Counting Simplices in Hypergraph Streams

Amit Chakrabarti
Dartmouth College, Hanover, NH, USA

Themistoklis Haris
Dartmouth College, Hanover, NH, USA

Abstract

We consider the problem of space-efficiently estimating the number of *simplices* in a hypergraph stream. This is the most natural hypergraph generalization of the highly-studied problem of estimating the number of *triangles* in a graph stream. Our input is a k-uniform hypergraph H with n vertices and m hyperedges, each hyperedge being a k-sized subset of vertices. A k-simplex in H is a subhypergraph on $k+1$ vertices X such that all $k+1$ possible hyperedges among X exist in H. The goal is to process the hyperedges of H, which arrive in an arbitrary order as a data stream, and compute a good estimate of $T_k(H)$, the number of k-simplices in H.

We design a suite of algorithms for this problem. As with triangle-counting in graphs (which is the special case $k = 2$), sublinear space is achievable but only under a promise of the form $T_k(H) \geq T$. Under such a promise, our algorithms use at most four passes and together imply a space bound of

$$O\left(\varepsilon^{-2} \log \delta^{-1} \operatorname{polylog} n \cdot \min\left\{\frac{m^{1+1/k}}{T}, \frac{m}{T^{2/(k+1)}}\right\}\right)$$

for each fixed $k \geq 3$, in order to guarantee an estimate within $(1 \pm \varepsilon)T_k(H)$ with probability $\geq 1 - \delta$. We also give a simpler 1-pass algorithm that achieves $O\left(\varepsilon^{-2} \log \delta^{-1} \log n \cdot (m/T)\left(\Delta_E + \Delta_V^{1-1/k}\right)\right)$ space, where Δ_E (respectively, Δ_V) denotes the maximum number of k-simplices that share a hyperedge (respectively, a vertex), which generalizes a previous result for the $k = 2$ case. We complement these algorithmic results with space lower bounds of the form $\Omega(\varepsilon^{-2})$, $\Omega(m^{1+1/k}/T)$, $\Omega(m/T^{1-1/k})$ and $\Omega(m\Delta_V^{1/k}/T)$ for multi-pass algorithms and $\Omega(m\Delta_E/T)$ for 1-pass algorithms, which show that some of the dependencies on parameters in our upper bounds are nearly tight. Our techniques extend and generalize several different ideas previously developed for triangle counting in graphs, using appropriate innovations to handle the more complicated combinatorics of hypergraphs.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases data streaming, graph algorithms, hypergraphs, sub-linear algorithms, triangle counting

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.32

Related Version *Full Version*: https://arxiv.org/abs/2112.11016

Funding This work was supported in part by NSF under Award CCF-1907738.

1 Introduction

Estimating the number of triangles in a massive input graph is a fundamental algorithmic problem that has attracted over two decades of intense research [1, 3, 22, 8, 40, 41, 28, 35, 6, 36, 21, 7, 32, 5, 13, 25, 20]. It is easy to see why. On the one hand, the problem arises in applications where a complex real-world network is naturally modeled as a graph and the number of triangles is a crucial statistic of the network. Such applications are found in many different domains, such as social networks [11, 29, 33], the web graph [4, 44], and biological networks [38]; see [41] for a more detailed discussion of such applications. On the other hand, a triangle is perhaps *the* most basic nontrivial pattern in a graph and as such, triangle counting is a problem with a rich theory and connections to many areas within computer science [1, 15, 18, 34, 14] and combinatorics [31, 17, 27, 37].

In this work, we study the natural generalization of this problem to massive *hypergraphs*. Just as graphs model pairwise interactions between entities in a network, hypergraphs model higher arity interactions. For instance, in an academic collaboration network with researchers being the vertices, it would be natural to model coauthorships on research papers and articles using *hyperedges*, each of which can be incident to more than two vertices. Just as we may use triangle counts to study clustering behaviors in graphs or even in different portions of a single graph, we may analyze higher-order clustering behaviors in the 3-uniform hypergraph H formed by all three-way coauthorships by counting 3-*simplices* in H. A 3-simplex on four vertices $\{u, v, w, x\}$ is the structure formed by the hyperedges $\{uvw, uvx, uwx, vwx\}$: it is the natural 3-dimensional analog of a triangle in an ordinary graph.

