Mining Frequent Patterns in Evolving Graphs

Cigdem Aslay
Aalto University
cigdem.aslay@aalto.fi

Muhammad Anis Uddin Nasir
King Digital Entertainment Ltd
anis.nasir@king.se

Gianmarco De Francisci Morales
ISI Foundation
gdfm@acm.org

Aristides Gionis
Aalto University
aristides.gionis@aalto.fi

ABSTRACT

Given a labeled graph, the frequent-subgraph mining (FSM) problem asks to find all the k-vertex subgraphs that appear with frequency greater than a given threshold. FSM has numerous applications ranging from biology to network science, as it provides a compact summary of the characteristics of the graph. However, the task is challenging, even more so for evolving graphs due to the streaming nature of the input and the exponential time complexity of the problem.

In this paper, we initiate the study of the approximate FSM problem in both incremental and fully-dynamic streaming settings, where arbitrary edges can be added or removed from the graph. For each streaming setting, we propose algorithms that can extract a high-quality approximation of the frequent k-vertex subgraphs for a given threshold, at any given time instance, with high probability. In contrast to the existing state-of-the-art solutions that require iterating over the entire set of subgraphs for any update, our algorithms operate by maintaining a uniform sample of k-vertex subgraphs with optimized neighborhood-exploration procedures local to the updates. We provide theoretical analysis of the proposed algorithms and empirically demonstrate that the proposed algorithms generate high-quality results compared to baselines.

ACM Reference Format:
Cigdem Aslay, Muhammad Anis Uddin Nasir, Gianmarco De Francisci Morales, and Aristides Gionis. 2018. Mining Frequent Patterns in Evolving Graphs. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271772

1 INTRODUCTION

Frequent-subgraph mining (FSM) is a fundamental graph-mining task with applications in various disciplines, including bioinformatics, security, and social sciences. The goal of FSM is to find subgraph patterns of interest that are frequent in a given graph. Such subgraphs might be indicative of an important protein interaction, a possible intrusion, or a common social norm. FSM also finds applications in graph classification and indexing.

Existing algorithms for subgraph mining are not scalable to large graphs that arise, for instance, in social domains. In addition, these graphs are usually produced as a result of a dynamic process, hence are subject to continuous changes. For example, in social networks new edges are added as a result of the interactions of their users, and the graph structure is in continuous flux. Whenever the graph changes, i.e., by adding or removing an edge, a large number of new subgraphs can be created, and existing subgraphs can be modified or destroyed. Keeping track of all the possible changes in the graph is subject to combinatorial explosion, thus, is highly challenging.

In this paper we address the problem of mining frequent subgraphs in an evolving graph, which is represented as a stream of edge updates — additions or deletions. Only a few existing works consider a similar setting [1, 4, 29]. Bifet et al. [4] deal with a transactional setting where the input is a stream of small graphs. Their setting is similar to the one considered by frequent-itemset mining, so many of the existing results can be reused. Conversely, in our case, there is a single graph that is continuously evolving. Ray et al. [29] consider a scenario similar to the one we study in this paper. They consider a single graph with continuous updates, although they only allow incremental ones (edge addition) rather than the fully-dynamic ones we consider (edge addition and deletion). Moreover, their approach is a simple heuristic that does not provide any correctness guarantee. Our approach, instead, is able to provably find the frequent subgraphs in a fully dynamic graph stream. Abdelhamid et al. [1] tackle a problem setting similar to ours, with a single fully-dynamic evolving graph. They propose an exact algorithm which tracks patterns which are at the "fringe" of the frequency threshold, and borrows heavily from existing literature on incremental pattern mining. As such, they need to use a specialized notion of frequency for graphs (minimum image support). Instead, our algorithm provides an approximate solution which uses the standard notion of induced subgraph isomorphism for frequency.

This paper is the first to propose an approximation algorithm for the frequent-subgraph mining problem on a fully-dynamic evolving graph. We propose a principled sampling scheme for subgraphs and provide theoretical justifications for its accuracy. Differently from previous work on sampling from graph streams, our method relies on sampling *subgraphs* rather than edges. This choice enables sampling any kind of subgraph of the same size with equal probability, and thus simplifies dramatically the design of the frequency estimators. We maintain a uniform sample of subgraphs via reservoir sampling, which in turn allows us to estimate the frequency of different patterns. To handle deletions in the stream, we employ an adapted version of *random pairing* [13]. Finally, to increase the efficiency of our sampling procedure during the exploration of the local neighborhood of updated edges, we employ an adaptation

Part of the work was done while the first author was at ISI Foundation, the second author was at KTH, and the third author was at QCRI..

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3271772

of the "skip optimization," proposed by Vitter [35] for reservoir sampling and by Gemulla et al. [14] for random pairing.

Concretely, our main contributions are the following:

- We are the first to propose an approximation algorithm for the frequent-subgraph mining problem for evolving graph.
- We propose a new subgraph-based sampling scheme.
- We show how to use random pairing to handle deletions.
- We describe how to implement neighborhood exploration efficiently via "skip optimization."
- We provide theoretical analysis and guarantees on the accuracy of the algorithm.

2 PROBLEM DEFINITION

We consider graphs with vertex and edge *labels*. We model dynamic graphs as a sequence of *edge additions* and *deletions*.

We assume monitoring a graph that changes over time. For any time $t \geq 0$, we let $G^t = (V^t, E^t)$ be the graph that has been observed up to and including time t, where V^t represents the set of vertices and E^t represents the set of edges. We assume that vertices and edges have labels, and we write L and Q for the sets of labels of vertices and edges, respectively. For each vertex $v \in V^t$ we denote its label by $\ell_v \in L$, and similarly, for each edge $e = (u, v) \in E^t$ we denote its label by $q_e \in Q$. Initially, at time $t = 0$, we have $V^0 = E^0 = \varnothing$. For any $t \geq 0$, at time $t + 1$ we receive an update tuple $\langle \mathbf{o}, e, q \rangle$ from a stream, where $\mathbf{o} \in \{+, -\}$ represents an update operation, addition or deletion, $e = (u, v)$ is a pair of vertices, and $q \in Q$ is an edge label. The graph $G^{t+1} = (V^{t+1}, E^{t+1})$ is obtained by adding a new edge or deleting an existing edge as follows:

$$E^{t+1} = \begin{cases} E^t \cup \{e\} \text{ and } q_e = q & \text{if } \mathbf{o} = + \\ E^t \setminus \{e\} & \text{if } \mathbf{o} = - \end{cases}.$$

Additions and deletions of vertices are treated similarly. Furthermore, we assume that when adding an edge $e = (u, v)$, the vertices u and v are added in the graph too, if they are not present at time t. Similarly, when deleting a vertex, we assume that all incident edges are deleted too, prior to the vertex deletion. Our model deals with the fully dynamic stream of edges, which is different from the stream of graphs [36]. For simplicity of exposition, in the rest of the paper we discuss only edge additions and deletions — vertex operations can be handled rather easily.

We use $n_t = |V^t|$ and $m_t = |E^t|$ to refer to the number of vertices and edges, respectively, at time t. In this work, we considered simple, connected, and undirected graphs. The *neighborhood* of a vertex $u \in V^t$ at time t is defined as $N_u^t = \{v \mid (u, v) \in E^t\}$, and its *degree* as $d_u^t = |N_u^t|$. Similarly, the h-hop neighborhood of u at time t is denoted as $N_{u,h}^t$, and indicates the set of the vertices that can be reached from u in h steps by following the edges E^t. To simplify the notation, we omit to specify the dependency on t when it is obvious from the context.

For any graph $G = (V, E)$ and a subset of vertices $S \subseteq V$, we say that $G_S = (S, E(S))$ is an *induced* subgraph of G if for all pairs of vertices $u, v \in S$ it is $(u, v) \in E(S)$ if and only if $(u, v) \in E$. We define $\mathcal{C}^k$ to be the set of all induced subgraphs with k vertices in G. All subgraphs considered in this paper are induced subgraphs, unless stated otherwise.

We say that two subgraphs of G, denoted by $G_S \in \mathcal{C}^k$ and $G_T \in \mathcal{C}^k$ are isomorphic if there exists a bijection $I : S \to T$ such that $(u, v) \in E(S)$ if and only if $(I(u), I(v)) \in E(T)$ and the mapping I preserves the vertex and edge labels, i.e., $\ell_u = \ell_{I(u)}$ and $q_{(u,v)} = q_{(I(u),I(v))}$, for all $u \in S$ and for all $(u, v) \in E(S)$. We write $G_S \simeq G_T$ to denote that G_S and G_T are isomorphic.

