIO.

Graph	cit-HE		com-DB		soc-TX		web-SF	
Pattern H	$\triangle$	$\wedge$	$\triangle$	$\wedge$	$\triangle$	$\wedge$	$\triangle$	$\wedge$
Time (h)	16.7	15.9	8.2	7.6	10.6	9.3	13.5	12.1

(5) Training. We first report the training time of counting triangles and wedges on four datasets under massive deletion scenarios in Table IV. We note that it normally takes several hours to train a satisfactory model. Then we study how the size of training graphs affects the model performance. We generate different graphs $G(n, p = 0.5)$ by FF model with $n \in \{0.5\text{M}, 1\text{M}, 2\text{M}, 4\text{M}, 8\text{M}\}$ for training. We then generate a graph $G(n = 10\text{M}, p = 0.5)$ by FF model, which contains around 50M edges, for testing. The results of counting triangles under massive deletion scenario are shown in Figure 2(c). We observe that as the the size of graph used for training grows exponentially, the training time also increases exponentially, but the effectiveness improves slightly only. Therefore, we learn the weight function on the graphs with size around 10%-20% of those of the testing graphs to balance the effectiveness and the training costs.

(6) Relationship between an edge's weight in WSD−L and its associated subgraph counts. We reveal the relationship by collecting the statistics of (1) each edge's weight and (2) the number of triangles, which contain the edge and are formed by the end of the stream. Since an edge's weight depends on those edges that have been sampled in the reservoir when it appears, this value corresponds to a random variable. Therefore, we run WSD−L 100 times, and for an edge, we report the mean of its weights. Figure 2(d) shows the results of counting triangles on cit-PT under massive deletion scenario. We observe the trend that the larger an edge's weight is, the more triangles the edge is involved in, which is well aligned with the intuition behind Eq. (21).

TABLE V
RESULTS ON TRANSFERABILITY OF WSD−L UNDER MASSIVE SCENARIO.

(**Training**)	cit-HE	com-DB	soc-TX	web-SF	synthetic	WSD-H
cit-PT	**0.076**	0.080	0.077	0.078	0.081	0.083
com-YT	0.049	**0.048**	0.053	0.052	0.050	0.053
soc-TW	0.653	0.567	**0.451**	0.510	0.687	0.711
web-GL	0.033	0.036	0.035	**0.032**	0.034	0.037

(7) Transferability of WSD−L. We test the transferability of WSD−L by applying the policy (i.e., the weight function), which has been trained on a graph under a certain category, to a graph under another category. To ensure fairness of comparison, we use the first 1M edges of the training graph in each dataset to train the model. Table V shows the ARE results of counting triangles. As expected, the model works the best on a graph under the same category as the training dataset and has its performance degraded to some extent when being used on a graph under a different category. In addition, the model, when used on a graph under a different category from the one of the training dataset, still works better than the heuristic-based method, which verifies the transferability of our method.

(8) Evaluation on insertion-only scenario. We evaluate a special case of the fully dynamic graph streams, in which it only consists of edge insertion events. Under the insertion-only scenario, WSD−H and GPS−A would be equivalent to GPS. We also compare Triest, ThinkD and WRS under this setting. The results are presented in Table VI. Consider the effectiveness. Two weight-sensitive sampling schemes (WSD−L and GPS) outperform other algorithms. Due to the data-

TABLE VI
EXPERIMENTAL RESULTS OF COUNTING TRIANGLES ON CIT-PT UNDER INSERTION-ONLY SCENARIO.

	WSD-L	GPS	Triest	ThinkD	WRS
ARE (%)	**0.30**	<u>0.34</u>	0.85	0.41	0.36
MARE (%)	**0.14**	<u>0.20</u>	0.66	0.24	0.22
Time (s)	49.6	48.5	**39.3**	<u>40.2</u>	41.1

driven fashion of WSD-L, it further improves the performance. Consider the efficiency. For WSD-L and GPS, it would cost $O(\log M)$ (worst case) to maintain the min-priority queue when an edge is inserted while for the other three algorithms, it would cost $O(1)$ to update the reservoir. Therefore, WSD-L and GPS run slightly slower.