1.1 Our Results

We design several algorithms for space-efficiently estimating the number of k-simplices in a k-uniform hypergraph H given as a stream of hyperedges. A *k-simplex* is a complete k-uniform hypergraph on $k+1$ vertices. The special case $k = 2$ is the triangle counting problem which, as noted, is intensely investigated. Indeed, even in this setting of streaming algorithms, triangle counting is highly studied, with new algorithmic techniques being developed as recently as 2021 [20]. There is also a body of work on generalizing these results to the problem of estimating the number of occurrences of patterns (a.k.a. *motifs*) more complicated than triangles, e.g., fixed-size cliques and cycles [30, 26, 36, 5, 24, 42]. Our work adds to this literature by generalizing in a different direction: we generalize the class of *inputs* from graphs to hypergraphs and focus on counting the simplest nontrivial symmetric motifs: k-simplices.

Our algorithms provide optimal space bounds (up to log factors) in certain parameter regimes; we prove this optimality by giving a set of matching lower bounds. In certain other parameter regimes, there remains a gap between our best upper bounds and lower bounds, which immediately provides a goal for future work on this problem. Below are informal statements of our major algorithmic results.

▶ **Theorem 1** (Upper bounds; informal). *Let H be an n-vertex k-uniform hypergraph, presented as a stream of m hyperedges, that is promised to contain T or more k-simplices. Suppose that each hyperedge is contained in at most Δ_E such simplices and each vertex is contained in at most Δ_V of them. Then, there are algorithms for $(1 \pm \varepsilon)$-estimating $T_k(H)$, the number of k-simplices in H, with the following guarantees:*

(1.1) *a 4-pass algorithm using $\widetilde{O}(m^{1+1/k}/T)$ space;*

(1.2) *a 2-pass algorithm using $\widetilde{O}(m/T^{2/(k+1)})$ space, provided $k \geq 3$; and*

(1.3) *a 1-pass algorithm using $\widetilde{O}(m(\Delta_E + \Delta_V^{1-1/k})/T)$ space.*

Each of these algorithms is randomized and fails with probability at most δ. [1]

Formal versions of these results appear as Theorem 14 in Section 3, Theorem 16 in Section 4.2, and in the full paper [9], respectively. Along the way, we also obtain some other algorithmic results that, despite being dominated by the algorithms behind Theorem 1, are technically interesting and possibly useful in future work. These are briefly described at the start of Sections 3 and 4 with full details appearing in the full paper [9]. As we shall see, we take several ideas from triangle-counting algorithms as inspiration, but the "correct" way to extend these ideas to hypergraphs is far from obvious. Indeed, some of the more "obvious" extensions lead to the less-than-best algorithms hinted at above.

[1] Throughout this paper, the notation $\widetilde{O}(\cdot)$ hides factors of $O(\varepsilon^{-2} \log \delta^{-1} \operatorname{polylog} n)$ and, in the context of k-uniform hypergraphs, treats k as a constant. Also, "log" with an unspecified base means $\log_2$.

Next, we informally state our lower bound results. These are not the main technical contributions of this paper, but they play the important role of clarifying where our algorithms are optimal and where there might be room for improvement. As in Theorem 1, we denote the number of k-simplices in H by $T_k(H)$.

▶ **Theorem 2** (Lower bounds; informal). *Let $k, n, m, H, \Delta_E, \Delta_V$ be as above. Suppose an algorithm makes p passes over a stream of hyperedges of H, using at most S bits of working memory, and distinguishes between the cases $T_k(H) = 0$ and $T_k(H) \geq T$ with probability at least $2/3$. Then the following lower bounds apply.*
(2.1) *With $T = 1$ and $p = O(1)$, sublinear space is impossible: we must have $S = \Omega(n^k)$.*
(2.2) *With $p = 1$, we must have $S = \Omega(m\Delta_E/T)$.*
(2.3) *With $p = O(1)$, we need $S = \Omega(\min\{m^{1+1/k}/T, m/T^{1-1/k}\})$, and $S = \Omega(m\Delta_V^{1/k}/T)$.*
[(2.1)] If, instead, the streaming algorithm distinguishes between the cases $T_k(H) < (1-\varepsilon)T$ and $T_k(H) > (1+\varepsilon)T$, then the following lower bound applies.
(2.4) *With $p = O(1)$, we must have $S = \Omega(\varepsilon^{-2})$.*
The full version of the paper [9] further discusses and proves these lower bounds.