The isomorphism relation $\simeq$ partitions the set of subgraphs $\mathcal{C}^k$ into T_k equivalence classes,[1] denoted by $C_1^k, \cdots, C_{T_k}^k$. Each equivalence class C_i^k is called a *subgraph pattern*.

We define the support set $\sigma(G_S)$ of any k-vertex subgraph $G_S \in \mathcal{C}^k$ as the number of k-vertex subgraphs of G that are isomorphic to G_S, i.e., $\sigma(G_S) = |C_i^k|$, where $G_S \in C_i^k$. We then define the frequency $f(G_S)$ of a subgraph G_S as the fraction of k-vertex subgraphs of G that are isomorphic to G_S, i.e., $f(G_S) = \sigma(G_S)/|\mathcal{C}^k|$.

Next we define the problem of mining frequent k-vertex subgraphs. Given a graph $G = (V, E, L, Q)$ and a frequency threshold $\tau \in (0, 1]$, the set $\mathcal{F}_\tau^k \subseteq \mathcal{C}^k$ of frequent k-vertex subgraphs of G with respect to τ is the collection of all k-vertex subgraphs with frequency at least τ, that is

$$\mathcal{F}_\tau^k = \{G_S \mid G_S \in \mathcal{C}^k \text{ and } f(G_S) \geq \tau\}.$$

Problem 2.1. *Given a graph $G = (V, E, L, Q)$, an integer k, and a frequency threshold τ, find the collection $\mathcal{F}_\tau^k$ of frequent k-vertex subgraphs of G.*

Let $p_i = |C_i^k|/|\mathcal{C}^k|$ denote the frequency of isomorphism class i, with $i = 1, \ldots, T_k$. The problem of finding the frequent k-vertex subgraphs requires finding all isomorphism classes C_i^k with $p_i \geq \tau$. Hence, we equivalently have

$$\mathcal{F}_\tau^k = \bigcup_{i \in [1, T_k]} \{G_S \mid G_S \in C_i^k \text{ and } p_i \geq \tau\}.$$

In this paper, our aim is to find an approximation to the collection $\mathcal{F}_\tau^k$ by efficiently estimating p_i, from a uniform sample $\mathcal{S}$ of $\mathcal{C}^k$. We say that a subset $\mathcal{S} \subseteq \mathcal{C}^k$, with $|\mathcal{S}| = M$, is a uniform sample of size M from $\mathcal{C}^k$ if the probability of sampling $\mathcal{S}$ is equal to the probability of sampling any $\mathcal{S}' \subseteq \mathcal{C}^k$ with $|\mathcal{S}'| = M$, i.e., all samples of the same size are equally likely to be produced.

Formally, we want to find an (ϵ, δ)-approximation to $\mathcal{F}_\tau^k$, denoted by $\tilde{\mathcal{F}}_\tau^k(\epsilon, \delta)$ such that

$$\tilde{\mathcal{F}}_\tau^k(\epsilon, \delta) = \bigcup_{i \in [1, T_k]} \{G_S \mid G_S \in C_i^k \cap \mathcal{S}, |\hat{p}_i - p_i| \leq \epsilon/2, p_i \geq \tau\},$$

where $\hat{p}_i$ is the estimation of p_i such that $|\hat{p}_i - p_i| \leq \epsilon/2$ holds with probability at least $1 - \delta$. In practice, the collection $\tilde{\mathcal{F}}_\tau^k(\epsilon, \delta)$ of approximate frequent patterns is computed from a sample $\mathcal{S} \subseteq \mathcal{C}^k$.

The problem of approximate frequent subgraph mining can now be formulated as follows.

Problem 2.2. *Given a graph $G = (V, E, L, Q)$, a frequency threshold τ, a small integer k, and constants $0 < \epsilon, \delta < 1$, find the collection $\tilde{\mathcal{F}}_\tau^k(\epsilon, \delta)$ that is an (ϵ, δ)-approximation to $\mathcal{F}_\tau^k$.*

We focus on the dynamic case with vertex and edge additions and insertions. As discussed above, at each time t we consider the $G^t = (V^t, E^t)$ that results from all vertex and edge operations. Our goal is to maintain the approximate collection of frequent subgraphs $\tilde{\mathcal{F}}_\tau^k(\epsilon, \delta)$ at each time t without having to recompute it from scratch after each addition or deletion.

[1] Notice that the value of T_k is simply determined by k, $|L|$, and $|Q|$.

In the following problem definition we assume that vertex/edge labels are specified when a vertex/edge is added in the graph stream and they do not change afterwards. We make this assumption without loss of generality, as a vertex/edge label change can be simulated by a vertex/edge deletion followed by an addition of the same vertex/edge with different label.

Problem 2.3. *Given an evolving graph $G^t = (V^t, E^t, L, Q)$, a frequency threshold τ, a small integer k, and constants $0 < \epsilon, \delta < 1$, maintain an approximate collection of frequent subgraphs $\tilde{\mathcal{F}}_\tau^k(\epsilon, \delta)$ at each time t.*

3 ALGORITHMS

This section describes the proposed algorithms, which are based on subgraph sampling. We present two algorithms, both of which are based on two components: a reservoir of samples and an exploration procedure. The goal of the reservoir is to capture the changes to already sampled connected k-subgraphs.[2] The goal of the exploration procedure is to include newly (dis)connected k-subgraphs into the sample. This separation of concerns allows the algorithm to minimize the amount of work per sample, e.g., by avoiding computation of expensive minimum DFS codes for the corresponding patterns [39].

The base algorithm requires to enumerate, at each time t, every newly (dis)connected k-subgraph at least once, by performing a *neighborhood exploration* of the updated edge. We show how to improve this algorithm by avoiding to materialize all the subgraphs via a *skip optimization*. This optimization enables picking subgraphs into the sample without having to list them all. We also propose an additional heuristic to speed up the neighborhood exploration. We provide an efficient implementation for the case $k = 3$, and describe how it generalizes to values $k > 3$ (although not as efficiently).

3.1 Incremental streams

We begin by describing our algorithm for maintaining a uniform sample $\mathcal{S}$ of fixed-size M of k-subgraphs of G^t for incremental streams (only edge addition). The algorithm relies on *reservoir sampling* [35] to ensure the uniformity of the sample $\mathcal{S}$.

The addition of an edge $(u, v) \notin E^{t-1}$ at time t affects only the subgraphs in the local neighborhoods up to $N_{u,h}^{t-1}$ and $N_{v,j}^{t-1}$, where $h + j = k - 2$, i.e., all the connected k-subgraphs that contain u, v, and $h + j$ additional nodes from their neighborhoods, for all admissible values of $h, j \geq 0$. Therefore, a uniform sample $\mathcal{S}$ of subgraphs can be maintained by iterating through the subgraphs in the neighborhood of the newly inserted edge. In particular, consider the addition of an edge (u, v) at time t. Let $H \subseteq N_{u,h}^{t-1} \cup N_{v,j}^{t-1}$ be a subset of vertices, for some h and j, such that $h+j = k-2$, $\{u, v\} \in H$, $|H| = k$. There are two possible cases: (i) if H is connected in G^{t-1}, a modified subgraph $H' = H \cup \{(u, v)\}$ is formed in G^t; (ii) if H is not connected in G^{t-1}, and $H' = H \cup \{(u, v)\}$ is connected in G^t, H' is a newly formed connected k-subgraph in G^t.

Example for $k = 3$. Assume an edge (u, v) arrives at time t. For case (i) to hold, there should be some $w \in N_u^{t-1} \cap N_v^{t-1}$ for which the edge (u, v) closes the wedge $\wedge = \{(u, w), (w, v)\}$ at G^{t-1}, forming a new triangle $\Delta = \wedge \cup \{(u, v)\}$ in G^t. For case (ii) to hold, there must be some $w \in N_u^{t-1}$ (or $w \in N_v^{t-1}$), for which a new wedge $\{(u, v), (u, w)\}$ (respectively, $\{(u, v), (v, w)\}$) is formed in G^t. □

When a modified subgraph H' is formed in G^t, if the previously connected subgraph $H = H' \setminus \{(u, v)\}$ is present in $\mathcal{S}$, we update the sample by substituting H' with H. Otherwise, we ignore the modified subgraph. Given that the elements in the sample are induced connected subgraphs, this operation is equivalent to maintaining the sample up-to-date.

