Approximately Counting Subgraphs in Data Streams

Hendrik Fichtenberger
hendrik.fichtenberger@univie.ac.at
University of Vienna
Vienna, Austria

Pan Peng
ppeng@ustc.edu.cn
School of Computer Science and Technology
University of Science and Technology of China
Hefei, China

ABSTRACT

Estimating the number of subgraphs in data streams is a fundamental problem that has received great attention in the past decade. In this paper, we give improved streaming algorithms for approximately counting the number of occurrences of an arbitrary subgraph H, denoted $\#H$, when the input graph G is represented as a stream of m edges. To obtain our algorithms, we provide a generic transformation that converts constant-round sublinear-time graph algorithms in the query access model to constant-pass sublinear-space graph streaming algorithms. Using this transformation, we obtain the following results.

- We give a 3-pass turnstile streaming algorithm for $(1 \pm \epsilon)$-approximating $\#H$ in $\tilde{O}(\frac{m^{\rho(H)}}{\epsilon^2 \cdot \#H})$ space, where $\rho(H)$ is the fractional edge-cover of H. This improves upon and generalizes a result of McGregor et al. [PODS 2016], who gave a 3-pass insertion-only streaming algorithm for $(1 \pm \epsilon)$-approximating the number $\#T$ of triangles in $\tilde{O}(\frac{m^{3/2}}{\epsilon^2 \cdot \#T})$ space if the algorithm is given additional oracle access to the degrees.
- We provide a constant-pass streaming algorithm for $(1 \pm \epsilon)$-approximating $\#K_r$ in $\tilde{O}(\frac{m\lambda^{r-2}}{\epsilon^2 \cdot \#K_r})$ space for any $r \geq 3$, in a graph G with degeneracy λ, where K_r is a clique on r vertices. This resolves a conjecture by Bera and Seshadhri [PODS 2020].

More generally, our reduction relates the adaptivity of a query algorithm to the pass complexity of a corresponding streaming algorithm, and it is applicable to all algorithms in standard sublinear-time graph query models, e.g., the (augmented) general model.

CCS CONCEPTS

• **Mathematics of computing → Graph algorithms**; • **Theory of computation → Sketching and sampling**.

KEYWORDS

Data streams, Graph sampling, Triangle counting, Subgraph counting

PODS '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9260-0/22/06.
https://doi.org/10.1145/3517804.3524145

ACM Reference Format:
Hendrik Fichtenberger and Pan Peng. 2022. Approximately Counting Subgraphs in Data Streams. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3517804.3524145

1 INTRODUCTION

Estimating the number of occurrences of a small target graph (e.g., a triangle or a clique) in a large graph is a fundamental problem that has received great attention in many domains, including database theory, network science, data mining and theoretical computer science. For example, in database theory, it is closely related to the subgraph enumeration problem and the join-size estimation problem (see, e.g., [2]). In network science, it has applications in estimating the transitivity coefficient and clustering coefficient of a social network (e.g., [29]), and motif detection in biological networks (e.g., [16]).

In this paper, we study this problem in the streaming setting. That is, we are given an n-vertex graph G with m edges that is represented as a stream of edge updates, and a (small) target graph H (e.g., a triangle or a clique). Our goal is to estimate the number of occurrences of H in G by using as small space as possible in a few number of passes over the stream. Throughout the paper, we focus on the *arbitrary-order model*, i.e., the order of the elements in the stream is arbitrary and may be adversarial. The baseline of graph streaming algorithms is the *insertion-only* setting (also known as *cash-register* setting), where the edges of G are given one by one as a stream. When explicitly stated, we also consider the *turnstile* setting, where the stream consists of insertions and deletions (similar to the model of dynamic algorithms). In the latter, the graph G results from applying these insertions and deletions to an initially empty graph on n vertices in the order as they are read from the stream. Each model is relevant for different types of applications: Multi-pass insertion-only algorithms allow to process very large graphs that do not fit into memory as entries in adjacency lists can be seen as insertions. Multi-pass turnstile algorithms can be applied even if a stream of insertions and deletions cannot be consolidated into an insertion-only stream of the final graph, e.g., because the stream is split into multiple substreams that cannot be joined for privacy reasons.

As we will discuss below, the special cases of H being a triangle, a cycle or a clique have been widely studied. However, to the best of our knowledge, the only previous streaming algorithms for $(1 \pm \epsilon)$-approximating the number of copies of an *arbitrary* subgraph H are the following:

(1) Kane et al. [22] gave a 1-pass turnstile algorithm that (i) uses $\tilde{O}(\frac{(m \cdot \Delta(G))^{|E(H)|}}{\epsilon^2 \cdot (\#H)^2})$ space for any subgraph H, where $\#H$

is the number of occurrences of H in G and $\Delta(G)$ is the maximum degree of G, or (ii) uses $\tilde{O}(\frac{m^{|E(H)|}}{\epsilon^2 \cdot (\#H)^2})$ space if the minimum degree of H is at least 2.

(2) Bera and Chakrabarti [5] gave a 2-pass algorithm with space $\tilde{O}(m^{\beta(H)}/(\epsilon^2 \#H))$, where $\beta(H)$ is the *integral edge cover* number[1] of H.

(3) Assadi et al. [2] gave a C-pass streaming algorithm with space complexity $\tilde{O}(\frac{m^{\rho(H)}}{\epsilon^2 \cdot \#H})$, where $\rho(H)$ is the *fractional edge cover* of H (see Definition 3) and C is some constant depending on H. We remark that C is not explicitly specified in [2], but as far as we can see, a straightforward transformation of their sublinear-time algorithm to the insertion-only streaming setting gives that $C \geq \rho(H) \in \Omega(|V(H)|)$.[2] Furthermore, it is known that $\rho(H) \leq \beta(H) \leq |E(H)|$ and that $\#H \leq m^{\rho(H)}$ [3], and thus the space complexity of the algorithm in [2] is always no worse than the ones in [5, 22].

Furthermore, it is known that 1-pass turnstile algorithms need at least $\tilde{\Omega}(m/(\#H)^{1/\tau})$ space, where τ is the fractional vertex-cover of H (analogous to Definition 3), even for bounded-degree graphs [19].

There exist many streaming algorithms for the special cases of H being an r-clique K_r, or a length-r cycle C_r, for any constant $r \geq 3$. The performance guarantees of these algorithms are parameterized by various parameters of G, e.g., $\#H$, the maximum degree, the maximum number of triangles which share a single vertex (or an edge), etc. In the following, we will mainly discuss the state-of-the-art results that are most relevant to our setting, i.e., those that provide $(1 \pm \epsilon)$-approximations with space complexity parameterized just by $\#H$ (and n, m, ϵ) in the arbitrary-order model.

Triangles. ($\rho(C_3) = 3/2$). Approximating the number of occurrences of a triangle T has been studied in a long line of work [1, 4, 5, 7–9, 12, 17, 18, 21, 23, 24, 26, 28, 29, 31]. In one pass, Manjunath et al. [24] gave one algorithm achieving $\tilde{O}(\frac{m^3}{\epsilon^2 \cdot (\#T)^2})$ space (in the turnstile model), which is nearly optimal as any 1-pass algorithm for this problem requires $\Omega(\frac{m^3}{(\#T)^2})$ space [8]. In two passes, McGregor, Vorotnikova and Vu [26] gave one algorithm using $\tilde{O}(\frac{m}{\epsilon^2 \cdot \sqrt{\#T}})$ space (see also [13]). This is in contrast with a lower bound $\Omega(\min\{\frac{m}{\sqrt{\#T}}, \frac{m^{3/2}}{\#T}\})$ for any multi-pass algorithm by Bera and Chakrabarti [5]. In three passes, McGregor, Vorotnikova and Vu [26] gave one algorithm using $\tilde{O}(\frac{m^{3/2}}{\epsilon^2 \cdot \#T})$ space, while *their algorithm is assumed to have oracle access to vertex degrees.* In four passes, Bera and Chakrabarti [5] gave an algorithm using $\tilde{O}(\frac{m^{3/2}}{\epsilon^2 \cdot \#T})$ space.

Cycles. ($\rho(C_r) = r/2$). The case of counting a length-r cycle C_r (for some constant $r \geq 4$) has been studied in [5, 24, 25]. In one pass, the turnstile algorithm in [24] achieves $\tilde{O}(\frac{m^r}{\epsilon^2 \cdot (\#C_r)^2})$ space, which is in contrast to a 1-pass space lower bound $\Omega(m^{r/2}/(\#C_r)^2)$ for even r and $\Omega(m^r/(\#C_r)^2)$ for odd r [5]. There exists an algorithm with space complexity $\tilde{O}(\frac{m^{r/2}}{\epsilon^2 \cdot \#C_r})$ using two passes for even r and four passes for odd r [5]. In contrast, any multi-pass streaming algorithm requires $\Omega(m^{r/2}/\#C_r)$ space for even r and $\Omega(\min\{\frac{m^{r/2}}{\#C_r}, \frac{m}{(\#C_r)^{1/(r-1)}}\})$ space for odd r [5]. In three passes, there exists an algorithm for C_4 using $\tilde{O}(\frac{m}{\epsilon^2 \cdot (\#C_4)^{1/4}})$ space [25].

Cliques. ($\rho(K_r) = r/2$). The case of counting an r-clique K_r (for some constant $r \geq 4$) has been studied in [5, 29]. In one pass, it is necessary to use $\Omega(\frac{m^r}{(\#K_r)^2})$ space. There exists one algorithm with $\tilde{O}(\frac{m^{r/2}}{\#K_r})$ space that uses two passes for even r and four passes for odd r. In contrast, any multi-pass streaming algorithm requires $\Omega(\min\{\frac{m^{r/2}}{\#K_r}, \frac{m}{(\#K_r)^{1/(r-1)}}\})$ space [5].

Finally, we mention that recently Bera and Seshadhri [6] motivated the study of streaming algorithms for subgraph counting in low *degeneracy* graphs (see Definition 5), which is a natural class of graphs arising in practice. In addition, the class of constant degeneracy graphs includes all planar graphs, minor-closed families of graphs and preferential attachment graphs. For a graph with degeneracy at most λ, they gave a 6-pass algorithm with space complexity $\frac{m\lambda}{\#T} \cdot \text{poly}(\log n, \epsilon^{-1})$ for $(1 \pm \epsilon)$-approximating the number of triangles in G, which breaks the worst-case lower bound for general graphs. It was conjectured that there exists a constant pass streaming algorithm for any clique with space complexity $\tilde{O}(m\lambda^{r-2}/\#K_r)$ in a graph with degeneracy at most λ [6].

1.1 Our results

Let n and m be the number of vertices and edges in the input graph G, respectively. Let $\#H$ be the number of subgraphs H in G. Sometimes, we use K_r and T to denote the subgraphs r-clique (i.e., a clique on r vertices) and triangle, respectively. Though we do not know $\#H$ in advance, we adopt the common convention from literature to parameterize our algorithms in terms of $\#H$. Since $\#H$ is unknown, $\#H$ can be replaced by a lower bound L on $\#H$ to obtain corresponding guarantees for our algorithms. Alternatively, one can phrase the problem as distinguishing if the number of subgraphs H is at most L or at least $(1+\epsilon)L$ for an input parameter L.

We first present the following algorithm.