VI. Related Work

Subgraph Counting. Subgraph counting (e.g., triangle counting) in a streaming graph is a fundamental problem in graph analysis and has been extensively studied. For example, many studies [16], [28], [14], [29], [19], [3], [30], [17], [31], [2], [32], [33], [34], [35], [36] propose algorithms to estimate triangle counts in a streaming graph. Among these studies, some studies [16], [28], [14], [29], [19], [3], [30], [17] assume some space constraint for sampled edges while others [31], [2], [32], [33], [34], [35], [36] assume no constraint on the number of sampled edges. Our work follows the settings in [16], [17], [3], [18], [19], which target fully dynamic graph steams and assume a constraint on the number of sampled edges. These existing methods all adopt the *random pairing* technique [37], which extends the standard reservoir sampling method to support deletions, where each deletion from the stream is "paired with" (or compensated by) a subsequent insertion. Specifically, Triest [16] is the first method to estimate the triangle count in fully graph streams with random pairing, where the estimation is only updated when an edge is sampled. Later, [3], [19] extend [16] and propose ThinkD. Instead of simply discarding the unsampled edges as Triest does, ThinkD first updates the estimation and then updates the sampled graph, which results in a smaller variance. More recently, WRS [17], [18] is proposed to further exploit the temporal locality information. It divides the storage budget into two parts, namely waiting room and reservoir, and stores the most recent edges in the waiting room unconditionally and the sampled edges in the reservoir. As explained in Section I, these existing studies [16], [3], [17], [19], [18] all sample the edges with uniform probabilities, e.g., each edge is treated equally for sampling, which would likely result in sub-optimal samples. In addition, there are some other studies on the problem of counting subgraphs in a static graph [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], which is different from the setting targeted in our work. It is worth noting that there are several recent studies, which develop learning-based algorithms for subgraph counting problem [54], [55], [56], [57], [58]. They differ from our study in (1) all these studies target static graphs; (2) [56], [57], [55] target the problem of counting subgraph isomorphism matches, where each vertex has a label. Some other studies [59], [60] target the subgraph counting problem on online social networks. Interested readers are referred to two surveys [61], [62] for more details of subgraph counting algorithms.

Reservoir Sampling. Reservoir sampling was first studied in [63], which samples items in a stream with equal probabilities. [64] implements the simple reservoir sampling via min-wise sampling, which provides the same effect as [63] but consumes more memory. To handle fully dynamic streams with deletions, [37] proposes a framework, namely random pairing, where each deletion from the stream is mapped to a subsequent insertion. Random pairing guarantees uniformity, and thus it is widely used for sampling fully dynamic streams in which items have the same weight. To sample weighted items, [21], [65] both propose a rank-based sampling scheme. They differ in their priority functions and are used in different applications. However, these two methods can only handle insertion-only streams while our weighted sampling method can handle fully dynamic streams.

Reinforcement Learning. Given a specific environment, which is generally formulated as a Markov Decision Process (MDP) [20], reinforcement learning helps an agent in the environment learn how to map the situations to actions so as to maximize the accumulative rewards [66]. In this paper, we model the process of deciding the weights of edges in a stream sequentially as an MDP and use a popular policy gradient method DDPG [23] for solving the problem. To the best of knowledge, this is the first deep reinforcement learning based solution applied to the subgraph counting in graph streams.

VII. Conclusion

In this paper, we study the subgraph counting problem in fully dynamic graph streams. We propose a weighted sampling algorithm WSD, which samples edges non-uniformly based on their weights, and construct an unbiased estimator based on the sampled edges by WSD. Furthermore, we develop a reinforcement learning based method for setting weights of edges in a data-driven fashion. Compared with existing algorithms, our algorithms can produce estimations with smaller errors and often run faster. One interesting direction for future research is to extend the reinforcement learning enhanced WSD method to other problems on fully dynamic graphs.