Conversely, when a new connected k-subgraph H' is formed in G^t, we can be sure that it appears at time t for the first time. Therefore, we use the standard reservoir sampling algorithm as follows: If $|\mathcal{S}| < M$, we directly add the new subgraph H' to the sample $\mathcal{S}$. Otherwise, if $|\mathcal{S}| = M$, we remove a randomly selected subgraph in $\mathcal{S}$ and insert the new one H' with probability M/N, where M is the upper bound on the sample size and N is the total number of (valid) k-subgraphs *encountered* since $t = 0$.[3] The modification of existing subgraphs in G^t (i.e., case (i)) does not affect N, since, by definition, they replace the previous subgraphs which were already present in G^{t-1}. Therefore, the only increase in the number N of subgraphs occurs in the case of new connected k-subgraph formations in G^t (i.e., case (ii)).

Algorithm 1 shows the pseudocode for incremental streams. Next, we show that the sample $\mathcal{S}$ maintained by Algorithm 1 is uniform at any given time t.

Claim 3.1. *Algorithm 1 ensures the uniformity of the sample $\mathcal{S}$ at any time t.*

Proof. To show that $\mathcal{S}$ is uniform, we need to consider two cases: (i) the inserted edge modifies an existing k-subgraph; (ii) the inserted edge forms a newly connected k-subgraph.

For the case of new subgraph formation, the uniformity property directly holds as it leverages the standard reservoir sampling algorithm. Now, we show that the uniformity property holds when a subgraph is modified.

Assume the edge $(u, v) \notin E^{t-1}$ is inserted at time t and let H denote the invalidated subgraph that is modified as $H' = H \cup \{(u, v)\}$ at time t. Let $\mathcal{S}'$ denote the sample after the invalidation of H and the formation of H'. For the sample to be truly uniform, the probability that $H' \in \mathcal{S}'$ should be equal to M/N, conditioned on $\mathcal{S} = M < N$ (conditioning on $\mathcal{S} = N < M$ is trivial since every k-subgraph of G^t would then be deterministically included in $\mathcal{S}'$). Now, given that $Pr(H \in \mathcal{S}) = M/N$, we have that

$$Pr(H' \in \mathcal{S}') = Pr(H \in \mathcal{S}, H' \in \mathcal{S}') + Pr(H \notin \mathcal{S}, H' \in \mathcal{S}')$$
$$= \frac{M}{N} \cdot 1 + \left(1 - \frac{M}{N}\right) \cdot 0 = \frac{M}{N},$$

hence uniformity is preserved. □

3.2 Fully dynamic streams

In this section we describe our algorithm for maintaining a uniform sample $\mathcal{S}$ of fixed size M for fully-dynamic edge streams (edge

[2] Hereafter, we simply refer to a k-vertex induced subgraph as k-subgraph.

[3] Note that the addition of an edge (u, v) translates to partially-dynamic k-subgraph streams in which the k-subgraphs are subject to addition and deletion operations, while k-cliques are subject to addition-only operations. Thus, we can impose, without loss of generality, an order of operation during the exploration of the neighborhood of the inserted edge.

Algorithm 1 Algorithm for Incremental Stream

```
N ← 0, S ← ∅
M ← log(T_k/δ) · (4 + ε)/ε^2
procedure addEdge(t, (u, v))
    for h ∈ [0, k − 2] do
        j ← k − 2 − h
        for H ⊆ N^{t−1}_{u,h} ∪ N^{t−1}_{v,j} do
            if H is connected in G^{t−1} then
                if H ∈ S then
                    H' ← H ∪ {(u, v)}
                    Replace(S, H, H')                ▷ replace H with H'
            else
                H' ← H ∪ {(u, v)}                    ▷ H' is connected in G^t
                ReservoirSampling(H', S, M, N)
procedure Replace(S, G_R, G_S)
    S ← S \ {G_R}
    S ← S ∪ {G_S}
```

Algorithm 2 Fully-Dynamic-Edge Stream

```
N ← 0, S ← ∅, c_1 ← 0, c_2 ← 0
M ← log(T_k/δ) · (4 + ε)/ε^2
procedure addEdge(t, (u, v))
    for h ∈ [0, k − 2] do
        j ← k − 2 − h
        for H ⊆ N^{t−1}_{u,h} ∪ N^{t−1}_{v,j} do
            if H is connected in G^{t−1} then
                if H ∈ S then
                    H' ← H ∪ {(u, v)}
                    Replace(S, H, H')                ▷ replace H with H'
            else
                H' ← H ∪ {(u, v)}                    ▷ H' is newly connected in G^t
                RandomPairing(H', S, M)
procedure deleteEdge(t, (u, v))
    for h ∈ [0, k − 2] do
        j ← k − 2 − h
        for H ⊆ N^{t−1}_{u,h} ∪ N^{t−1}_{v,j} do
            if H is still connected in G^t then
                if H ∈ S then
                    H' ← H \ {(u, v)}
                    Replace(S, H, H')                ▷ replace H with H'
            else
                if H ∈ S then
                    S ← S \ H
                    c_1 ← c_1 + 1
                else
                    c_2 ← c_2 + 1
                N ← N − 1
procedure RandomPairing(G_S, S, M)
    if c_1 + c_2 = 0 then
        ReservoirSampling(G_S, S, M, N)
    else
        if uniform() < c_1/(c_1+c_2) then
            S ← S ∪ G_S
            c_1 ← c_1 − 1
        else
            c_2 ← c_2 − 1
```

insertions and deletions). Our algorithm relies on *random pairing* (RP) [14], a sampling scheme that extends traditional reservoir sampling for evolving data streams, in which elements are subject to both addition and deletion operations.

We first give a brief background on the RP scheme. In RP, the uniformity of the sample is guaranteed by randomly pairing an inserted element with an uncompensated "partner" deletion, without necessarily keeping the identity of the partner. At any time, there can be 0 or more uncompensated deletions, denoted by d, which is equal to the difference between the cumulative number of insertions and the cumulative number of deletions. The RP algorithm maintains (i) a counter c_1 that records the number of uncompensated deletions in which the deleted element was in the sample, (ii) a counter c_2 that records the number of uncompensated deletions in which the deleted element was not in the sample, hence, $d = c_1 + c_2$. When $d = 0$, i.e., when there are no uncompensated deletions, inserted elements are processed as in standard reservoir-sampling. When $d > 0$, the algorithm flips a coin at each inserted element and includes it in the sample with probability $c_1/(c_1 + c_2)$, otherwise it excludes it from the sample (and decreases c_1 or c_2 as appropriate).

Next, we describe our adaptation of the RP scheme for fully-dynamic edge streams, which translate to fully-dynamic k-subgraph streams. First, remember that the incremental stream translates to an incremental k-subgraph stream, in which connected k-subgraphs are only added (the first time they are created) or modified (when new induced edges arrive).

In the case of fully-dynamic edge streams, the k-connected subgraph stream is also subject to addition and deletion operations, as we explain next. The events of interest regarding the addition of an edge have been discussed extensively in the previous section, hence we do not repeat it here. Consider the deletion of an edge $(u, v) \in E^{t-1}$ at time t, and a subgraph $H \subseteq N^{t-1}_{u,h} \cup N^{t-1}_{v,j}$ in G^{t-1}, with $h + j = k - 2$. The effect of the edge deletion is the following: either (i) the vertices of H remain connected, hence, H is replaced by a new subgraph H' in G^t; or (ii) H gets disconnected, hence H does not exist in G^t. The first case corresponds to a modification of an existing connected k-subgraph. As such, it does not cause an addition or deletion in the subgraph stream.

Example for k=3. In the case a triangle Δ in G^{t-1} that contains an edge (u, v) deleted at time t, if $\Delta \in \mathcal{S}$, we modify the corresponding induced subgraph into a subgraph $\wedge = \Delta \setminus \{(u, v)\}$. □

The second case corresponds to a deletion of a subgraph in the stream. To handle this case, our sampling strategy follows the RP scheme. In the case that a subgraph H in G^{t-1} is deleted due to the deletion of edge (u, v) at time t, if $H \in \mathcal{S}$, we increment the counter c_1, otherwise we increment the counter c_2. In the case that a new subgraph H' is formed in G^t due to the addition of edge (u, v) at time t, we include it in $\mathcal{S}$ with probability $c_1/(c_1 + c_2)$. The approach is shown in Algorithm 2. Next, we show that the sample $\mathcal{S}$ maintained by Algorithm 2 is uniform at any given time t.