Theorem 1. *Let $\epsilon > 0$ and let H be an arbitrary subgraph of constant size. There exists a 3-pass turnstile streaming algorithm for computing a $(1+\epsilon)$-approximation of the number of copies of H in the input graph G with high probability[3] that has space complexity $\tilde{O}(m^{\rho(H)}/(\epsilon^2 \#H))$.*

Note that the space complexity of our turnstile algorithm matches the insertion-only algorithm in [2] for approximating $\#H$, while our algorithm uses only three instead of $\Omega(|V(H)|)$ passes, even in the turnstile setting (see discussion above). Furthermore, for the special case of triangles, the space complexity of our turnstile algorithm also matches the state-of-the-art of insertion-only algorithms, which is $\tilde{O}(m^{3/2}/\#T)$ [5, 26], while these algorithms either use three passes together with the assumption that the algorithm is given oracle access to the degrees [26], or use four passes [5].

[1] The integral edge cover of H, denoted $\beta(H)$, is the cardinality of its smallest edge cover, where an edge cover of H is a set of edges that covers all its vertices. It is known that for an r-clique K_r, $\beta(K_r) = \lceil \frac{r}{2} \rceil$, and for a length-$r$ cycle C_r, $\beta(C_r) = \lceil \frac{r}{2} \rceil$.

[2] In [2], a so-called sampler tree of depth at least $\rho(H)$ is built top-down by querying the graph. To obtain a sufficiently small bound on the space, level i of the tree must be fully constructed before level $i+1$ can be constructed. It seems necessary that the algorithm makes at least one full pass to construct a single level.

[3] In this paper, 'with high probability' refers to 'with probability at least $1 - n^{-C}$, for some constant $C > 0$'.

Our second result is a constant-pass algorithm for approximating $\#K_r$ in low degeneracy graphs, which resolves a conjecture by Bera and Seshadhri [6]. This algorithm also generalizes the algorithm in [6] that only considers $r = 3$ (i.e., the triangle case).

Theorem 2. *For any $\epsilon > 0, r \geq 3$, there exists a $5r$-pass insertion-only streaming algorithm for computing an $(1+\epsilon)$-approximation to $\#K_r$ in graphs with degeneracy λ that has space complexity*

$$\frac{m\lambda^{r-2}}{\#K_r} \cdot \mathrm{poly}(\log n, \epsilon^{-1}, r^r)$$

and succeeds with high probability.

Both of our two main results are obtained by a generic transformation between streaming algorithms and sublinear-time algorithms in the query access model (see Definition 6). More precisely, we relate the *adaptivity* of any query algorithm to the *pass complexity* of a corresponding streaming algorithm that is obtained by the transformation.

1.2 Our techniques

The adaptivity of sublinear query algorithms, is usually classified into *non-adaptive* and *adaptive* algorithms. Non-adaptive algorithms must specify all queries on their input in advance, and adaptive algorithms may ask arbitrary queries during their computation (in particular, a query might depend on the previous query answers). Inspired by a notion by Cannone and Gur [10], we define the *round-adaptivity* of a sublinear-time graph query algorithm, which formalizes "the number of levels of dependencies" needed in an adaptive algorithm. Intuitively, each level corresponds to a set of queries that only depends on the queries in the previous levels and not on the queries on the same level (or later levels). Then we argue that exploiting this round-adaptivity leads to a fruitful connection between query algorithms and streaming algorithms. In particular, we show that if an algorithm is allowed to ask a batch of non-adaptive queries not just once, but for $k > 1$ rounds, this translates very naturally into a k-pass streaming algorithm. To illustrate our transformation, we show that known sublinear-time algorithms for sampling and counting subgraphs [14, 15] lead to novel streaming algorithms that advance the state of the art.

1.3 Other related work

There has been a line of works for approximately counting subgraphs in other graph stream models, including the random order model (in which the stream consists of a random permutation of the edges) [25, 26], as well as in the adjacency list model (in which each edge appears twice and the edges in the stream are grouped by their endpoints) [20, 26].

2 PRELIMINARIES

Graphs. We consider undirected graphs. For the input graph $G = (V, E)$ of an algorithm, we use $n := |V|$ and $m := |E|$, and we denote the number of subgraphs H in G by $\#H$. The degree of a vertex v is denoted $\mathrm{dg}(v)$. Our algorithms' space complexities are parameterized by the following concepts.

Definition 3 (Fractional Edge-Cover Number)**.** *A fractional edge-cover of $H = (V_H, E_H)$ is a mapping $\psi : E_H \to [0,1]$ such that for each vertex $v \in V_H$, $\sum_{e \in E_H, v \in e} \psi(e) \geq 1$. The fractional edge-cover number $\rho(H)$ of H is the minimum value of $\sum_{e \in E_H} \psi(e)$ among all fractional edge-covers ψ.*

Let C_k denote the cycle of length k. Let S_k denote a star with k petals, i.e., $S_k = (\{u, v_1, \ldots, v_k\}, \cup_{i \in [k]} \{u, v_k\})$. Let K_k denote a clique on k vertices. It is known that $\rho(C_{2k+1}) = k + 1/2$, $\rho(S_k) = k$ and $\rho(K_k) = k/2$. The following result is known [2, 27], see also [30, Theorem 30.10].

Lemma 4. *For any subgraph H, there exist $\alpha, \beta \geq 0$ so that H can be decomposed into a collection of vertex-disjoint odd cycles $\overline{C_1}, \ldots, \overline{C_\alpha}$ and star graphs $\overline{S_1}, \ldots, \overline{S_\beta}$ such that*

$$\rho(H) = \sum_{i=1}^{\alpha} \rho(\overline{C_i}) + \sum_{j=1}^{\beta} \rho(\overline{S_j}).$$

Definition 5 (degeneracy)**.** *The degeneracy of a graph is the smallest $\kappa \geq 0$ so that every subgraph has maximum vertex degree κ.*

Graph query algorithms. In the augmented general graph model, an algorithm gets the input size n and query access to an input graph $G = (V, E)$, where $V = [n]$, and it may ask for random edges, query degrees and neighbors of vertices, and check the existence of edges in E. Formally, it is defined as follows.

Definition 6. *The* augmented general graph model *is defined for the set of all graphs. For a graph $G = (V, E)$, where $V = [n]$, it allows four types of queries: (f_1) return a uniformly random edge $e \in E$; (f_2) given $v \in V$, return the degree of v; (f_3) given $v \in V$ and $i \in [\mathrm{dg}(v)]$, return the i^{th} neighbor of v; (f_4) given $u, v \in V$, return whether $(u, v) \in E$.*

The query complexity *of a graph query algorithm is the total number of queries it asks on its input; and its* space complexity *is the maximal amount of space it uses during its execution (including space to store query answers, but excluding the space used to store the whole input).*

The *general graph model* is the augmented general graph model without random edge queries, i.e., f_1.

Streaming algorithms. For our transformation, we use the following result on ℓ_0-samplers.

Lemma 7 ([11])**.** *Let $c > 0$. There exists an ℓ_0-sampler for turnstile streams on $\mathbb{Z}^n$ that requires $O(\log^4 n)$ bits of space, succeeds with probability $1 - 1/n^c$ and, if successful, outputs a non-zero entry i with probability $1/N \pm 1/n^c$, where N is the number of non-zero entries.*

In pseudo code of streaming algorithms, we number the passes with respect to the current procedure, i.e., the first pass on the input in a procedure is numbered 1. Sometimes, we use parallel computation (in particular, "**parallel for**" loops) to enable different computations to utilize the same pass on the input. Computation that is performed during a pass on the input is placed inside a "**pass**"-block that specifies the corresponding pass(es). For the sake of clarity, we also specify the knowledge / variables ("input") that are available at the beginning of the pass. To make it more explicit that a streaming algorithm has queried the degrees of some vertex set V', we use $d[V']$ to denote a dictionary that maps every $v \in V'$ to $d[V']_v = \mathrm{dg}(v)$. We use $\tilde{O}(\cdot)$ to omit polylogarithmic factors and dependencies on the size of H.

3 TRANSFORMATION

In this section, we present and prove a transformation that allows us to obtain constant-pass streaming algorithms from sublinear query algorithms. The number of passes depends on the level of adaptivity of the query algorithm. In particular, we consider the number of batches (*rounds*) that queries can be grouped into so that a query in batch i depends only on the algorithm's random coins and the answer's to queries in batches $1, \dots, i-1$.

Definition 8 (round-adaptive graph algorithm, cf. [10])**.** *Let $\mathcal{G}$ be a set of graphs, let $\mathcal{F} = (f_1, \dots)$ be a finite family of functions, where $f_i : \mathcal{G} \times X_i \to Y_i$ and X_i, Y_i are sets, and let $k : \mathcal{G} \to \mathbb{N}$. A* graph query *algorithm for a graph problem on $\mathcal{G}$ and* query types *$\mathcal{F}$ can access its input $G \in \mathcal{G}$ only by asking queries $f \in \mathcal{F}$ on G. It is said to be* k-round adaptive *if the following holds:*

The algorithm proceeds in $k(G)$ rounds. In round $\ell > 0$, it produces a sequence of queries $Q_\ell := (q^{\ell,i} = (t^{\ell,i}, x^{\ell,i}))_{i \in [|Q_\ell|]}$, where $t^{\ell,i} \in [|\mathcal{F}|]$ is a query type and $x^{\ell,i} \in X_{t^{\ell,i}}$ is the query's arguments. The sequence of queries Q_ℓ is based on the algorithm's own internal randomness and the answers $\mathcal{F}(Q_1), \dots, \mathcal{F}(Q_{\ell-1})$ to the previous sequences of queries $Q_1, \dots, Q_{\ell-1}$. In return to Q_ℓ, the algorithms receives a sequence of query answers $\mathcal{F}(Q_\ell) = (f_{t^{\ell,i}}(q^{\ell,i}))_{i \in [|Q_\ell|]}$.

Example. Let us consider a very simple subroutine in the augmented general graph model where the goal is to find a triangle in a graph G. This subroutine simply does the following:

(1) Sample one edge $e = (u, v)$ uniformly at random,
(2) Query the degrees of u, v and find the one, say u, whose degree is no larger than the other,
(3) Sample a random neighbor w of u, and
(4) Query if there exists an edge between v and w.

The above subroutine is a 4-round adaptive graph query algorithm: In round 1, the query set $Q_1 = (q^{1,1} = (1, \cdot))$ is simply one random edge and the query answer $\mathcal{F}(Q_1)$ is $(e = (u, v))$; in round 2, the query set $Q_2 = (q^{2,1} = (2, u), q^{2,2} = (3, v))$ are the degree queries of u, v and the query answer is $\mathcal{F}(Q_2) = (\deg(u), \deg(v))$; in round 3, the query set $Q_3 = (q^{3,1} = (3, (u, x))$, where x is drawn uniformly random from $[\deg(u)]$, is for a random neighbor of u, and the query answer is $\mathcal{F}(Q_3) = w$; in round 4, the query set $Q_4 = (q^{4,1} = (4, (v, w)))$ is if there exists an edge between v, w, and the query answer is $\mathcal{F}(Q_4) =$ (**Yes**) if edge (v, w) exists and $\mathcal{F}(Q_4) =$ (**No**) otherwise.