Acknowledgments. This research is supported by the Ministry of Education, Singapore, under its Academic Research Fund (Tier 2 Award MOE-T2EP20221-0013 and Tier 1 Award (RG77/21)). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore. This work was also partially supported by the A*STAR Cyber-Physical Production System (CPPS) – Towards Contextual and Intelligent Response Research Program, under the RIE2020 IAF-PP Grant A19C1a0018, and Model Factory@SIMTech.

REFERENCES

[1] J.-P. Eckmann and E. Moses, "Curvature of co-links uncovers hidden thematic layers in the world wide web," *Proceedings of the national academy of sciences*, vol. 99, no. 9, pp. 5825–5829, 2002.

[2] Y. Lim and U. Kang, "Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams," in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 685–694.

[3] K. Shin, S. Oh, J. Kim, B. Hooi, and C. Faloutsos, "Fast, accurate and provable triangle counting in fully dynamic graph streams," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 14, no. 2, pp. 1–39, 2020.

[4] J. Leskovec, L. A. Adamic, and B. A. Huberman, "The dynamics of viral marketing," *ACM Transactions on the Web (TWEB)*, vol. 1, no. 1, pp. 5–es, 2007.

[5] P. Zhao, C. Aggarwal, and G. He, "Link prediction in graph streams," in *2016 IEEE 32nd international conference on data engineering (ICDE)*. IEEE, 2016, pp. 553–564.

[6] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annual review of sociology*, pp. 415–444, 2001.

[7] L. M. Aiello, A. Barrat, R. Schifanella, C. Cattuto, B. Markines, and F. Menczer, "Friendship prediction and homophily in social media," *ACM Transactions on the Web (TWEB)*, vol. 6, no. 2, pp. 1–33, 2012.

[8] S. Wasserman, K. Faust *et al.*, "Social network analysis: Methods and applications," 1994.

[9] M. Jamali and M. Ester, "A transitivity aware matrix factorization model for recommendation in social networks," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[10] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler, "Counting triangles in data streams," in *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2006, pp. 253–262.

[11] M. Jha, C. Seshadhri, and A. Pinar, "A space efficient streaming algorithm for triangle counting using the birthday paradox," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 589–597.

[12] U. Kang, B. Meeder, and C. Faloutsos, "Spectral analysis for billion-scale graphs: Discoveries and implementation," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2011, pp. 13–25.

[13] U. Kang, B. Meeder, E. E. Papalexakis, and C. Faloutsos, "Heigen: Spectral analysis for billion-scale graphs," *IEEE Transactions on knowledge and data engineering*, vol. 26, no. 2, pp. 350–362, 2012.

[14] N. K. Ahmed, N. Duffield, T. L. Willke, and R. A. Rossi, "On sampling from massive graph streams," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1430–1441, 2017.

[15] L. Zhang, H. Jiang, F. Wang, D. Feng, and Y. Xie, "T-sample: A dual reservoir-based sampling method for characterizing large graph streams," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1674–1677.

[16] L. D. Stefani, A. Epasto, M. Riondato, and E. Upfal, "Triest: Counting local and global triangles in fully dynamic streams with fixed memory size," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 11, no. 4, pp. 1–50, 2017.

[17] D. Lee, K. Shin, and C. Faloutsos, "Temporal locality-aware sampling for accurate triangle counting in real graph streams," *The VLDB Journal*, vol. 29, no. 6, pp. 1501–1525, 2020.

[18] K. Shin, "Wrs: Waiting room sampling for accurate triangle counting in real graph streams," in *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017, pp. 1087–1092.

[19] K. Shin, J. Kim, B. Hooi, and C. Faloutsos, "Think before you discard: Accurate triangle counting in graph streams with deletions," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2018, pp. 141–157.