Claim 3.2. *Algorithm 2 ensures the uniformity of the sample $\mathcal{S}$ at any time t.*

Proof. To show that $\mathcal{S}$ is uniform, we need to consider four cases: (i) added edge forms a newly connected subgraph; (ii) deleted edge disconnects a subgraph; (iii) added edge modifies an existing a subgraph; (iv) deleted edge modifies an existing a subgraph. For cases (i) and (ii), the correctness follows from RP hence we only show the correctness in cases (iii) and (iv). Assume the edge $(u, v) \in E^{t-1}$ is deleted (resp. added) at time t. Let H' denote the new subgraph due to the deletion (addition) of the edge, so that $H' = H \setminus \{(u, v)\}$ (resp. $H' = H \cup \{(u, v)\}$). Let $\mathcal{S}'$ denote the sample after the invalidation of H and the formation of H'. Recall that N remains unchanged since H' replaces H in G^t. Given that the random pairing scheme guarantees uniformity of the sample at each time instance independently from the current value of d [13], we have $Pr(H \in \mathcal{S}) = |\mathcal{S}|/N$. For the sample to be truly uniform, the probability that $H' \in \mathcal{S}'$ should also be equal to $|\mathcal{S}|/N$ since the values

Algorithm 3 Compute-$\mathcal{W}$, N° of new connected k-subgraphs

```
1: procedure Compute-W(t, (u, v))
2:     W ← 0
3:     for h ∈ [0, k − 2] do
4:         j ← k − 2 − h
5:         V_h ← N^{t−1}_{u,h} \ N^{t−1}_{v,j}
6:         V_j ← N^{t−1}_{v,j} \ N^{t−1}_{u,h}
7:         x ← |{G_S = (S, E(S)) : u ∈ S, |S| = h + 1, S ⊆ V_h, E(S) ⊆ E(V_h)}|
8:         y ← |{G_S = (S, E(S)) : v ∈ S, |S| = j + 1, S ⊆ V_j, E(S) ⊆ E(V_j)}|
9:         W ← W + x · y
```

of both N and $\mathcal{S}$ remain unchanged as we either replace H with H' in $\mathcal{S}$ or we ignore H' if $H \notin \mathcal{S}$, hence $|\mathcal{S}|$ remains unchanged. Thus, we have,

$$\begin{aligned} Pr\left(H' \in \mathcal{S}'\right) &= Pr\left(H \in \mathcal{S}\right) \cdot Pr\left(H' \in \mathcal{S}' \mid H \in \mathcal{S}\right) \\ &+ Pr\left(H \notin \mathcal{S}\right) \cdot Pr\left(H' \in \mathcal{S}' \mid H \notin \mathcal{S}\right) \\ &= \frac{|\mathcal{S}|}{N} \cdot 1 + \left(1 - \frac{|\mathcal{S}|}{N}\right) \cdot 0 = \frac{|\mathcal{S}|}{N}, \end{aligned}$$

hence uniformity is preserved. □

3.3 Skip optimization

The basic algorithm for incremental streams we described requires to process each subgraph $H \subseteq N^{t-1}_{u,h} \cup N^{t-1}_{v,j}$, for all admissible values of $h, j \geq 0$ s.t. $h + j = k - 2$, to identify among them the newly created k-subgraphs. All these new subgraphs are then provided as input to the standard reservoir sampling algorithm that needs to generate random numbers for each. To reduce the cost of traversing the local neighborhood and generating a random number for each new subgraph, we employ Vitter's acceptance-rejection algorithm that generates skip counters for reservoir sampling [35] as follows: let Z_{RS} be the random variable that denotes the number of rejected subgraphs after the last time a subgraph was inserted to the sample $\mathcal{S}$. Then, the probability that the next z new subgraphs will not be accepted in $\mathcal{S}$ is given by:

$$Pr[Z_{RS} = z] = \frac{M}{N + z + 1} \prod_{z'=0}^{z-1} \left(1 - \frac{M}{N + z' + 1}\right). \tag{1}$$

Thus, rather than identifying all the new subgraphs and calling the reservoir algorithm for each, we can keep a skip counter Z_{RS} that is distributed with the probability mass function given in Eq. (1), and compute its value in constant time using Vitter's acceptance-rejection algorithm for reservoir sampling [35]. Then, based on the value of Z_{RS} that denotes the number of new subgraph insertions we can safely skip, we can decide on the fly whether we should insert into the sample any of the new subgraphs created due to the insertion of edge (u, v). Given that a new k-subgraph $G_S = (S, E(S))$ can be formed only when $E(S) \setminus (u, v)$ is not already an induced subgraph, we can compute the *exact* value of $\mathcal{W}$ as in Algorithm 3. The pseudocode of the optimized algorithm for incremental streams is given in Algorithm 4.

A similar optimization is also possible for fully-dynamic streams by proper adjustment of the skip counter based on the value $d = c_1 + c_2$ of uncompensated deletions . Recall that when $d = 0$, reservoir sampling is effective, hence, we can compute the value of the skip counter Z_{RS} as in the case of incremental streams. When $d > 0$,

Algorithm 4 Optimized Algorithm for Incremental Stream

```
1: N ← 0, S ← ∅, sum ← 0
2: M ← log(T_k/δ) · (4 + ε)/ε^2
3: SkipRS(N, M): skip function as in [35]        ▷ [SkipRS(N, M) = 0 if N < M]
4: procedure addEdge(t, (u, v))
5:     for H ∈ S : u ∈ H ∧ v ∈ H do
6:         H' ← H ∪ {(u, v)}
7:         Replace(S, H, H')                      ▷ replace H with H'
8:     W ← Compute-W(t, (u, v))                   ▷ Algorithm 3
9:     I ← 0
10:    while sum ≤ W do
11:        I ← I + 1
12:        Z_RS ← SkipRS(N, M)
13:        N ← N + Z_RS + 1
14:        sum ← sum + Z_RS + 1
15:    replace I random elements of S with I random subgraphs drawn from N^t_{u,h} ∪
       N^t_{v,j}, ∀h ∈ [0, k − 2], j = k − 2 − h
16:    sum ← sum − W
```

the random-pairing step is effective, for which we adapt Vitter's improvements to the list-sequential sampling [34].

Let Z_{RP} be the random variable that denotes the number of new subgraphs that are not accepted into the sample after the last time a subgraph was deleted (not necessarily from the sample) due to the deletion of an edge. Assume without loss of generality that the deletion of a subgraph was followed by the creation of d new subgraphs due to at least one edge insertion. Following the fact that the new elements that random pairing includes into the sample form a uniform random sample of size c_1 among d new elements [14], the probability that the random pairing will not accept the next z new subgraphs in $\mathcal{S}$ is given by:

$$Pr[Z_{RP} = z] = \frac{c_1}{d - z} \prod_{z'=0}^{z-1} \left(1 - \frac{c_1}{d - z'}\right). \tag{2}$$

Thus, after each edge deletion, we can compute in constant time the value of skip counter Z_{RP} for random pairing using acceptance-rejection algorithm for list-sequential sampling [34] and decide on the fly whether and how many we should insert into the sample any of the new $\mathcal{W}$ subgraphs created in the pairing step. The algorithm to compute the exact number $\mathcal{D}$ of deleted induced subgraphs when an edge (u, v) is deleted at time t is similar to the computation of $\mathcal{W}$, but operates on the neighborhoods at time t instead of time $t - 1$ (omitted due to space constraints). The pseudocode of the optimized algorithm is given in Algorithm 5.

3.4 Derivation for sample size

Now we provide a lower bound on the size of the sample $\mathcal{S}$ such that $\tilde{\mathcal{F}}^k_\tau(\epsilon, \delta)$ computed on $\mathcal{S}$ provides an (ϵ, δ)-approximation to $\mathcal{F}^k_\tau$.