We state and prove the transformation from sublinear-time algorithms in the augmented general graph model that yields insertion-only streaming algorithms. Since the augmented general graph model subsumes the standard models for dense graphs, bounded-degree graphs and general graphs, one can directly obtain a streaming algorithm from essentially any sublinear graph query algorithm with small round-adaptivity.

Theorem 9. *Let $\mathcal{A}_Q$ be a k-round adaptive graph query algorithm for the augmented general graph model with query complexity $q = q(n)$ and space complexity $s = s(n)$. Then, there exists a k-pass algorithm $\mathcal{A}_S$ in the arbitrary-order insertion-only graph streaming model with space complexity $O(q \log n + s)$ bits so that $\mathcal{A}_S$ and $\mathcal{A}_Q$ have the same output distribution.*

Proof. Let $Q_1, \dots, Q_k$ be the k query sets that are asked by $\mathcal{A}_Q$. We define $\mathcal{A}_S$ to be the algorithm that sequentially computes $\mathcal{F}(Q_i)$, given $Q_1, \dots, Q_{i-1}$ and $\mathcal{F}(Q_1), \dots, \mathcal{F}(Q_{i-1})$, for $i \le k$. We prove that $\mathcal{A}_S$ can compute the answers to $\mathcal{F}(Q_i)$ in pass i. Let $i \in [k]$, let $j \in [|Q_i|]$ and consider query $q^{i,j} = (t^{i,j}, x^{i,j}) \in Q_i$. We distinguish the query type $f_{t^{i,j}}$ and explain how the algorithm emulates the query oracle:

- f_1 **(uniform edge):** A uniformly random edge can be obtained from the stream via reservoir sampling using $O(\log n)$ bits of space.
- $f_2(v)$ **(degree):** A counter of the degree of v can be maintained while reading edges from the stream, using $O(\log n)$ bits of space.
- $f_3(v, i)$ **(neighbor):** The algorithm initializes a counter for v that counts the number of edges read from the stream that are incidient to v. Once the counter reaches the value i, the algorithm returns u from the edge (u, v) it just read. This requires $O(\log n)$ bits of space.
- $f_4(u, v)$ **(adjacency):** The algorithm maintains a boolean variable that indicates whether the edge (u, v) was read from the stream, which requires $O(\log n)$ bits of space.

The total space required to store all query answers is thus $O(q \log n)$. To emulate the original algorithm, one needs $O(s)$ space. □

To adapt sublinear graph query algorithms to *turnstile* streams, we propose the following relaxed version of the augmented general graph model.

Definition 10. *Let $c > 0$. The* relaxed augmented general graph model *is defined for the set of all graphs. For a graph $G = (V, E)$, where $V = [n]$, it allows four types of queries:*

(f_1) for every edge $e \in E$, returns e with probability $1/m \pm 1/n^c$, or fails with probability at most $1/n^c$;
(f_2) given $v \in V$, returns the degree of v;
(f_3) given $v \in V$, for every $u \in \Gamma(v)$, returns u with probability $1/\deg(u) \pm 1/n^c$, or fails with probability at most $1/n^c$;
(f_4) given $u, v \in V$, returns whether $(u, v) \in E$.

The probabilities are taken over the random coins of the respective query.

This model differs in two aspects from the augmented general graph model: First, random edges that are queried via f_1 are not exacty uniformly random. Second, instead of asking for the i^{th} neighbor of a vertex v via f_3, one can only obtain an approximately uniformly random neighbor of v. Intuitively, these relaxed guarantees weaken the solution quality and the complexity of most sublinear algorithms only slightly. In particular, we prove this for subgraph counting. As a benefit of this model, we show that k-round adaptive graph query algorithms translate into k-pass *turnstile* streaming algorithms.

Theorem 11. *Let $\mathcal{A}_Q$ be a k-round adaptive graph query algorithm for the relaxed augmented general graph model with query complexity $q = q(n)$ and space complexity $s = s(n)$. Then, there exists a k-pass algorithm $\mathcal{A}_S$ in the arbitrary-order turnstile graph streaming model with space complexity $O(q \log^4 n + s)$ bits so that $\mathcal{A}_S$ and $\mathcal{A}_Q$ have the same output distribution.*

Proof. Let $Q_1, \ldots, Q_k$ be the k query sets that are asked by $\mathcal{A}_Q$. We define $\mathcal{A}_S$ to be the algorithm that sequentially computes $\mathcal{F}(Q_i)$, given $Q_1, \ldots, Q_{i-1}$ and $\mathcal{F}(Q_1), \ldots, \mathcal{F}(Q_{i-1})$, for $i \le k$. We prove that $\mathcal{A}_S$ can compute the answers to $\mathcal{F}(Q_i)$ in pass i. Let $i \in [k]$, let $j \in [|Q_i|]$ and consider query $q^{i,j} = (t^{i,j}, x^{i,j}) \in Q_i$. We distinguish the query type $f_{t^{i,j}}$ and explain how the algorithm emulates the query oracle:

- f_1 **(uniform edge):** The algorithm maintains an ℓ_0-sampler of the adjacency matrix of the graph. By Lemma 7, this requires $O((\log n^2)^4) = O(\log^4 n)$ bits of space.
- $f_2(v)$ **(degree):** A counter of the degree of v can be maintained while reading insertions and deletions of edges that are incident to v from the stream, using $O(\log n)$ bits of space.
- $f_3(v)$ **(random neighbor):** The algorithm maintains an ℓ_0-sampler of the adjacency list of v. This requires $O(\log^4 n)$ bits of space by Lemma 7.
- $f_4(u, v)$ **(adjacency):** The algorithm maintains a boolean variable that indicates whether the last update of (u, v) that was read from the stream was an insertion or a deletion, which requires $O(\log n)$ bits of space.

The total space required to store all query answers is thus $O(q \log^4 n)$. To emulate the original algorithm, one needs $O(s)$ space. □

4 SUBGRAPH COUNTING AND SAMPLING

In this section, we analyze the round-adaptivity of the sublinear algorithm for sampling uniformly random copies of a given subgraph H by Fichtenberger, Gao and Peng [15], which can also be easily adapted to obtain a subgraph counting algorithm, in the augmented general graph model. We show that this algorithm is 3-round adaptive. Therefore, it yields a 3-pass streaming algorithm for sampling and counting subgraphs via Theorem 9.

4.1 A sublinear-time algorithm for counting arbitrary subgraphs

We make use of a subroutine from [15] for sampling a copy of subgraph H in G, and we refer to this subroutine as the *FGP algorithm* in the following. To describe the FGP algorithm, we state the relevant definitions.

Definition 12 (vertex order)**.** *Let $G = (V, E)$ be a graph, let $u, v \in V$. We define $u \prec_G v$ if and only if $\deg_G(u) < \deg_G(v)$, or $\deg_G(u) = \deg_G(v)$ and $id(u) < id(v)$.*

Definition 13 (canonical cycle)**.** *Let $G = (V, E)$ be a graph and let $E' \subseteq E$. A sequence of vertices $(u_1, \ldots, u_k)$ is a canonical k-cycle in $(E', \prec_G)$ if, for all $i \in [k]$, $(u_i, u_{i+1 \mod k+1}) \in E'$ and, for $2 \le i \le k$, $u_1 \prec u_i$ and $u_k \prec u_2$.*

Definition 14 (canonical star)**.** *Let $G = (V, E)$ be a graph and let $E' \subseteq E$. A sequence of vertices $(u_0 u_1, \ldots, u_k)$ is a canonical k-star in $(E', \prec_G)$ if, for all $i \ge 1$, $(u_0, u_i) \in E'$, and, for $1 \le i < k$, $u_i \prec u_{i+1}$.*

High-level idea of the FGP algorithm. The idea of the FGP algorithm is to first compute a decomposition of the subgraph H into odd cycles and stars according to Lemma 4, and design subroutines to sample each canonical cycle of length $2k + 1$ with probability $1/(2m)^{k+1/2}$, and each canonical k-petal star with probability $1/(2m)^k$. Together with the relation between the fractional edge cover $\rho(H)$ of a subgraph H and its decomposition into odd length cycles and stars, the FGP algorithm then uses these samples to output a subgraph such that for any copy of H, it is output with probability $1/(2m)^{\rho(H)}$.

More precisely, the algorithms tries to sample a copy of H by sampling canonical cycles and stars according to Definitions 13 and 14. For a canonical k-star $(u_0, \ldots, u_k)$, it simply samples k random edges $(v_1, w_1), \ldots, (v_k, w_k)$ and checks if the sampled edges form indeed a k-star subgraph, $v_1 = \ldots = v_k$ and $w_i \prec w_{i+1}$ for all $i \in [k-1]$. For a canonical odd cycle $(u_1, \ldots, u_{2k+1})$, it tries to sample every second edge, i.e., $(u_1, u_2), (u_3, u_4), \ldots, (u_{2k-1}, u_{2k})$. Note that only u_{2k+1} is still unknown to the algorithm. Now, the algorithm proceeds based on the following case distinction: either, all the vertices in the cycle have degree greater than $\sqrt{2m}$. Then, one endpoint of a uniformly random edge is u_{2k+1} with probability $\deg(u)/2m \approx 1/\sqrt{2m}$. Otherwise, it samples the i^{th} neighbor of u_1 (if it exists), where i is drawn uniformly at random from $[\sqrt{2m}]$. Since u_1 has degree less than $\sqrt{2m}$, u_{2k+1} is sampled with probability $1/\sqrt{2m}$. Then, the algorithm checks whether the sampled edges form a cycle of length $2k + 1$, and whether it is canonical.

For the sake of completeness, we provide pseudo code of the FGP algorithm (i.e., Algorithm 9 SampleSubgraph) in Appendix B. Its performance guarantee is given in the following lemma.

Lemma 15 ([15], Lemma 8)**.** *Let H be an arbitrary subgraph of constant size. The FGP algorithm takes an input graph $G = (V, E)$ and the number of edges m in G, uses $O(1)$ queries in expectation and guarantees the following: For a fixed copy of H in G, the probability that H is returned is $1/(2m)^{\rho(H)}$.*

4.2 Insertion-only streaming algorithm

For the sake of presentation, we start with an insertion-only algorithm. Since Theorem 9 transforms round-adaptive query algorithms in the (standard) augmented general graph model to insertion-only streaming algorithms, our only objective in this section is to prove the round-adaptivity of the FPG algorithm.

Lemma 16. *Let H be an arbitrary subgraph of constant size. There exists a 3-pass insertion-only streaming algorithm (i.e., Algorithm 1) that has space complexity $O(\log n)$ and returns a copy of H or nothing. For any fixed copy of H in the input graph, it is returned with probability $1/(2m)^{\rho(H)}$.*

Proof. We prove that the FGP-algorithm (see Algorithm 9 SampleSubgraph in Appendix B) is 3-round adaptive by devising a partition of its queries into 3 rounds.

First, the FGP-algorithm computes a decomposition of H into $\alpha \ge 0$ odd cycles with lengths $c_1, \ldots, c_\alpha$ and $\beta \ge 0$ stars with $s_1, \ldots, s_\beta$ petals according to Lemma 4 without making any queries. In the first round, the FGP-algorithm samples a set of edges that will be used to form potential cycles and stars; in the second round, the algorithm samples a random neighbor of some vertex in each of the potential cycles; in the third round, the algorithm performs vertex pair queries to check if these potential cycles and stars are indeed cycles and stars, respectively. Then we run a postprocessing on the collected subgraph and output a copy H if it is found. By the proof of Lemma 15, any copy of H is output with probability $1/(2m)^{\rho(H)}$.