[20] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[21] N. Duffield, C. Lund, and M. Thorup, "Priority sampling for estimation of arbitrary subset sums," *Journal of the ACM (JACM)*, vol. 54, no. 6, pp. 32–es, 2007.

[22] K. Wang, C. Long, D. Yan, J. Zhang, and H. Jagadish, "Reinforcement learning enhanced weighted sampling for accurate subgraph counting on fully dynamic graph streams," *arXiv preprint arXiv:2211.06793*, 2022.

[23] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning." in *ICLR (Poster)*, 2016.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[25] R. Rossi and N. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[26] R. A. Rossi and N. K. Ahmed, "An interactive data repository with visual analytics," *ACM SIGKDD Explorations Newsletter*, vol. 17, no. 2, pp. 37–41, 2016.

[27] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, pp. 2–es, 2007.

[28] P. Wang, Y. Qi, Y. Sun, X. Zhang, J. Tao, and X. Guan, "Approximately counting triangles in large graph streams including edge duplicates with a fixed memory usage," *Proceedings of the VLDB Endowment*, vol. 11, no. 2, pp. 162–175, 2017.

[29] M. Jung, Y. Lim, S. Lee, and U. Kang, "Furl: Fixed-memory and uncertainty reducing local triangle counting for multigraph streams," *Data Mining and Knowledge Discovery*, vol. 33, no. 5, pp. 1225–1253, 2019.

[30] L. Zhang, H. Jiang, F. Wang, D. Feng, and Y. Xie, "Reservoir-based sampling over large graph streams to estimate triangle counts and node degrees," *Future Generation Computer Systems*, vol. 108, pp. 244–255, 2020.

[31] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "Doulion: counting triangles in massive graphs with a coin," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 837–846.

[32] P. Wang, P. Jia, Y. Qi, Y. Sun, J. Tao, and X. Guan, "Rept: A streaming algorithm of approximating global and local triangle counts in parallel," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 758–769.

[33] R. Etemadi and J. Lu, "Pes: Priority edge sampling in streaming triangle estimation," *IEEE Transactions on Big Data*, 2019.

[34] Y. Lim, M. Jung, and U. Kang, "Memory-efficient and accurate sampling for counting local triangles in graph streams: from simple to multigraphs," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 12, no. 1, pp. 1–28, 2018.

[35] M. Yu, C. Song, J. Gu, and M. Liu, "Distributed triangle counting algorithms in simple graph stream," in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2019, pp. 294–301.

[36] G. Han and H. Sethu, "On counting triangles through edge sampling in large dynamic graphs," in *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. Springer, 2017, pp. 133–157.

[37] R. Gemulla, W. Lehner, and P. J. Haas, "A dip in the reservoir: Maintaining sample synopses of evolving datasets," in *Proceedings of the 32nd international conference on Very large data bases*, 2006, pp. 595–606.

[38] N. Alon, R. Yuster, and U. Zwick, "Finding and counting given length cycles," *Algorithmica*, vol. 17, no. 3, pp. 209–223, 1997.

[39] S. Arifuzzaman, M. Khan, and M. Marathe, "Distributed-memory parallel algorithms for counting and listing triangles in big graphs," *arXiv preprint arXiv:1706.05151*, 2017.

[40] ——, "Fast parallel algorithms for counting and listing triangles in big graphs," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 14, no. 1, pp. 1–34, 2019.

[41] Z. Ouyang, S. Wu, T. Zhao, D. Yue, and T. Zhang, "Memory-efficient gpu-based exact and parallel triangle counting in large graphs," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2019, pp. 2195–2199.

[42] V. S. Dave, N. K. Ahmed, and M. Al Hasan, "E-clog: Counting edge-centric local graphlets," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 586–595.

[43] P. Wang, J. C. Lui, B. Ribeiro, D. Towsley, J. Zhao, and X. Guan, "Efficiently estimating motif statistics of large networks," *ACM Transactions*

on Knowledge Discovery from Data (TKDD), vol. 9, no. 2, pp. 1–27, 2014.