Lemma 3.1. *Suppose that $|\mathcal{S}| = M$ satisfies*

$$M \geq \log\left(\frac{T_k}{\delta}\right) \cdot \frac{(4 + \epsilon)}{\epsilon^2} \tag{3}$$

Then, for any isomorphism class $i \in [1, T_k]$, $|\hat{p_i} - p_i| \leq \epsilon/2$ holds with probability at least $1 - \delta/T_k$:

Proof. Let X_i denote an indicator random variable that equals 1 if a randomly sampled subgraph G_S from C^k belongs in C^k_i and 0 otherwise, $\forall i \in [1, T_k]$. Notice that $X_i \sim Bernoulli(p_i)$. W.l.o.g, let G_j, $j \in [1, M]$ denote the j-th subgraph in $\mathcal{S}$ for an arbitrary

Algorithm 5 Optimized Algorithm for Fully-Dynamic-Edge Stream

1: $N \leftarrow 0$, $\mathcal{S} \leftarrow \emptyset$, $c_1 \leftarrow 0$, $c_2 \leftarrow 0$
2: $M \leftarrow \log(T_k/\delta) \cdot (4+\epsilon)/\epsilon^2$
3: SkipRS(N, M): skip function as in [35] ▷ [SkipRS(N, M) = 0 if $N < M$]
4: SkipRP(c_1, $c_1 + c_2$): skip function as in [34]
5: sum1 $\leftarrow$ 0, sum2 $\leftarrow$ 0
6: **procedure** addEdge(t, (u, v))
7: **for** $H \in \mathcal{S} : u \in H \wedge v \in H$ **do**
8: $H' \leftarrow H \cup \{(u, v)\}$
9: Replace($\mathcal{S}$, H, H') ▷ replace H with H'
10: $\mathcal{W} \leftarrow$ Compute-$\mathcal{W}(t, (u, v))$
11: **if** $c_1 + c_2 = 0$ **then**
12: $I \leftarrow 0$
13: **while** sum1 $\leq \mathcal{W}$ **do**
14: $I \leftarrow I + 1$
15: $N \leftarrow N + Z_{RS} + 1$
16: $Z_{RS} \leftarrow$ SkipRS(N, M)
17: sum1 $\leftarrow$ sum1 $+ Z_{RS} + 1$
18: replace I random elements of $\mathcal{S}$ with I random subgraphs drawn from $N^t_{u,h} \cup N^t_{v,j}$, $\forall h \in [0, k-2]$, $j = k-2-h$
19: sum1 $\leftarrow$ sum1 $- \mathcal{W}$
20: **else**
21: $I \leftarrow 0$, $sum2 \leftarrow 0$
22: **while** c1+c2 > 0 and $sum2 < W$ **do**
23: $I \leftarrow I + 1$
24: $c_1 \leftarrow c_1 - 1$
25: $Z_{RP} \leftarrow$ SkipRP(c_1, $c_1 + c_2$)
26: $c_2 \leftarrow c_2 - Z_{RP}$ ▷ [$c_2 = 0$ if $c_2 < 0$]
27: sum2 $\leftarrow$ sum2 $+ Z_{RP} + 1$
28: replace I random elements of $\mathcal{S}$ with I random subgraphs drawn from $N^t_{u,h} \cup N^t_{v,j}$, $\forall h \in [0, k-2]$, $j = k-2-h$
29: $\mathcal{W} \leftarrow \mathcal{W} - sum2$
30: **if** $W > 0$ **then**
31: Jump to line 12
32: **procedure** deleteEdge(t, (u, v))
33: **for** $H \in \mathcal{S} : u \in H \wedge v \in H$ **do**
34: **if** H is still connected in G^t **then**
35: $H' \leftarrow H \setminus \{(u, v)\}$
36: Replace($\mathcal{S}$, H, H') ▷ replace H with H'
37: **else**
38: $\mathcal{S} \leftarrow \mathcal{S} \setminus H$
39: $c_1 \leftarrow c_1 + 1$
40: $d \leftarrow d +$ Compute-$\mathcal{D}(t, (u, v))$
41: $c_2 \leftarrow d - c_1$
42: $N \leftarrow N - \mathcal{D}$

ordering of the subgraph and let $X_i^1, \cdots, X_i^M$ be iid copies of X_i where each X_i^j denotes the event $\mathbb{1}_{[G_j \in C_i^k]}$.

Using the two-sided Chernoff bounds we have

$$Pr\left(\left|\sum_{j=1}^{M} X_i^j - p_i M\right| \geq \theta M p_i\right) \leq 2\exp\left(-\frac{\theta^2}{2+\theta} \cdot p_i M\right),$$

which implies

$$Pr\left(|\hat{p}_i - p_i| \geq \theta p_i\right) \leq 2\exp\left(-\frac{\theta^2}{2+\theta} \cdot p_i M\right).$$

Now, let $\epsilon = 2p_i\theta$. Substituting $\theta = \epsilon/(2p_i)$ we have

$$Pr\left(|\hat{p}_i - p_i| \geq \epsilon/2\right) \quad \leq \quad 2\exp\left(-\frac{\epsilon^2/4}{2p_i + \epsilon/2} \cdot M\right).$$

To obtain a failure probability of at most δ/T_k for each isomorphism class $i \in [1, T_k]$, we should have:

$$2\exp\left(-\frac{\epsilon^2/4}{2p_i + \epsilon/2} \cdot M\right) \leq \frac{\delta}{T_k}.$$

Rearranging the terms we obtain:

$$M \geq \log\left(\frac{T_k}{\delta}\right) \cdot \frac{2p_i + \epsilon/2}{(\epsilon^2/2)}.$$

As we want this to hold $\forall i \in [1, T_k]$, M should satisfy:

$$M \geq \log\left(\frac{T_k}{\delta}\right) \cdot \frac{2p_{max} + \epsilon/2}{(\epsilon^2/2)},$$

where $p_{max} = \max_{i \in [1, T_k]} p_i$. Using the worst-case $p_{max} = 1$, we obtain the following lower bound on M:

$$M \quad \geq \quad \log\left(\frac{T_k}{\delta}\right) \cdot \frac{(4+\epsilon)}{\epsilon^2}.$$

□

Theorem 3.2. *Given a uniform sample $\mathcal{S} \subseteq C^k$ of size M that satisfies Eq. (3), $\tilde{\mathcal{F}}^k_\tau(\epsilon, \delta)$ provides (ϵ, δ)-approximation to $\mathcal{F}^k_\tau$.*

Proof. Given M that satisfies Eq. (3), using union bound over all T_k estimation failure scenarios, we have $|\hat{p}_i - p_i| \leq \epsilon/2$, for all $i \in [1, T_k]$, with probability at least $1 - \delta$. Then, there should be no $i \in [1, T_k]$ with $p_i \geq \tau$, for which $\hat{p}_i < \tau - \epsilon/2$. Hence, we ensure $\tilde{F}(\mathcal{S}, \tau - \epsilon/2) \subseteq F(C^k, \tau)$ with probability at least $1 - \delta$. Now, assume that there is a subgraph $G_S \in C_i^k$ such that $p_i < \tau - \epsilon$. We have that $\hat{p}_i < \tau - \epsilon/2$, hence, there is no subgraph G_S such that $G_S \notin F(C^k, \tau)$ and $G_S \in \tilde{F}(\mathcal{S}, \tau - \epsilon/2)$, with probability at least $1 - \delta$. □

4 NEIGHBORHOOD EXPLORATION

The skip optimizations allows us to efficiently maintain the uniformity of the sample $\mathcal{S}$ by eliminating the need to test the inclusion of each newly created k-subgraph in the local neighborhood of the inserted edge. However, the skip optimizations require to know the number $\mathcal{W}$ of new k-subgraphs. Unfortunately, exact computation of $\mathcal{W}$ requires costly traversal of the neighborhood of the inserted edge. Moreover, for dynamic streams, the value of the skip counter directly depends on c_1 and c_2, which require to compute the number $\mathcal{D}$ of deleted induced subgraphs after each edge deletion operation. Thus, we resort on efficient methods to approximate the values of $\mathcal{W}$ and $\mathcal{D}$.

To efficiently approximate the value of $\mathcal{W}$ after an edge (u, v) is inserted at time t, we use sketches to estimate $|N^{t-1}_{u,h} \cap N^{t-1}_{v,j}|$ for all possible values of $h \in [0, k-2]$ and $j \in [0, k-2]$. Similarly, to efficiently approximate $\mathcal{D}$ after an edge (u, v) is deleted at time t, we use sketches to estimate $|N^t_{u,h} \cap N^t_{v,j}|$ for all possible values of $h \in [0, k-2]$ and $j \in [0, k-2]$.