For the sake of presentation, we state the algorithm as a streaming algorithm in Algorithm 1. The space complexity of this algorithm then follows from Lemma 15 and Theorem 9.

We argue that Algorithm 1 is indeed a 3-pass algorithm. To emulate the queries, we invoke Theorem 9. In the first pass, the algorithm only needs to know the decomposition of H to sample sufficiently many edges, and it computes the number of edges m. Prior to the second pass, it needs to know m to sample i from $\{1, \ldots, \sqrt{2m}\}$. The third pass checks only the existence of edges between the known vertices in $V' = (C'_i)_{i\in[\alpha]} \cup (S_i)_{i\in[\beta]}$ and computes their degrees. Afterwards, the algorithm has obtained the subgraph induced by all vertices and the degrees of these vertices in G.

We note that the information collected by Algorithm 1 is enough to check whether the sampled cycles and stars are canonical and to check whether $G[V']$ spans or induces a copy of H, which are the only checks performed by Line 31 – Line 33. □

Since there are $\#H$ copies of H in G, the probability that Algorithm 1 returns a copy of H is $\#H/(2m)^{\rho(H)}$ by Lemma 16. Then the subgraph counting algorithm can be obtained by viewing the above subgraph sampler as a biased coin. That is, let $p = \frac{\#H}{(2m)^{\rho(H)}}$ be the probability of a coin getting a Heads on a flip (which corresponds to a copy of H returned by Algorithm 1). By a standard Chernoff bound argument, one can obtain a multiplicative approximation of p by flipping it sufficiently many times and counting how often it turns up heads. Formally, we have the following theorem.

Theorem 17. *Let $\epsilon > 0$ and let H be an arbitrary subgraph of constant size. There exists a 3-pass insertion-only streaming algorithm for computing a $(1+\epsilon)$-approximation of the number of copies of H in the input graph G with high probability that has space complexity $\tilde{O}(m^{\rho(H)}/(\epsilon^2 L))$, where L is a lower bound on $\#H$.*

Proof. We run $k = 30(2m)^{\rho(H)} \ln n \,/\, (\epsilon^2 L)$ copies of Algorithm 1 in parallel. The probability that a single instance returns a subgraph is $\#H/(2m)^{\rho(H)}$ by Lemma 16. Let x denote the fraction of invocations that returned a subgraph. Since all invocation are independent of each other, using Chernoff bound it follows that $|\#H - (2m)^{\rho(H)}x| \le \epsilon \cdot \#H$ with probability at least $1 - \frac{1}{n^{\Omega(1)}}$. Since each instance of the algorithm requires $O(\log n)$ space by Lemma 16, the total space bound is $O(k \log n) = O(m^{\rho(H)} \log^2(n) \,/\, (\epsilon^2 L))$. □

Algorithm 1 Sampling a subgraph candidate from a stream

1: **procedure** StreamSubg($C = (c_1, \ldots, c_\alpha), S = (s_1, \ldots, s_\beta)$)
2: **pass 1**; input: C, S
3: **for all** $c_i \in C$ **do**
4: sample $\lceil c_i/2 \rceil + 1$ edges $C_i = \{u_{i,0}, v_{i,0}, \ldots, u_{i,\lceil c_i/2\rceil}, v_{i,\lceil c_i/2\rceil}\}$ ▷ f_1
5: **for all** $s_i \in S$ **do**
6: sample k edges $S_i = \{x_{i,1}, y_{i,1}, \ldots, x_{i,k}, y_{i,k}\}$ ▷ f_1
7: count the number of edges m
8: **pass 2**; input: $C, S, (C_i)_{i\in[\alpha]}, (S_i)_{i\in[\beta]}, m$
9: **for all** $c_i \in C$ **do**
10: sample $j \in [\sqrt{2m}]$ uniformly at random
11: $w_i \leftarrow j^{\text{th}}$ neighbor of $u_{i,1}$ ▷ f_3
12: $C'_i = C_i \cup \{w_i\}$
13: **pass 3**; input: $C, S, (C'_i)_{i\in[\alpha]}, (S_i)_{i\in[\beta]}, m$
14: **for all** $z, z' \in (C'_i)_{i\in[\alpha]} \cup (S_i)_{i\in[\beta]}$ **do**
15: check whether $(z, z') \in E$ ▷ f_4
16: count degree of z ▷ f_2
17: Let $V' := (C'_i)_{i\in[\alpha]} \cup (S_i)_{i\in[\beta]}$, $E' := E \cap (V' \times V')$, $V'' := \emptyset$
18:
19: ▷ Postprocessing
20: **for all** $c_i \in C'_i$ **do**
21: **if** $d[V']_{u_{i,1}} \le \sqrt{2m}$ **then**
22: check if $(w_i, u_{i,1}, v_{i,1}, \ldots, u_{i,\lceil c_i/2\rceil}, v_{i,\lceil c_i/2\rceil})$ is a canonical c_i-cycle in $(E', \prec_G)$
23: $V'' = V'' \cup \{w_i, u_{i,1}, \ldots, u_{i,c_i}, v_{i,1}, \ldots, v_{i,\alpha}\}$
24: **else**
25: sample $t \in [0, 1]$ uniformly at random
26: check if $t \le \sqrt{2m}/\deg(u_{i,0})$
27: check if $(u_{i,0}, u_{i,1}, v_{i,1}, \ldots, u_{i,\lceil c_i/2\rceil}, v_{i,\lceil c_i/2\rceil})$ is a canonical c_i-cycle in $(E', \prec_G)$
28: $V'' = V'' \cup \{u_{i,0}, u_{i,1}, \ldots, u_{i,c_i}, v_{i,1}, \ldots, v_{i,\alpha}\}$
29: **for all** $s_i \in S_i$ **do**
30: check if $x_{i,1} = \ldots = x_{i,s_i}$ and $(x_{i,1}, y_{i,1}, \ldots, y_{i,s_i})$ is a canonical s_i-star in $(E'', \prec_G)$
31: $V'' = V'' \cup \{x_{i,1}, y_{i,1}, \ldots, y_{i,s_i}\}$
32: check if $G[V'']$ contains a copy H_G of H as subgraph
33: **return** H_G if all checks were successful

4.3 Turnstile streaming algorithm

In this section, we adapt the analysis of the insertion-only algorithm from Section 4.2 to the *relaxed* augmented general graph model. In particular, we show that essentially the same algorithm yields *approximately* uniformly randomly sampled subgraphs, which in turn is still sufficient to approximately count subgraphs.

Lemma 18. *Let $\epsilon > 0$, and let H be an arbitrary subgraph of constant size. There exists an algorithm for the relaxed augmented general graph model that has space complexity $O(\log^4 n)$ and returns a copy of H or nothing. For any fixed copy of H in the input graph, it is returned with probability $(1 \pm \epsilon)/(2m)^{\rho(H)}$.*

Proof. Let $C = (c_1, \ldots, c_\alpha), S = (s_1, \ldots, s_\beta)$ be a decomposition of H into odd-length cycles and stars that satisfies the guarantees of Lemma 4 and that is passed to Algorithm 1. We slightly modify Algorithm 1 as follows. In Algorithm 1, we replace Lines 10 and 11 by setting w_i to the answer of a query $f_3(u_{i,1})$, i.e., an approximately uniformly random neighbor of $u_{i,1}$. After Line 21, we sample a uniformly random number t from $[\sqrt{2m}]$ and check whether it is at most $\deg(u_{i,1})$. Since no additional queries are asked, the space complexity follows from Lemma 15 and Theorem 11. Pseudo code and the proof of correctness are provided in Appendix A. □

We can obtain the subgraph counting algorithm in the turnstile streaming model similarly as we prove Theorem 17 by considering the sampler as a biased coin and estimating its heads probability. Now we are ready to prove Theorem 1.

Proof of Theorem 1. Consider the algorithm from Lemma 18 (i.e., Algorithm 5) with $\varepsilon = \frac{\epsilon}{3}$. Let p be the probability that some copy of subgraph H is returned. By Lemma 18, $p = \#H(1 \pm \frac{\epsilon}{3})/(2m)^{\rho(H)}$. We can think of the above algorithm as tossing a coin with bias p. By the Chernoff bound, with high probability, the bias p can be estimated up to a multiplicative factor $(1 \pm \frac{\epsilon}{3})$ in $O(\log n/\varepsilon^2 p) = \tilde{O}(m^{\rho(H)}/(\epsilon^2 \#H))$ tosses, which can be implemented by running in parallel the same number of copies of Algorithm 5. Given the estimate p, $\#H$ can be approximated within a multiplicative factor $(1 \pm \epsilon)$. Since each instance of the algorithm requires $O(\log^4 n)$ space by Lemma 18, the total space bound is $\tilde{O}(m^{\rho(H)}/(\epsilon^2 \#H)) \cdot O(\log^4 n) = \tilde{O}(m^{\rho(H)}/(\epsilon^2 \#H))$. This finishes the proof of the theorem. □

5 LOW-DEGENERACY CLIQUE COUNTING

5.1 A sublinear-time algorithm for counting cliques in a graph with low degeneracy

Now we describe the sublinear-time algorithm in the general graph model (i.e., the augmented general graph model without edge sampling queries) for approximating $\#K_r$ of a graph with degeneracy at most λ, for any $r \geq 3$ and $\lambda > 0$. The algorithm is given by Eden, Ron and Seshadhri [14]. In the following, we call the algorithm in [14] the *ERS algorithm*.

High-level description of the ERS algorithm. For the sake of a concise presentation, we will assume that the algorithm is given $\#K_r$. To obtain an estimate when only a lower bound L on $\#K_r$ is available, it is straightforward to use (parallel) geometric search for values greater than L (see Lemma 21 in Appendix C). The ERS algorithm makes use of a notion of *ordered* cliques. More precisely, for any $t \in \{2, \ldots, r\}$, an *ordered t-clique* $\vec{T} = (v_1, \cdots, v_t)$ is a tuple of t vertices such that $T = \{v_1, \cdots, v_t\}$ forms a t-clique (and T is called an *unordered t-clique*).

Let $s_1, \ldots, s_r$ be some parameters. The ERS algorithm iteratively does the following: in iteration 1, it samples a set $\mathcal{R}_1$ of s_1 ordered vertices; in iteration 2, it samples a set $\mathcal{R}_2$ of s_2 ordered edges (incident to the sampled vertices); and then in iteration $t > 2$, it samples a set $\mathcal{R}_t$ of s_t ordered t-cliques, based on the set of $(t-1)$-cliques from the previous iteration. Concretely, the sample set $\mathcal{R}_t$ is obtained by repeating the following s_t times:

(1) sample an ordered clique $\vec{T}$ from $\mathcal{R}_{t-1}$ with probability proportional to $\frac{\deg(\vec{T})}{\deg(\mathcal{R}_{t-1})}$, where $\deg(\vec{T})$ is the degree of the *minimum-degree* vertex in $\vec{T}$ and for a set of order cliques $\mathcal{R}$, $\deg(\mathcal{R}) := \sum_{T \in \mathcal{R}} \deg(\vec{T})$;
(2) select a uniformly random neighbor w of the least degree vertex in $\vec{T}$; and
(3) check if $\vec{T}$ and w forms a t-clique, and if so, add it to $\mathcal{R}_t$.