[44] G. Han and H. Sethu, "Waddling random walk: Fast and accurate mining of motif statistics in large graphs," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 181–190.

[45] C. Yang, M. Lyu, Y. Li, Q. Zhao, and Y. Xu, "Ssrw: A scalable algorithm for estimating graphlet statistics based on random walk," in *International Conference on Database Systems for Advanced Applications*. Springer, 2018, pp. 272–288.

[46] X. Chen, Y. Li, P. Wang, and J. C. Lui, "A general framework for estimating graphlet statistics via random walk," *Proceedings of the VLDB Endowment*, vol. 10, no. 3, 2016.

[47] T. K. Saha and M. Al Hasan, "Finding network motifs using mcmc sampling," in *Complex Networks VI*. Springer, 2015, pp. 13–24.

[48] S. Huang, Y. Li, Z. Bao, and Z. Li, "Towards efficient motif-based graph partitioning: An adaptive sampling approach."

[49] N. K. Ahmed, T. L. Willke, and R. A. Rossi, "Estimation of local subgraph counts," in *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2016, pp. 586–595.

[50] M. Bressan, F. Chierichetti, R. Kumar, S. Leucci, and A. Panconesi, "Counting graphlets: Space vs time," in *Proceedings of the tenth ACM international conference on web search and data mining*, 2017, pp. 557–566.

[51] ——, "Motif counting beyond five nodes," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 12, no. 4, pp. 1–25, 2018.

[52] M. Bressan, S. Leucci, and A. Panconesi, "Motivo: fast motif counting via succinct color coding and adaptive sampling," *Proceedings of the VLDB Endowment*, vol. 12, no. 11, pp. 1651–1663, 2019.

[53] J. M. Klusowski and Y. Wu, "Counting motifs with graph sampling," in *Conference On Learning Theory*. PMLR, 2018, pp. 1966–2011.

[54] Z. Chen, L. Chen, S. Villar, and J. Bruna, "Can graph neural networks count substructures?" *Advances in neural information processing systems*, vol. 33, pp. 10 383–10 395, 2020.

[55] X. Liu, H. Pan, M. He, Y. Song, X. Jiang, and L. Shang, "Neural subgraph isomorphism counting," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1959–1969.

[56] K. Zhao, J. X. Yu, H. Zhang, Q. Li, and Y. Rong, "A learned sketch for subgraph counting," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2142–2155.

[57] H. Wang, R. Hu, Y. Zhang, L. Qin, W. Wang, and W. Zhang, "Neural subgraph counting with wasserstein estimator," in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 160–175.

[58] I. Roy, V. S. B. R. Velugoti, S. Chakrabarti, and A. De, "Interpretable neural subgraph matching for graph retrieval," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 7, 2022, pp. 8115–8123.

[59] Y. Wu, C. Long, A. W.-C. Fu, and Z. Chen, "Counting edges and triangles in online social networks via random walk," in *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*. Springer, 2017, pp. 346–361.

[60] Y. Wu, C. Long, A. W. Fu, and Z. Chen, "Counting edges with target labels in online social networks via random walk," in *EDBT/ICDT 2018 Joint Conference*. EDBT Association, 2018.

[61] M. Al Hasan and V. S. Dave, "Triangle counting in large networks: a review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 2, p. e1226, 2018.

[62] P. Ribeiro, P. Paredes, M. E. Silva, D. Aparicio, and F. Silva, "A survey on subgraph counting: concepts, algorithms, and applications to network motifs and graphlets," *ACM Computing Surveys (CSUR)*, vol. 54, no. 2, pp. 1–36, 2021.

[63] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software (TOMS)*, vol. 11, no. 1, pp. 37–57, 1985.

[64] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 4, no. 2, pp. 1–40, 2004.

[65] P. S. Efraimidis and P. G. Spirakis, "Weighted random sampling with a reservoir," *Information Processing Letters*, vol. 97, no. 5, pp. 181–185, 2006.

[66] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.