Any sketching technique for set-size estimation can be used. For our purpose, we choose to use the bottom-k sketch [11] in conjunction with recently-proposed improved estimators for union and intersections of sketches [32]. A bottom-k sketch uses a hash function $h(\cdot)$ to map elements of a universe into real numbers in $[0, 1]$, and stores the k minimum values in a set. The smaller the k-th stored value is, the larger the size of the original set should be; a simple estimate of the size is given by $\frac{k-1}{\gamma}$, where γ is the largest stored hash value.

In our case, the universe of elements is the set of vertices V^t that belong to the graph at time t. We build a sketch for each vertex $v \in V^t$ that summarizes N^t_v. These sketches can be efficiently

combined to create a sketch for the union of the neighbors of a given vertex while exploring the neighborhood via a breadth first search (BFS).

Bottom-k sketches can easily be built incrementally. When a new edge (u, v) is added, we simply add the hash value of v to the sketch of u if it is smaller than the current maximum, and vice versa. Alas, bottom-k sketches do not directly support deletions. However, traditionally the sketches are used in a streaming setting where memory is the main concern. In our case, the universe of elements already resides in memory (i.e., the vertices of the graph), and our goal is to improve the speed of computation of Algorithm 3 and its counterpart for deletion. Therefore, we can easily store the global hash value of each vertex to be used for sketching. Then, we can implement the sketch by using a pair of min-heap/max-heap. The max-heap A^+ has bounded size and contains the hash values of the corresponding bottom-k vertices. The min-heap A^- contains the hash values of the rest of the neighborhood. Whenever an edge (u, v) is deleted, if $h(v) \in A^-$ we remove the value from A^- but the sketch remains unchanged; if $h(v) \in A^+$ we remove the value from A^+, and we also transfer the minimum value from A^- to A^+ to maintain the fixed size of the sketch.

4.1 Efficient implementation of $\mathcal{S}$

The reservoir sample $\mathcal{S}$ needs to support two main access operations efficiently: (1) Random access (to replace subgraphs in the sample, for reservoir sampling); (2) Access by vertex id (to identify modified subgraphs, as in Algorithm 4).

In order to support both operations in constant time, we resort to an array for the basic random access, supplemented by hash-based indexes for the access by vertex id.

The basic array is straightforward to implement, as the size of the sample M is fixed, and the size of its element is constant k^2 (to store both vertices and edges). On top of this basic array, we maintain and index $\mathcal{I} : V \to \{S \subset V\}$ such that $v \to S$ for all $v \in S$ and all $S \in \mathcal{S}$. That is, we have a pointer from each vertex part of a subgraph in the sample, to the set of subgraphs containing it. Therefore, when an edge (u, v) is modified at time t (either added or deleted), retrieving the set of potentially affected subgraphs takes constant time. For each potentially affected subgraph, checking whether it is actually affected also takes constant time: for a subgraph $S \in \mathcal{I}(u)$ (respectively, $S \in \mathcal{I}(v)$) we simply need to check whether $v \in S$ (respectively, $u \in S$). If so, the subgraph needs to be updated, and so the corresponding counters for its pattern.

4.2 Time complexity

Our proposed algorithms contain two components: an exploration procedure and a reservoir of samples. The addition of an edge $(u, v) \notin E^{t-1}$ at time t affects only the subgraphs in the local neighborhoods up to $N^{t-1}_{u,h}$ and $N^{t-1}_{v,j}$, where $h + j = k - 2$. The base algorithms, for both incremental and fully dynamic settings, iterate through the set of subgraphs in the local neighborhoods up to $N^{t-1}_{u,h}$ and $N^{t-1}_{v,j}$. Moreover, the subgraphs are added into the reservoir in constant time, i.e., $O(1)$ per subgraph, which implies that the running time of the algorithms are propotional to the expensive exploration procedure, i.e., $O(N^{t-1}_{u,h} \cup N^{t-1}_{v,j})$. The skip optimization improves the execution time by avoiding materializing and computing the expensive DFS code for many subgraphs, but does not change its worst case upper bound.

Table 1: Datasets used in the experiments.

Dataset	Symbol	$\|V\|$	$\|E\|$	$\|L\|$
Patents	PT	3M	14M	37
Youtube	YT	4.6M	43M	108

5 EXPERIMENTS

We conduct an extensive empirical evaluation of the proposed algorithms, and provide a comparison with the existing solutions. In particular, we answer the following interesting questions:

Q1: What is the quality of frequent patterns for incremental streams?

Q2: What is the quality of frequent patterns for dynamic streams?

Q3: What is the performance in terms of average update time?

5.1 Experimental setup

Datasets. Table 1 shows the graphs used as input in our experiments. All datasets used are publicly available. Patent (PT) [15] contains citations among US Patents from January 1963 to December 1999; the label of a patent is the year it was granted. YouTube (YT) [10] lists crawled videos and their related videos posted from February 2007 to July 2008. The label is a combination of a video's rating and length. The streams are generated by permuting the edges in a random order.

Metrics. We use the following metrics to evaluate the quality of all the algorithms:

- *Average Relative Error (RE):* measures how close the estimation of the frequency of the subgraph patterns compared to the ground truth. For the set of patterns $\mathcal{F}^k_\tau$, the average RE of the estimation is defined as $\frac{1}{T_k} \sum_{i=1}^{T_k} \frac{|\hat{p}_i - p_i|}{p_i}$.
- *Precision:* measures the fraction of frequent subgraph patterns among the ones returned by the algorithm.
- *Recall:* measures the fraction of frequent subgraph patterns returned by the algorithm over all frequent subgraphs (as computed by the exact algorithm).

Additionally, we evaluate the efficiency of the algorithms by reporting the average update time. We provide an extensive comparison of all the algorithms for $k = 3$. We report the results of experiments averaged over 5 runs.

Algorithms. We use two baselines. Exact counting (**EC**) performs exhaustive exploration of the neighborhood of the updated edge, and counts all possible subgraph patterns. Edge reservoir (**ER**) is a scheme inspired by Stefani et al. [31], which maintains a reservoir of edges during the dynamic edge updates. The edge reservoir is used to estimate the frequency of subgraph patterns by applying the appropriate correcting factor for the sampling probability of each pattern. We compare these baselines with our proposed algorithms, subgraph reservoir (**SR**) and its optimized version (**OSR**). The size of the subgraphs reservoir is set as in Section 3.4. Unless otherwise specified, we fix $\epsilon = 0.01$ and $\delta = 0.1$. To have a fair comparison with **ER**, following the evaluation of Stefani et al. [31], we set the size of edge reservoir as the maximum number of edges used in

the subgraph reservoir, averaged over 5 runs. Note that **EC** and **ER** algorithms are more competitive than any offline algorithm, e.g., GraMi [12], which require processing the whole graph upon any update. **EC** takes less than 2×10^{-5} seconds to process an edge of the PT dataset, on average, while one execution of GraMi on the same dataset takes around 30 seconds, which is several orders of magnitude larger, and we need to execute it once per edge.

Experimental environment. We conduct our experiments on a machine with 2 Intel Xeon Processors E5-2698 and 128GiB of memory. All the algorithms are implemented in Java and executed on JRE 7 running on Linux. The source code is available online.[4]

5.2 Incremental case

We first evaluate our proposed algorithm on incremental streams. Starting from an empty graph, we add one edge per timestamp, for both the PT and YT datasets, and run the algorithms for several values of the frequency threshold τ.

Figure 1 shows the results. For the PT dataset, the three algorithms behave similarly in terms of RE. The subgraph versions offer slightly higher precision at the expense of decrease in recall. However, for the highest frequency threshold, we see a marked deterioration of the performance of **ER**. This behavior is a result of higher variance in **ER** due to non-uniform subgraph-sampling probabilities. Conversely, for YT, both versions of the subgraph reservoir algorithm provide superior results in terms of average relative error. Considering YT is the larger and more challenging dataset (in terms of number of labels), this result shows the power of subgraph sampling. The improved estimation performance translates to much higher precision for **SR** and **OSR** compared to **ER**. The recall of all the algorithms are very similar. Overall, the results indicate that **ER** generates a larger number of false positives in the result set, while **SR** and **OSR** are able to avoid such errors while at the same time still having a low false-negative rate.

5.3 Fully-dynamic case

Now, we proceed to evaluate the algorithms for fully-dynamic streams. To produce edge deletions, we execute the algorithms in a sliding window model. This model is of practical interest as it allows to observe recent trends in the stream. We evaluate the algorithms for the YT dataset, and use a sliding window of size 10M. We choose a sliding window large enough so so that the number of edges (subgraphs) do not fit in the edge (subgraph) reservoir, otherwise both algorithms are equivalent to exact counting. We only report the results for YT dataset, as the result for the PT dataset are similar to the incremental case.