Once we have these sampled sets $\mathcal{R}_1, \cdots, \mathcal{R}_r$, we can use their *weights* to approximate $\#K_r$. Roughly speaking, for each ordered t-clique $\vec{T}$, its *weight* $\omega(T)$ is a number that is close to the number of r-cliques that are assigned to $\vec{T}$ according to an *assignment rule* specified below. Throughout the process, the ERS algorithm carefully chooses the parameters so that $\omega(\mathcal{R}_1)$ is close to $\frac{\#K_r}{n} \cdot s_1$, and for any $t \geq 1$, $\omega(\mathcal{R}_{t+1})$ is close to $\frac{\omega(\mathcal{R}_t)}{\deg(\mathcal{R}_t)} \cdot s_{t+1}$. Thus, it suffices to compute $\omega(\mathcal{R}_r)$ for estimating $\#K_r$, as for any $t \geq 2$, $\omega(\mathcal{R}_t)$ is close to $\frac{\#K_r}{n} \cdot \frac{s_1 \cdots s_t}{\deg(\mathcal{R}_1) \cdots \deg(\mathcal{R}_{t-1})}$.

To guarantee the above, the ERS algorithm defines an *assignment rule* to assign each r-clique C to an ordered t-clique $\vec{T}$, for any $t \leq r$. To check if an ordered r-clique C is assigned the corresponding unordered clique, the algorithm

(1) considers, for every $t \in \{2, \ldots, r\}$, all the prefixes $\vec{C}_{\leq t}$, where a prefix $\vec{C}_{\leq t}$ of $\vec{C}$ is the ordered t-clique whose vertices are the first t vertices in $\vec{C}$. For each prefix $\vec{C}_{\leq t}$, it invokes a subroutine IsActive to check if it is *active*, which in turn iteratively samples sets $\mathcal{R}_{t+1}, \cdots, \mathcal{R}_r$, starting from $\mathcal{R}_t = \{\vec{C}_{\leq t}\}$ and decides the activeness by the statistics of these sample sets;
(2) if $\vec{C}$ is the lexicographically smallest active ordered r-clique among all active ordered r-cliques induced by C, then it returns 1 (indicating that C is assigned the corresponding unordered clique); otherwise, it returns 0.

For the sake of completeness, we provide pseudo code of this algorithm (i.e., Algorithm 12 CountClique) in Appendix C.

Simplifying the ERS algorithm in the augmented graph model. The above ERS algorithm in [14] was described in the general graph model, in which the algorithm can not perform edge sampling queries. The authors of [14] need to carefully select s_1 so that it can handle different cases of $\#K_r$ and remedy the defect of not being able to sample uniform edges. The choice of s_1 causes an additional term $\min\{\frac{n\lambda^{r-1}}{\#K_r}, \frac{n}{(\#K_r)^{1/r}}\}$ in the query complexity $\min\left\{\frac{n\lambda^{r-1}}{\#K_r}, \frac{n}{(\#K_r)^{1/r}} + \frac{m\lambda^{r-2}}{\#K_r}\right\} \cdot \mathrm{poly}(\log n, 1/\epsilon, r^r)$ of their algorithm (see Theorem 1.1 in [14] and also Lemma 21 in Appendix C).

We note that this algorithm can be simplified in the augmented general graph model, which can be further transformed to the streaming setting by Theorem 11. More precisely, we note that in the augmented graph model, one can directly start with sampling a set $\mathcal{R}_2$ of s_2 edges independently and uniformly at random, and then iteratively sample a set of t-cliques $\mathcal{R}_t$ based on $\mathcal{R}_{t-1}$, for any $3 \leq t \leq r$. That is, there is no need to sample a set $\mathcal{R}_1$ of vertices (and we simply set $\mathcal{R}_1 := E(G)$ and $\deg(\mathcal{R}_1) := m$ at the beginning of the algorithm). Then we choose parameters to ensure that $\omega(\mathcal{R}_2)$ is close $\frac{\#K_r}{m} \cdot s_2$, where $\omega(\mathcal{R}_2)$ is the total weight of ordered edges in $\mathcal{R}_2$ and the weight of an ordered edge is the number of r-cliques assigned to it. Then one can still guarantee that for any $t \geq 3$, $\omega(\mathcal{R}_t)$ is close to $\frac{\#K_r}{m} \cdot \frac{s_2 \cdots s_t}{\deg(\mathcal{R}_2) \cdots \deg(\mathcal{R}_{t-1})}$, by setting the parameters $s_2, \cdots, s_r, \vec{\tau}$ similarly as in [14] (while we start with a slightly different choice s_2 due to the uniform edge sampling). By the analysis in [14], we have the following lemma regarding the performance guarantee of the ERS algorithm in the augmented graph model.

Lemma 19 ([14]). *Given query access to a graph with degeneracy λ in the augmented general graph model, the ERS algorithm has expected running time and query complexity*

$$\frac{m\lambda^{r-2}}{\#K_r} \cdot \mathrm{poly}(\log n, 1/\epsilon, r^r)$$

and outputs a $(1+\epsilon)$-approximation to $\#K_r$ with high probability. Additionally, the maximum query complexity is $O(m+n)$.

5.2 The round-adaptivity of the ERS algorithm

Now we show that the ERS algorithm in the augmented general graph model is an $O(r)$-round algorithm. We describe our streaming version of the ERS algorithm, based on the discussion before. The first observation is that the algorithm consists of two sequentially aligned blocks: sampling the sets $(\mathcal{R}_i)_{i\in\{2,\dots,r\}}$ and checking the assignments. In both blocks, $(i+1)$-cliques are iteratively constructed from i-cliques. These constructions are also inherently sequential. However, constructing multiple i-cliques and some other computations can be done in parallel. Details follow below.

We state the pseudo code of the $5r$-pass streaming version of the ERS in Algorithms 3 and 4. To obtain the final result, we use probability amplification and return the median from running sufficiently many and accordingly parameterized instances of Algorithm 3, which is described in Algorithm 2.

Construction of $(\mathcal{R}_t)_{t\in\{2,\dots,r\}}$. The algorithm StreamApproxClique (Algorithm 3) constructs sets of ordered t-cliques $\mathcal{R}_t$ iteratively, for $t = 2,\dots,r$. Given a set $\mathcal{R}_t$, it calls a 2-pass procedure StreamSet to construct $\mathcal{R}_{t+1}$. After the construction of $\mathcal{R}_r$, it checks how many of the sampled ordered cliques in $\mathcal{R}_t$ are cliques that are assigned their respective unordered clique using a $2r$-pass procedure StrIsAssigned (see Appendix D), and it outputs this number scaled accordingly as its estimate of $\#K_r$.

Sampling $\mathcal{R}_{t+1}$. Given $\mathcal{R}_t$, the procedure StreamSet (Algorithm 4) samples up to s_{t+1} many ordered $(t+1)$-cliques to include into $\mathcal{R}_{t+1}$. To sample *one* ordered $(t+1)$-clique, the algorithm samples an ordered t-clique $\vec{T}$ from $\mathcal{R}_t$ proportionally to $\deg(\vec{T})$. We note that the algorithm can maintain a data structure $d[\mathcal{R}_t]$ for every $t \in \{2,\dots,r\}$ so that this sampling can be done offline without a pass on the input. To select a uniformly random neighbor of the smallest-degree vertex u of $\vec{T}$, the algorithm samples $i \in [\deg(u)]$ and queries the i^{th} neighbor of u in a single pass. In another pass, the algorithm checks whether $(\vec{T}, w)$ is a $(t+1)$-clique and adds it to $\mathcal{R}_{t+1}$ if this is the case. Sampling s_{t+1} ordered $(t+1)$-cliques like this can be parallelized.

Checking the assignments of $\mathcal{R}_r$. Given $\vec{C} \in \mathcal{R}_r$, for an ordered r-clique $\vec{C}'$ that is isomorphic to $\vec{C}$ and a prefix $\vec{C}'_{\le t}$, where $t \in \{2,\dots,r\}$, StrIsAssigned (Algorithm 17 in Appendix D) calls StrAct (Algorithm 18 in Appendix D). The latter is similar to a "warm-start" of (multiple instances of) StreamApprox with $\mathcal{R}_t = \{\vec{C}'_{\le t}\}$: it uses $2(r-t)$ passes to iteratively construct sets $\mathcal{R}_{t+1},\dots,\mathcal{R}_r$ via StreamSet. Then, it returns whether sufficiently many instances of StreamApprox satisfy a threshold condition that can be computed offline.

Since the parameters of a call to StrAct do not depend on calls for other (shorter) prefixes of $\vec{C}'$, all calls corresponding to different prefixes can be parallelized. Note that this differs from the iterative construction of $(\mathcal{R}_t)_{t\in\{2,\dots,r\}}$ in StreamApprox. In addition, the calls corresponding to different ordered r-cliques can be parallelized. Once the algorithm has determined, for all ordered r-cliques and their prefixes, whether they are active, the fully active ordered r-cliques can be compared lexicographically against the sampled r-clique $\vec{C}$. If $\vec{C}$ is active and lexicographically smallest, the algorithm accepts, otherwise it rejects. It follows that due to parallel computation, we require as many passes as the most costly call to StrAct, which is at most $2r$ passes.

Theorem 20. *The ERS algorithm in the augmented graph model can be implemented as a $5r$-round adaptive algorithm.*

Applying Theorem 9 proves Theorem 2.

Algorithm 2 Approximately counting the number of K_r

procedure StreamCountClique(n, r, λ, ϵ)
  $\gamma \leftarrow \epsilon/(8r\cdot r!)$, $\beta \leftarrow 1/(6r)$
  $\tau_r \leftarrow 1$; for each $t \in [2, r-1]$, set $\tau_t \leftarrow \frac{r^{4r}}{\beta^r\cdot\gamma^2}\cdot\lambda^{r-t}$; $\vec{\tau} \leftarrow \{\tau_2,\dots,\tau_r\}$
  for all $j = 1,\dots,q = \Theta(\log(n))$ **do**
    Invoke StreamApproxCliques($n, r, \lambda, \epsilon, \vec{\tau}$).
    Let χ_j be the returned value.
  Let $\hat{n}_r$ be the the median value of $\chi_1,\cdots,\chi_q$
  return $\hat{n}_r$.

Algorithm 3 The basic subroutine

procedure StreamApproxClique($n, r, \lambda, \epsilon, \vec{\tau}$)
  set $\tilde{\omega}_1 \leftarrow (1-\epsilon/2)\#K_r$, $\beta \leftarrow 1/(18r)$, $\gamma \leftarrow \epsilon/(2r)$
  virtually set $\mathcal{R}_1 \leftarrow E$
  pass 1; input:
    count number of edges m and set $\deg(\mathcal{R}_1) \leftarrow m$
  pass 2; input: m
    sample $s_2 \leftarrow \lceil \frac{m\cdot\tau_2}{\tilde{\omega}_0}\cdot\frac{3\ln(2/\beta)}{\gamma^2}\rceil$ edges u.a.r
      and let $\mathcal{R}_2$ be the chosen multiset ▷ f_1
  pass 3; input: $\mathcal{R}_2$
    construct $d[\mathcal{R}_2]$ ▷ f_2
  for all $t \in \{2,\dots,r-1\}$ **do**
    $\deg(\mathcal{R}_t) \leftarrow \sum_{\vec{T}\in\mathcal{R}_t}\deg(\vec{T}) = \sum_{\vec{T}\in\mathcal{R}_t}\min_{v\in\vec{T}} d[\mathcal{R}_t]_v$
    $\tilde{\omega}_t \leftarrow (1-\gamma)\frac{\tilde{\omega}_{t-1}}{\deg(\mathcal{R}_{t-1})}\cdot s_t$
      and $s_{t+1} \leftarrow \lceil \frac{\deg(\mathcal{R}_t)\tau_{t+1}}{\tilde{\omega}_t}\cdot\frac{3\ln(2/\beta)}{\gamma^2}\rceil$
    if $s_{t+1} > \frac{4m\lambda^{t-1}\cdot\tau_{t+1}}{\#K_r}\cdot\frac{(r!)^2\cdot 3\ln(2/\beta)}{\beta^t\cdot\gamma^2}$, **then abort**
    passes 2t to 2t+1; input: $t, \mathcal{R}_t, d[\mathcal{R}_t], s_{t+1}$
      $\mathcal{R}_{t+1}, d[\mathcal{R}_{t+1}] \leftarrow$ StreamSet($t, \mathcal{R}_t, d[\mathcal{R}_t], s_{t+1}$)
  passes 2r to 4r+1; input:
    let $\hat{n}_r = \frac{m\cdot\deg(\mathcal{R}_2)\cdot\cdots\cdot\deg(\mathcal{R}_{r-1})}{s_1\cdot\cdots\cdot s_r}\cdot\sum_{\vec{C}\in\mathcal{R}_r}$ StrIsAssigned($\vec{C}, r, \lambda, \epsilon, m, \vec{\tau}$)
  return $\hat{n}_r$

6 CONCLUSION

We studied the problem of estimating the number of occurrences of a subgraph H in a graph G in the streaming setting. We provide a transformation that converts sublinear-time graph algorithms in the query access model to sublinear-space streaming algorithms. For an arbitrary subgraph H, we obtained a 3-pass algorithm in the turnstile streaming model with space complexity $\tilde{O}(\frac{m^{\rho(H)}}{\epsilon^2\cdot\#H})$ for

Algorithm 4 Sample K_{t+1}-candidates from a stream

procedure StreamSet($t, \mathcal{R}_t, d[\mathcal{R}_t], s_{t+1}$)
  set up a data structure $\mathcal{D}$ to sample each $\vec{T} \in \mathcal{R}_t$ with probability $\mathrm{dg}(\vec{T})/\mathrm{dg}(\mathcal{R}_t)$
  initialize $\mathcal{R}_{t+1} = \emptyset, d[\mathcal{R}_{t+1}]$
  parallel for all $\ell \in [s_{t+1}]$
    invoke $\mathcal{D}$ to generate $\vec{T}_\ell$
    $u \leftarrow$ minimum degree vertex of $\vec{T}_\ell$
    pass 1; input: $u, d[\mathcal{R}_t]_u$
      query a random neighbor w of u ▷ f_3
    pass 2; input: $\vec{T}_\ell, w$
      if $(\vec{T}_\ell, w)$ is a $(t+1)$-clique, **then** add it to $\mathcal{R}_{t+1}$ ▷ f_4
      update $d[\mathcal{R}_{t+1}]$ ▷ f_2
  return $\mathcal{R}_{t+1}, d[\mathcal{R}_{t+1}]$

$(1 \pm \epsilon)$-approximating $\#H$, where $\rho(H)$ is the fractional edge-cover of H. For a clique $\#K_r$ such that $r \geq 3$, we obtained a constant-pass streaming algorithm for $(1 \pm \epsilon)$-approximating $\#K_r$ in $\tilde{O}(\frac{m\lambda^{r-2}}{\epsilon^2 \cdot \#K_r})$ space, in a graph G with degeneracy λ.

It would be interesting to reduce the number of passes of our algorithms even further: Can we obtain a 2-pass algorithm for $\#H$ with space complexity $\tilde{O}(\frac{m^{\rho(H)}}{\epsilon^2 \cdot \#H})$? Can we achieve a C-pass streaming algorithm for $\#K_r$ with space complexity $\tilde{O}(\frac{m\lambda^{r-2}}{\epsilon^2 \cdot \#K_r})$ space in a graph G with degeneracy λ, for some universal constant C that does not depend on r?

ACKNOWLEDGMENTS

P.P. is supported by "the Fundamental Research Funds for the Central Universities".

REFERENCES

[1] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2012. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*. 5–14.

[2] Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. 2019. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. In *10th Innovations in Theoretical Computer Science, ITCS 2019*. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 6.

[3] Albert Atserias, Martin Grohe, and Dániel Marx. 2008. Size bounds and query plans for relational joins. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 739–748.

[4] Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. 2002. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*. 623–632.

[5] Suman K Bera and Amit Chakrabarti. 2017. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *34th Symposium on Theoretical Aspects of Computer Science*.

[6] Suman K. Bera and C. Seshadhri. 2020. How the Degeneracy Helps for Triangle Counting in Graph Streams. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'20)*. Association for Computing Machinery, New York, NY, USA, 457–467. https://doi.org/10.1145/3375395.3387665

[7] Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. 2013. How hard is counting triangles in the streaming model?. In *International Colloquium on Automata, Languages, and Programming*. Springer, 244–254.

[8] Laurent Bulteau, Vincent Froese, Konstantin Kutzkov, and Rasmus Pagh. 2016. Triangle counting in dynamic graph streams. *Algorithmica* 76, 1 (2016), 259–278.

[9] Luciana S Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. 2006. Counting triangles in data streams. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 253–262.

[10] Clément L. Canonne and Tom Gur. 2018. An Adaptivity Hierarchy Theorem for Property Testing. *computational complexity* 27, 4 (Dec. 2018), 671–716. https://doi.org/10.1007/s00037-018-0168-4

[11] Graham Cormode and Donatella Firmani. 2014. A Unifying Framework for l0-Sampling Algorithms. *Distributed and Parallel Databases* 32, 3 (Sept. 2014), 315–335. https://doi.org/10.1007/s10619-013-7131-9

[12] Graham Cormode and Hossein Jowhari. 2014. A second look at counting triangles in graph streams. *Theoretical Computer Science* 552 (2014), 44–51.

[13] Graham Cormode and Hossein Jowhari. 2017. A second look at counting triangles in graph streams (corrected). *Theoretical Computer Science* 683 (2017), 22–30.

[14] Talya Eden, Dana Ron, and C Seshadhri. 2020. Faster sublinear approximation of the number of k-cliques in low-arboricity graphs. In *Symposium on Algorithms and Data Structures (SODA)*.

[15] Hendrik Fichtenberger, Mingze Gao, and Pan Peng. 2020. Sampling Arbitrary Subgraphs Exactly Uniformly in Sublinear Time. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[16] Joshua A Grochow and Manolis Kellis. 2007. Network motif discovery using subgraph enumeration and symmetry-breaking. In *Annual International Conference on Research in Computational Molecular Biology*. Springer, 92–106.

[17] Rajesh Jayaram and John Kallaugher. 2021. An Optimal Algorithm for Triangle Counting. *arXiv preprint arXiv:2105.01785* (2021).

[18] Hossein Jowhari and Mohammad Ghodsi. 2005. New streaming algorithms for counting triangles in graphs. In *International Computing and Combinatorics Conference*. Springer, 710–716.

[19] John Kallaugher, Michael Kapralov, and Eric Price. 2018. The sketching complexity of graph and hypergraph counting. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 556–567.

[20] John Kallaugher, Andrew McGregor, Eric Price, and Sofya Vorotnikova. 2019. The complexity of counting cycles in the adjacency list streaming model. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 119–133.

[21] John Kallaugher and Eric Price. 2017. A hybrid sampling scheme for triangle counting. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1778–1797.

[22] Daniel M Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. 2012. Counting arbitrary subgraphs in data streams. In *International Colloquium on Automata, Languages, and Programming*. Springer, 598–609.

[23] Mihail N Kolountzakis, Gary L Miller, Richard Peng, and Charalampos E Tsourakakis. 2012. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics* 8, 1-2 (2012), 161–185.

[24] Madhusudan Manjunath, Kurt Mehlhorn, Konstantinos Panagiotou, and He Sun. 2011. Approximate counting of cycles in streams. In *European Symposium on Algorithms*. Springer, 677–688.

[25] Andrew McGregor and Sofya Vorotnikova. 2020. Triangle and four cycle counting in the data stream model. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 445–456.

[26] Andrew McGregor, Sofya Vorotnikova, and Hoa T Vu. 2016. Better algorithms for counting triangles in data streams. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 401–411.

[27] Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2018. Worst-case optimal join algorithms. *Journal of the ACM (JACM)* 65, 3 (2018), 1–40.

[28] Rasmus Pagh and Charalampos E Tsourakakis. 2012. Colorful triangle counting and a mapreduce implementation. *Inform. Process. Lett.* 112, 7 (2012), 277–281.

[29] A Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. 2013. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment* 6, 14 (2013), 1870–1881.

[30] Alexander Schrijver. 2003. *Combinatorial optimization: polyhedra and efficiency*. Vol. 24. Springer Science & Business Media.

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

A TURNSTILE SUBGRAPH COUNTING ALGORITHM

Lemma 18. *Let $\epsilon > 0$, and let H be an arbitrary subgraph of constant size. There exists an algorithm for the relaxed augmented general graph model that has space complexity $O(\log^4 n)$ and returns a copy of H or nothing. For any fixed copy of H in the input graph, it is returned with probability $(1 \pm \epsilon)/(2m)^{\rho(H)}$.*

Proof. Let $C = (c_1, \ldots, c_\alpha), S = (s_1, \ldots, s_\beta)$ be a decomposition of H into odd-length cycles and stars that satisfies the guarantees of Lemma 4 and that is passed to Algorithm 1. We slightly modify Algorithm 1 as follows. In Algorithm 1, we replace Lines 10 and 11 by setting w_i to the answer of a query $f_3(u_{i,1})$, i.e., an approximately uniformly random neighbor of $u_{i,1}$. After Line 21, we sample a uniformly random number t from $[\sqrt{2m}]$ and check whether it is at most $\deg(u_{i,1})$. Since no additional queries are asked, the space complexity follows from Lemma 15 and Theorem 11. The modified algorithm is provied in Algorithm 5.

Let $i \in [\beta]$ and set $k := s_i$. Let $(u_0, \ldots, u_k)$ be a *canonical* k-star in G, and let p be the probability that it is sampled by queries of type f_1. Then, we have $(1/(2m) - 1/n^c)^k \le p \le (1/(2m) + 1/n^c)^k$.