Figure 2 contains the results for YT dataset. **ER** obtains higher relative error compared to **SR**, and poor precision and recall. **SR** is clearly the best performing algorithm in terms of accuracy, however, as we show next, it pays in terms of efficiency. **OSR** has consistently better accuracy than **ER**, although the approximations it deploys introduce some errors. This effect is more evident for larger frequency thresholds, where the precision drops noticeably.

5.4 Performance

Lastly, we evaluate the algorithms in terms of the average update time for both incremental and fully-dynamic streams on PT and YT datasets. The size of the sliding window is 10M for the fully-dynamic streams. Figure 3 reports the results of the experiments which show that both **SR** and **OSR** provide significant performance gains compared to the **EC** while they are both outperformed by **ER**. However, given the superior accuracy of **SR** and **OSR** compared to **ER**, it can be easily observed that **OSR** provides a good trade-off between accuracy and efficiency.

[4] https://github.com/anisnasir/frequent-patterns

6 RELATED WORK

Triangle counting. Exact and approximate triangle counting in static graphs has attracted a great deal of attention. We refer the reader to the survey by Latapy [26] for a comprehensive treatment of the topic, and include only related work on approximate triangle counting in a streaming setting. Tsourakakis et al. [33] proposed triangle sparsifiers to approximate the triangle counts with a single pass of the graph, hence, the technique can also be applied to incremental streams. Pavan et al. [28] and Jha et al. [20] proposed sampling a set of connected paths of length for approximately counting the triangles in incremental streams. Lim and Kang [27] proposed an algorithm based on Bernoulli sampling of edges for incremental streams, in which the edges are kept in the sample with a fixed user-defined probability. Recently, Stefani et al. [31] proposed an algorithm for fully-dynamic streams via reservoir sampling [35] and random pairing [13].

General k-vertex graphlet counting. Approximate counting of 3-, 4-, 5-vertex graphlets in static graphs has received much more attention than exact counting, which has an exponential cost. Most of the literature on approximate counting of graphlets uses random-walks to collect a uniform sample of graphlets on static graphs [3, 8, 16, 37]. Alternatively, Bressan et al. [6] proposed a color coding based scheme for estimating k-vertex graphlet statistics. Unlike the static case, approximating graphlet statistics in a streaming setting has received much less attention, and the literature is limited to incremental streams for $k > 3$. Wang et al. [38] are the first to propose an algorithm that estimates graphlet statistics from a uniform sample of edges in incremental streams. A recent work by Chen and Lui [9] examines approximate counting of graphlets in incremental streams for different choice of edge sampling and probabilistic counting methods.

Transactional FSM. Inokuchi et al. [19] introduced the problem of FSM in the transactional setting, where the goal is to mine all the frequent subgraphs on a given dataset of many, usually small, graphs. Following [19], a good number of algorithms for this task were provided [18, 23, 39]. The transactional FSM setting is similar to the one considered by frequent-itemset mining [17], allowing to reuse many existing results, thanks to the anti-monotonicity of its support metric. In addition to the exact mining approaches, a line of work has studied the approximate mining of frequent subgraphs by MCMC sampling from the space of graph patterns [2, 30] with efficient pruning strategies based on anti-monotonicity of the support metric. For a comprehensive treatment, see the survey by Jiang et al. [21].

Single-Graph FSM. Kuramochi and Karypis [25] proposed an algorithm for exact mining of all frequent subgraphs in a given static graph that enumerates all the isomorphisms of the given graph and relies on the maximum-independent set (MIS) metric whose computation is NP-Complete. Elseidy et al. [12] proposed an apriori-like

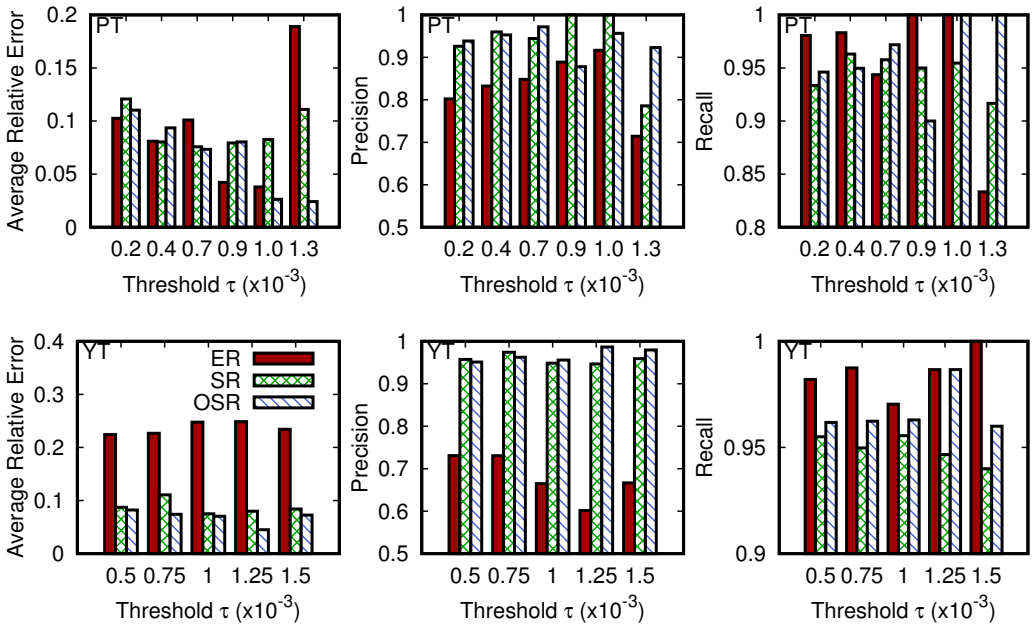

Figure 1: Relative error, precision, and recall for incremental streams on PT and YT datasets, for different values of threshold τ.

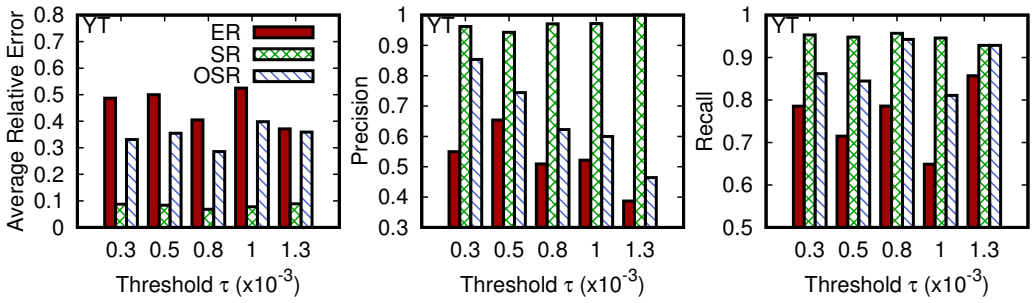

Figure 2: Relative error, precision, and recall for fully-dynamic stream on YT dataset, for different values of threshold τ.

algorithm for exact mining of all frequent subgraphs based on the MIS metric from a given static graph. Apart from the exact mining algorithms, a line of work focused on approximate mining of frequent subgraphs in a given static graph. Kuramochi and Karypis [24] proposed a heuristic approach that prunes largely the search space however discovers only a small subset of frequent subgraphs without provable guarantees. Chen et al. [7] uses an approximate version of the MIS metric, allowing approximate matches during the pruning. Khan et al. [22] propose proximity patterns, which, by relaxing the connectivity constraint of subgraphs, identify frequent patterns that cannot be found by other approaches.