Let $i \in [\alpha]$ and set $k := (c_i - 1)/2$. Let $(u_1, \ldots, u_{2k+1})$ be a *canonical* odd-length cycle in G. Let p be the probability that $((u_1, u_2), (u_3, u_4), \ldots, (u_{2k-1}, u_{2k}))$ are sampled via queries of type f_1. Then, we have $(1/(2m) - 1/n^c)^k \le p \le (1/(2m) + 1/n^c)^k$. Let q be the probability that $u' := u_{2k+1}$ is sampled. We distinguish two cases: $\deg(u_1) \le \sqrt{2m}$ and $\deg(u_1) > \sqrt{2m}$. If $\deg(u_1) \le \sqrt{2m}$, then $u' = w_i$ is sampled according to the modification described above via a query of type f_3 and

$$\frac{\deg(u_1)}{\sqrt{2m}} \cdot \left(\frac{1}{\deg(u_1)} - \frac{1}{n^c}\right) \le q \le \frac{\deg(u_1)}{\sqrt{2m}} \cdot \left(\frac{1}{\deg(u_1)} + \frac{1}{n^c}\right).$$

Now, consider the case $\deg(u') > \sqrt{2m}$. Then, $u' = u_{i,0}$ is sampled via a query of type f_1. Note that sampling a vertex *exactly* proportional to its degree is equivalent to sampling an edge *exactly* uniformly at random and choosing one of its endpoints by flipping a fair coin. Since f_1 returns exactly one edge, for any $e, e' \in E$, $e \neq e'$, the events of sampling e and sampling e' are disjoint. For the corresponding query of type f_1 in the *relaxed* augmented general graph model it follows that

$$\frac{\sqrt{2m}}{\deg(u')} \cdot \left(\frac{\deg(u')}{2m} - \frac{\deg(u')}{n^c}\right) \le q \le \frac{\sqrt{2m}}{\deg(u')} \cdot \left(\frac{\deg(u')}{2m} + \frac{\deg(u')}{n^c}\right).$$

Fix a copy of H and let p be the probability that H is sampled. It follows from the discussion above that

$$\frac{1}{(2m)^{\rho(H)}} - \frac{2^{|H|}n}{n^c} \le p \le \frac{1}{(2m)^{\rho(H)}} + \frac{2^{|H|}n}{n^c}$$

Choosing $c = 5|H| \ge \log(2^{|H|}n(2m)^{\rho(H)}/\epsilon)/\log(n)$ concludes the proof. □

Algorithm 5 Sampling a subgraph candidate from a stream

1: **procedure** StreamSubg($C = (c_1, \ldots, c_\alpha), S = (s_1, \ldots, s_\beta)$)
2: **pass 1**; input: C, S
3: **for all** $c_i \in C$ **do**
4: sample $\lceil c_i/2 \rceil + 1$ edges $C_i = \{u_{i,0}, v_{i,0}, \ldots, u_{i,\lceil c_i/2 \rceil}, v_{i,\lceil c_i/2 \rceil}\}$ ▷ f_1
5: **for all** $s_i \in S$ **do**
6: sample k edges $S_i = \{x_{i,1}, y_{i,1}, \ldots, x_{i,k}, y_{i,k}\}$ ▷ f_1
7: count the number of edges m
8: **pass 2**; input: $C, S, (C_i)_{i\in[\alpha]}, (S_i)_{i\in[\beta]}, m$
9: **for all** $c_i \in C$ **do**
10: $w_i \leftarrow$ random neighbor of $u_{i,1}$ ▷ f_3
11: $C'_i = C_i \cup \{w_i\}$
12: **pass 3**; input: $C, S, (C'_i)_{i\in[\alpha]}, (S_i)_{i\in[\beta]}, m$
13: **for all** $z, z' \in (C'_i)_{i\in[\alpha]} \cup (S_i)_{i\in[\beta]}$ **do**
14: check whether $(z, z') \in E$ ▷ f_4
15: count degree of z ▷ f_2
16: Let $V' := (C'_i)_{i\in[\alpha]} \cup (S_i)_{i\in[\beta]}$, $E' := E \cap (V' \times V')$, $V'' := \emptyset$
17:
18: ▷ Postprocessing
19: **for all** $c_i \in C'_i$ **do**
20: **if** $d[V']_{u_{i,1}} \le \sqrt{2m}$ **then**
21: sample $t \in [\sqrt{2m}]$ uniformly at random
22: check if $t \le \deg(u_{i,1})$
23: check if $(w_i, u_{i,1}, v_{i,1}, \ldots, u_{i,\lceil c_i/2 \rceil}, v_{i,\lceil c_i/2 \rceil})$ is a canonical c_i-cycle in $(E', \prec_G)$
24: $V'' = V'' \cup \{w_i, u_{i,1}, \ldots, u_{i,c_i}, v_{i,1}, \ldots, v_{i,\alpha}\}$
25: **else**
26: sample $t \in [0, 1]$ uniformly at random
27: check if $t \le \sqrt{2m}/\deg(u_{i,0})$
28: check if $(u_{i,0}, u_{i,1}, v_{i,1}, \ldots, u_{i,\lceil c_i/2 \rceil}, v_{i,\lceil c_i/2 \rceil})$ is a canonical c_i-cycle in $(E', \prec_G)$
29: $V'' = V'' \cup \{u_{i,0}, u_{i,1}, \ldots, u_{i,c_i}, v_{i,1}, \ldots, v_{i,\alpha}\}$
30: **for all** $s_i \in S_i$ **do**
31: check if $x_{i,1} = \ldots = x_{i,s_i}$ and $(x_{i,1}, y_{i,1}, \ldots, y_{i,s_i})$ is a canonical s_i-star in $(E'', \prec_G)$
32: $V'' = V'' \cup \{x_{i,1}, y_{i,1}, \ldots, y_{i,s_i}\}$
33: check if $G[V'']$ contains a copy H_G of H as subgraph
34: **return** H_G if all checks were successful

B A SUBLINEAR-TIME ALGORITHM FOR APPROXIMATING #H OF A GENERAL GRAPH

Now we present the pseudo code of the subliner-time algorithm for approximating sampling and counting an arbitrary subgraph in the query access model given in [15]. The FGP algorithm refers to Algorithm 9 (SampleSubgraph), which invokes two subroutines Algorithm 7 (SampleOddCycle) and Algorithm 8 (SampleStar) for sampling an odd length cycle and a star, respectively.

Algorithm 6 Sampling a wedge

procedure SampleWedge(G, u, v)
  if $d_u \le \sqrt{2m}$ **then**
    sample a number $i \in \{1, \dots \sqrt{2m}\}$ uniformly at random
    if $i > d_u$ **then**
      return Fail
    $w \leftarrow i^{th}$ neighbor of u
  else
    sample a vertex w with prob. proportional to its degree
    sample $t \in [0, 1]$ uniformly at random
    if $t > \sqrt{2m}/\mathrm{dg}(w)$ **then**
      return Fail
  return w

Algorithm 7 Sampling a cycle of length $2k + 1$

procedure SampleOddCycle($G, 2k + 1$)
  Obtain k directed edges $(u_1, v_1), \dots, (u_k, v_k)$
    by calling SampleEdge k times
  if $u_1, v_1, \dots, u_k, v_k$ is a path of length $2k - 1$
      and $u_1 \prec v_1, \forall i > 1 : u_1 \prec u_i, v_i$ **then**
    if SampleWedge(G, u_1, v_k) returns w and $w \prec v_1$ **then**
      return $\{(u_1, v_1), \dots, (u_k, v_k)\} \cup \{(v_k, w), (w, u_1)\}$
  return Fail

Algorithm 8 Sampling a star with k petals

procedure SampleStar(G, k)
  Obtain k directed edges $(u_1, v_1), \dots, (u_k, v_k)$
    by calling SampleEdge k times
  if $u_1 = u_2 = \dots = u_k$ and $v_1 \prec v_2 \prec \dots \prec v_k$ **then**
    return $(u_1, v_1, \dots, v_k)$
  return Fail

Algorithm 9 Sampling a copy of subgraph H

procedure SampleSubgraph(G, H)
  Let $\overline{T} = \{\overline{C_1}, \dots, \overline{C_o}, \overline{S_1}, \dots, \overline{S_s}\}$ denote
    a (decomposition) type of H.
  for all $i = 1 \dots o$ **do**
    if SampleOddCycle($G, |E(\overline{C_i})|$) returns a cycle C **then**
      $C_i \leftarrow C$
    else
      return Fail
  for all $j = 1 \dots s$ **do**
    if SampleStar($G, |V(\overline{S}_j)| - 1$) returns a star $\mathcal{S}$ **then**
      $\mathcal{S}_j \leftarrow \mathcal{S}$
    else
      return Fail
  Query all edges $(\bigcup_{i\in[o]} V(C_i) \cup \bigcup_{j\in[s]} V(\mathcal{S}_j))^2$
  if $S := (C_1, \dots, C_o, \mathcal{S}_1, \dots, \mathcal{S}_s)$ forms a copy of H **then**
    flip a coin and with probability $\frac{1}{f_{\overline{T}}(H)}$: **return** S
  return Fail

The authors of [15] then make use of the FGP algorithm as a subroutine to obtain a uniform sampler of a copy of H (i.e., Algorithm 10 SampleSubgraphUniformly) and an estimator of $\#H$ (i.e., Algorithm 11 CountSubgraph).

Algorithm 10 Sampling a copy of subgraph H uniformly at random

procedure SampleSubgraphUniformly(G, H)
  for all $j = 1, \dots, q = 10 \cdot (2m)^{\rho(H)}/T$ **do**
    Invoke SampleSubgraph(G, H)
    if a subgraph H is returned **then**
      return H
  return Fail

Algorithm 11 Approximately counting the number of instances of H

procedure CountSubgraph(G, H)
  $X = 0$
  for all $j = 1, \dots, q = 10 \cdot (2m)^{\rho(H)}/(T\epsilon^2)$ **do**
    Invoke SampleSubgraph(G, H)
    if a subgraph H is returned **then**
      $X \leftarrow X + 1$
  return X

C A SUBLINEAR-TIME ALGORITHM FOR APPROXIMATING $\#K_r$ OF A GRAPH WITH DEGENERACY AT MOST λ

This section lists pseudo code for the algorithm from [6]. The ERS-algorithm refers to Algorithm 12 CountClique, which invokes $\Theta(\log n)$ times Algorithm 13 ApproxClique and takes the median of these outputs.

In Algorithm 13 ApproxClique, it invokes a subroutine Algorithm 14 for sampling a set of larger cliques, and a subroutine Algorithm 15 IsAssigned for checking if an ordered r-clique $\vec{C}$ is assigned or not. Finally, Algorithm 15 IsAssigned invokes Algorithm 16 IsActive to check if all the prefixes of $\vec{C}$ is active or not.

Lemma 21 ([14]). *Let G be a graph with degeneracy λ. The ERS algorithm (i.e., Algorithm 12) in the general graph model satisfies the following:*

- *if $L_r \in [\frac{\#K_r}{4}, \#K_r]$, then CountClique$(n, r, \lambda, \epsilon, m, L_r)$ outputs a value $\hat{n}_r$ such that with probability at least $1 - n^{-\Omega(1)}$, $\hat{n}_r$ is a $(1 \pm \varepsilon)$-approximation of $\#K_r$;*
- *if $L_r > \#K_r$, then CountClique$(n, r, \lambda, \epsilon, m, L_r)$ outputs a value $\hat{n}_r$ such that with probability at least $1-n^{-\Omega(1)}$, $\hat{n}_r < L_r$;*
- *the expected running time and query complexity of the algorithm are*

$$O\left(\min\{\frac{n\lambda^{r-1}}{L_r}, \frac{n}{(\#K_r)^{1/r}}\} + \frac{m\lambda^{r-1}}{L_r} \cdot \frac{\#K_r}{L_r}\right) \cdot \mathrm{poly}(\log n, 1/\varepsilon, r^r)$$

Algorithm 12 Approximately counting the number of instances of K_r

procedure CountClique($n, r, \lambda, \epsilon, m, L_r$)
  $\gamma \leftarrow \epsilon/(8r \cdot r!)$, $\beta \leftarrow 1/(6r)$
  for each $t \in [2, r-1]$, set $\tau_t \leftarrow \frac{r^{4r}}{\beta^r \cdot \gamma^2} \cdot \lambda^{r-t}$;
    $\tau_r \leftarrow 1$; $\tau_1 \leftarrow \frac{r^{4r}}{\gamma^2} \cdot \min\{\lambda^{r-1}, L_r^{(r-1)/r}\}$,
    $\vec{\tau} \leftarrow \{\tau_1, \ldots, \tau_r\}$
  for all $j = 1, \ldots, q = \Theta(\log(n))$ **do**
    Invoke ApproxCliques($n, r, \lambda, \epsilon, L_r, m, \vec{\tau}$).
    Let χ_j be the returned value.
  Let $\hat{n}_r$ be the the median value of $\chi_1, \cdots, \chi_q$
  return $\hat{n}_r$.

Algorithm 13 Approximately counting the number of instances of K_r

procedure ApproxClique($n, r, \lambda, \epsilon, L_r, m, \vec{\tau}$)
  set $\mathcal{R}_0 \leftarrow V$, $\mathrm{dg}(\mathcal{R}_0) \leftarrow n$, $\tilde{\omega}_0 = (1-\epsilon/2)L_r$, $\beta \leftarrow 1/(18r)$
    and $\gamma \leftarrow \epsilon/(2r)$
  sample $s_1 = \lceil \frac{n\tau_1}{\tilde{\omega}_0} \cdot \frac{3\ln(2/\beta)}{\gamma^2} \rceil$ vertices u.a.r
    and let $\mathcal{R}_1$ be the chosen multiset
  for all $t = 1, \ldots, r-1$ **do**
    Compute $\mathrm{dg}(\mathcal{R}_t)$ and set $\tilde{\omega}_t = (1-\gamma)\frac{\tilde{\omega}_{t-1}}{\mathrm{dg}(\mathcal{R}_{t-1})} \cdot s_t$
      and $s_{t+1} \leftarrow \lceil \frac{\mathrm{dg}(\mathcal{R}_t)\tau_{t+1}}{\tilde{\omega}_t} \cdot \frac{3\ln(2/\beta)}{\gamma^2} \rceil$
    If $s_{t+1} > \frac{4m\lambda^{t-1}\cdot\tau_{t+1}}{L_r} \cdot \frac{(r!)^2 \cdot 3\ln(2/\beta)}{\beta^t \cdot \gamma^2}$ then **abort**
    Invoke SampleASet($t, \mathcal{R}_t, s_{t+1}$)
      and let $\mathcal{R}_{t+1}$ be the returned multiset.
  $\hat{n}_r = \frac{n \cdot \mathrm{dg}(\mathcal{R}_1) \cdot \cdots \cdot \mathrm{dg}(\mathcal{R}_{r-1})}{s_1 \cdot \cdots \cdot s_r} \sum_{\vec{C} \in \mathcal{R}_r} \mathrm{IsAssigned}(\vec{C}, r, \lambda, \epsilon, L_r, m, \vec{\tau})$
  return $\hat{n}_r$.

Algorithm 14 Sampling a set of ordered $(t+1)$-cliques

procedure SampleASet($t, \mathcal{R}_t, s_{t+1}$)
  Compute $\mathrm{dg}(\mathcal{R}_t)$ and set up a data structure
    to sample each $\vec{T} \in \mathcal{R}_t$ with probability $\mathrm{dg}(\vec{T})/\mathrm{dg}(\mathcal{R}_t)$
  initialize $\mathcal{R}_t = \emptyset$
  for all $\ell = 1, \ldots, s_{t+1}$ **do**
    invoke the above to generate $\vec{T}_\ell$
    find the minimum degree vertex u of $\vec{T}_\ell$
    sample a random neighbor w of u
    If the $(t+1)$-tuple $(\vec{T}_\ell, w)$ is an ordered $(t+1)$-clique, add it to $\mathcal{R}_{t+1}$
  return $\mathcal{R}_{t+1}$.

Algorithm 15 Check if an ordered r-clique $\vec{C}$ is assigned to the un-ordered clique

procedure IsAssigned($\vec{C}, r, \lambda, \epsilon, L_r, m, \vec{\tau}$)
  Let C be the un-ordered clique corresponding to $\vec{C}$
  for all ordered r-clique $\vec{C}'$ whose un-ordered clique equals C **do**
    for all prefix $\vec{C}'_{\le t}$, $t \in [r-1]$ **do**
      invoke IsActive($t, \vec{C}'_{\le t}, r, \lambda, \epsilon, L_r, m, \vec{\tau}$) and if it returns **Non-Active** then **abort** and **return** 0
  if $\vec{C}$ is the lexicographically first ordered r-clique in the above set of active ordered cliques **then**
    return 1
  else
    return 0.

Algorithm 16 Check if an ordered i-clique $\vec{I}$ is active

procedure IsActive($i, \vec{I}, r, \lambda, \epsilon, L_r, m, \vec{\tau}$)
  for all $\ell = 1, \cdots, q = 12\ln(n^{r+10})$ **do**
    set $\mathcal{R}_i = \{\vec{I}\}$, $\tilde{\omega}_i = (1-\epsilon/2)\tau_i$, $\beta = 1/(6r)$,
      and $\gamma = \epsilon/(8r \cdot r!)$.
    for all $t = i, \ldots, r-1$ **do**
      Compute $\mathrm{dg}(\mathcal{R}_t)$
      for $t > i$, set $\tilde{\omega}_t = (1-\gamma)\frac{\tilde{\omega}_{t-1}\cdot s_t}{\mathrm{dg}(\mathcal{R}_{t-1})}$
        and $s_{t+1} = \frac{\mathrm{dg}(\mathcal{R}_t)\cdot\tau_{t+1}}{\tilde{\omega}_t} \cdot \frac{3\ln(2/\beta)}{\gamma^2}$.
      if $s_{t+1} > \frac{2m\lambda^{t-1}\cdot\tau_{t+1}}{L_r} \cdot \frac{12\ln(1/\beta)}{\beta^r \cdot \gamma^3}$,
        then set $\chi_\ell = 0$ and continue to next ℓ.
      invoke SampleASet($t, \mathcal{R}_t, s_{t+1}$)
        and let $\mathcal{R}_{t+1}$ be the returned multiset.
    set $\hat{c}_r(\vec{I}) = \frac{\mathrm{dg}(\mathcal{R}_i)\cdot\cdots\cdot\mathrm{dg}(\mathcal{R}_{r-1})}{s_{i+1}\cdot\cdots\cdot s_r} \cdot |\mathcal{R}_r|$.
    if $\hat{c}_r(\vec{I}) \le \frac{\tau_i}{4}$, then $\chi_\ell = 1$, otherwise $\chi_\ell = 0$.
  if $\sum_{\ell=1}^{q} \chi_\ell \ge q/2$ **then**
    return Active
  else
    return Non-Active.

D MISSING ALGORITHMS FROM SECTION 5.2

Algorithm 17 Check if an ordered r-clique $\vec{C}$ is assigned to its unordered clique

procedure StrIsAssigned($\vec{C}, r, \lambda, \epsilon, m, \vec{\tau}$)
  $C \leftarrow$ the unordered clique corresponding to $\vec{C}$
  initialize dictionary A
  parallel for all ordered r-cliques $\vec{C}'$ isomorphic to C
    parallel for all $t \in \{2, \ldots, r\}$
      passes 1 to 2t-1; input: $t, \vec{C}'_{\leq t}, r, \lambda, \epsilon, m, \vec{\tau}$
        $A[\vec{C}'_{\leq t}] \leftarrow$ StrAct$(t, \vec{C}'_{\leq t}, r, \lambda, \epsilon, m, \vec{\tau})$
  for ordered r-clique $\vec{C}'$ of C **do**
    if $\vec{C}'$ "lex. <" $\vec{C}$
      $\wedge \forall t \in \{2, \ldots, r\} : A[\vec{C}'_{\leq t}] =$ **active then**
      return 0
  return 1

Algorithm 18 Check if an ordered i-clique $\vec{I}$ is active

procedure StrAct($i, \vec{I}, r, \lambda, \epsilon, m, \vec{\tau}$)
  parallel for all $\ell \leftarrow 1, \ldots, q = 12 \ln(n^{r+10}/\delta)$
    $\mathcal{R}_i \leftarrow \{\vec{I}\}$, $\tilde{\omega}_i \leftarrow (1 - \epsilon/2)\tau_i$, $\beta \leftarrow 1/(6r)$, and $\gamma \leftarrow \epsilon/(8r \cdot r!)$
    pass 1; input: $\mathcal{R}_i$
      construct $d[\mathcal{R}_i]$ ▷ f_2
    for all $t \leftarrow i, \ldots, r-1$ **do**
      $\deg(\mathcal{R}_t) \leftarrow \sum_{\vec{T} \in \mathcal{R}_t} \deg(\vec{T}) = \sum_{\vec{T} \in \mathcal{R}_t} \min_{v \in \vec{T}} d[\mathcal{R}_t]_v$
      for $t > i$, set $\tilde{\omega}_t = (1 - \gamma)\frac{\tilde{\omega}_{t-1} \cdot s_t}{\deg(\mathcal{R}_{t-1})}$ and $s_{t+1} = \frac{\deg(\mathcal{R}_t) \cdot \tau_{t+1}}{\tilde{\omega}_t} \cdot \frac{3 \ln(2/\beta)}{\gamma^2}$
      if $s_{t+1} > \frac{2m\lambda^{t-1} \cdot \tau_{t+1}}{\#K_r} \cdot \frac{12 \ln(1/\beta)}{\beta^r \cdot \gamma^3}$ **then**
        $\chi_\ell \leftarrow 0$ and break loop iteration for ℓ
      passes 2t to 2t+1; input: $t, \mathcal{R}_t, d[\mathcal{R}_t], s_{t+1}$
        $\mathcal{R}_{t+1}, d[\mathcal{R}_{t+1}] \leftarrow$ StreamSet$(t, \mathcal{R}_t, d[\mathcal{R}_t], s_{t+1})$
    $\hat{c}_r(\vec{I}) \leftarrow \frac{\deg(\mathcal{R}_i) \cdot \cdots \cdot \deg(\mathcal{R}_{r-1})}{s_{i+1} \cdot \cdots \cdot s_r} \cdot |\mathcal{R}_r|$
    if $\hat{c}_r(\vec{I}) \leq \frac{\tau_i}{4}$, **then** $\chi_\ell \leftarrow 1$, **else** $\chi_\ell \leftarrow 0$
  if $\sum_{\ell=1}^{q} \chi_\ell \geq q/2$ **then**
    return active
  else
    return non-active