While the discussed work for solving FSM problem on a static graph are promising, none of them are applicable to streaming graphs. The closest to our setting is the work by Ray et al. [29] which consider a single graph with continuous updates, however their approach is a simple heuristic applicable only to incremental streams and without provable guarantees. Likewise, Abdelhamid et al. [1] consider an analogous setting, and propose an exact algorithm which borrows from the literature on incremental pattern mining. The algorithm keeps track of "fringe" subgraph patterns, which are around the frequency threshold, and all their possible expansions/contractions (by adding/removing one edge). While the algorithm uses clever indexing heuristics to reduce the runtime, an exact algorithm still needs to enumerate and track an exponential number of candidate subgraphs. Finally, Borgwardt et al. [5] look at the problem of finding dynamic patterns in graphs, i.e., patters over a graph time series, where persistence in time is a key element of the pattern. By transforming the time series of a labeled edge

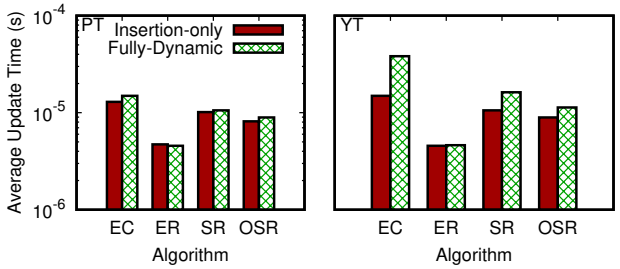

Figure 3: Average update time for incremental and fully-dynamic streams on PT and YT datasets.

into a binary string, the authors are able to leverage suffix trees and string-manipulation algorithms to find common substrings in the graph. While dynamic graph patterns capture the time-series nature of the evolving graph, in our streaming scenario, only the latest instance of the graph is of interest, and the graph patterns found are comparable to the ones found for static graphs.

7 CONCLUSION

We initiated the study of approximate frequent-subgraph mining (FSM) in both incremental and fully-dynamic streaming settings, where the edges can be arbitrarily added or removed from the graph. For each streaming setting, we proposed algorithms that can extract a high-quality approximation of the frequent k-vertex subgraph patterns, for a given threshold, at any given time instance, with high probability. Our algorithms operate by maintaining a uniform sample of k-vertex subgraphs at any time instance, for which we provide theoretical guarantees. We also proposed several optimizations to our algorithms that allow achieving high accuracy with improved execution time. We showed empirically that the proposed algorithms generate high-quality results compared to natural baselines.

Acknowledgements. Cigdem Aslay and Aristides Gionis are supported by three Academy of Finland projects (286211, 313927, and 317085), and the EC H2020 RIA project "SoBigData" (654024).

REFERENCES

[1] Ehab Abdelhamid, Mustafa Canim, Mohammad Sadoghi, Bishwaranjan Bhattacharjee, Yuan-Chi Chang, and Panos Kalnis. 2017. Incremental Frequent Subgraph Mining on Large Evolving Graphs. *TKDE* 29, 12 (2017), 2710–2723.
[2] Mohammad Al Hasan and Mohammed J Zaki. 2009. Output space sampling for graph patterns. *PVLDB* 2, 1 (2009), 730–741.
[3] Mansurul A Bhuiyan, Mahmudur Rahman, and M Al Hasan. 2012. Guise: Uniform sampling of graphlets for large graph analysis. In *ICDM*. 91–100.
[4] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Ricard Gavaldà. 2011. Mining frequent closed graphs on evolving data streams. In *KDD '11*. 591–599.
[5] Karsten M Borgwardt, Hans-Peter Kriegel, and Peter Wackersreuther. 2006. Pattern mining in frequent dynamic subgraphs. In *ICDM*. 818–822.
[6] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2017. Counting Graphlets: Space vs Time. In *WSDM*. 557–566.
[7] Chen Chen, Xifeng Yan, Feida Zhu, and Jiawei Han. 2007. gapprox: Mining frequent approximate patterns from a massive network. In *ICDM*.
[8] Xiaowei Chen, Yongkun Li, Pinghui Wang, and John Lui. 2016. A general framework for estimating graphlet statistics via random walk. *PVLDB* 10, 3 (2016), 253–264.
[9] Xiaowei Chen and John Lui. 2017. A unified framework to estimate global and local graphlet counts for streaming graphs. In *ASONAM*. 131–138.
[10] Xu Cheng, Cameron Dale, and Jiangchuan Liu. 2008. Statistics and social network of youtube videos. In *IWQoS*. 229–238.
[11] Edith Cohen and Haim Kaplan. 2007. Summarizing data using bottom-k sketches. In *PODC*. 225–234.
[12] Mohammed Elseidy, Ehab Abdelhamid, Spiros Skiadopoulos, and Panos Kalnis. 2014. Grami: Frequent subgraph and pattern mining in a single large graph. *PVLDB* 7, 7 (2014), 517–528.
[13] Rainer Gemulla, Wolfgang Lehner, and Peter J Haas. 2006. A dip in the reservoir: Maintaining sample synopses of evolving datasets. In *PVLDB*. 595–606.
[14] Rainer Gemulla, Wolfgang Lehner, and Peter J Haas. 2008. Maintaining bounded-size sample synopses of evolving datasets. *VLDBJ* 17, 2 (2008), 173–201.
[15] Bronwyn H Hall, Adam B Jaffe, and Manuel Trajtenberg. 2001. *The NBER patent citation data file: Lessons, insights and methodological tools.* Technical Report. National Bureau of Economic Research.
[16] Guyue Han and Harish Sethu. 2016. Waddling random walk: Fast and accurate sampling of motif statistics in large graphs. *arXiv:1605.09776* (2016).
[17] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data mining: concepts and techniques.* Elsevier.
[18] Jun Huan, Wei Wang, and Jan Prins. 2003. Efficient mining of frequent subgraphs in the presence of isomorphism. In *ICDM*.
[19] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. 2000. An apriori-based algorithm for mining frequent substructures from graph data. In *ECML-PKDD*.
[20] Madhav Jha, C Seshadhri, and Ali Pinar. 2015. A space-efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox. *TKDD* (2015).
[21] Chuntao Jiang, Frans Coenen, and Michele Zito. 2013. A survey of frequent subgraph mining algorithms. *The Knowledge Eng. Review* 28, 1 (2013), 75–105.
[22] Arijit Khan, Xifeng Yan, and Kun-Lung Wu. 2010. Towards proximity pattern mining in large graphs. In *SIGMOD*. 867–878.
[23] Michihiro Kuramochi and George Karypis. 2001. Frequent subgraph discovery. In *ICDM*. 313–320.
[24] Michihiro Kuramochi and George Karypis. 2004. Grew-a scalable frequent subgraph discovery algorithm. In *ICDM*.
[25] Michihiro Kuramochi and George Karypis. 2005. Finding frequent patterns in a large sparse graph. *Data mining and knowledge discovery* 11, 3 (2005), 243–271.
[26] Matthieu Latapy. 2008. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Comp. Sci.* 407, 1-3 (2008), 458–473.
[27] Yongsub Lim and U Kang. 2015. Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams. In *KDD*. 685–694.
[28] Aduri Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. 2013. Counting and sampling triangles from a graph stream. *PVLDB* (2013).
[29] Abhik Ray, Larry Holder, and Sutanay Choudhury. 2014. Frequent Subgraph Discovery in Large Attributed Streaming Graphs. In *BigMine*. 166–181.
[30] Tanay Kumar Saha and Mohammad Al Hasan. 2015. FS3: A sampling based method for top-k frequent subgraph mining. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 8, 4 (2015), 245–261.
[31] Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. 2017. Trièst: Counting local and global triangles in fully dynamic streams with fixed memory size. *TKDD* 11, 4 (2017), 43.
[32] Daniel Ting. 2016. Towards optimal cardinality estimation of unions and intersections with sketches. In *KDD '16*. 1195–1204.
[33] Charalampos E Tsourakakis, Mihail N Kolountzakis, and Gary L Miller. 2011. Triangle Sparsifiers. *J. Graph Algorithms Appl.* 15, 6 (2011), 703–726.
[34] Jeffrey Scott Vitter. 1984. Faster methods for random sampling. *CACM* 27, 7 (1984), 703–718.
[35] Jeffrey S Vitter. 1985. Random sampling with a reservoir. *TOMS* 11, 1 (1985), 37–57.
[36] Bianca Wackersreuther, Peter Wackersreuther, Annahita Oswald, Christian Böhm, and Karsten M Borgwardt. 2010. Frequent subgraph discovery in dynamic networks. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*. ACM, 155–162.
[37] Pinghui Wang, John Lui, Bruno Ribeiro, Don Towsley, Junzhou Zhao, and Xiaohong Guan. 2014. Efficiently estimating motif statistics of large networks. *TKDD* 9, 2 (2014), 8.
[38] Pinghui Wang, John CS Lui, Don Towsley, and Junzhou Zhao. 2016. Minfer: A method of inferring motif statistics from sampled edges. In *ICDE*.
[39] Xifeng Yan and Jiawei Han. 2002. gSpan: Graph-Based Substructure Pattern Mining. In *ICDM*. 721–724.