1.2 Closely Related Previous Work

To the best of our knowledge, this is the first work to study the simplex counting problem in hypergraphs in the setting just described (the work of [23] on counting general hypergraph patterns is not closely related; see Section 1.3). We now summarize some highlights of previous work on the triangle counting problem (in graphs), with a focus on streaming algorithms, so as to provide context for our contributions.

Suppose an n-vertex m-edge graph G is given as a stream of edges and we wish to estimate the number of triangles, $T_3(G)$. It is not hard to show that, absent any promises on the structure of G, this problem requires $\Omega(n^2)$ space, even with multiple passes, thus precluding a sublinear-space solution. Therefore, all work in this area seeks bounds under a promise that $T_3(G) \geq T$, for some nontrivial threshold T. Intuitively, the larger this threshold, the easier the problem, so we expect the space complexity to decrease. The earliest nontrivial streaming solution [3] reduced triangle counting to a combination of ℓ_0, ℓ_1, and ℓ_2 estimation and achieved $\widetilde{O}((mn/T)^2)$ space by using suitable linear sketches. Almost all algorithms developed since then have instead used some sort of *sampling* to extract a small portion of G, perform some computation on this sample, and then extrapolate to estimate $T_3(G)$.

Over the years, a number of different sampling strategies have been developed, achieving different, sometimes incomparable, guarantees. Here is a whirlwind tour through this landscape of strategies. One could sample an edge uniformly at random (using reservoir sampling), then count common neighbors of its endpoints [22]; or sample an edge uniformly and sample a vertex not incident to it [8]; or sample a subset of edges by independently picking each with a carefully adjusted probability p [28, 6]; or choose a random color for each vertex and collect all monochromatic edges [35]; or sample a subset of vertices at random and collect all edges incident to the sample [6]. One could collect two random subsets of vertices at different sampling rates and further sample edges between the two subsets [25]; or, as in a very recent algorithm, sample a subset of vertices at rate p and further sample edges incident to this sample at rate q, for well-chosen p and q [20]. Notice that in the just-mentioned algorithms, the sampling technique is not actively trying to "grow" a triangle around a sampled edge. Instead, elements of the graph (vertices or edges) are sampled indiscriminately (typically in one streaming pass) and the triangles seen inside the sample are counted in another pass. We shall call such sampling strategies *oblivious*.

Besides the above oblivious sampling strategies, another set of works used what we shall call *targeted sampling* strategies,[2] where information previously stored about the stream guides what gets sampled subsequently. Here is another quick tour through these. One could sample *wedges* (defined as length-2 paths) in the input graph, using a more sophisticated reservoir sampling approach [21]; or sample an edge uniformly and then sample a second edge that touches the first [36]; or sample a vertex with probability proportional to its squared degree, then sample two neighbors of that vertex [32]; or sample an edge uniformly and then sample a neighbor of the lower-degree endpoint of that edge [5].

There are also a handful of algorithms that add further twists on top of the sample-count-extrapolate framework. The algorithm of [7] combines the vertex coloring idea of [35] with ℓ_2 estimation sketches to obtain a solution that can handle *dynamic* graph streams, where each stream update may either insert or delete an edge. The algorithm of [13] combines multiple runs of [35] with a *heavy/light edge partitioning* technique: an edge is deemed "heavy" if it participates in "too many" triangles. A key observation is that the variance of an estimator constructed by oblivious sampling – which needs to be small in order to guarantee good results in small space – can be bounded better if no heavy edges are involved. On the other hand, triangles involving a heavy edge are easier to pick up (because there are many of them!) by randomly sampling vertices at a low rate. Thus, by carefully picking the threshold for heaviness, one can combine an algorithm that counts all-light triangles efficiently with one that counts heavy-edged triangles efficiently for a good overall space bound. In this way, [13] obtains a space bound of $O(\varepsilon^{-2.5} \log \delta^{-1} \operatorname{polylog} n \cdot m/\sqrt{T})$, while [32] provides a tight dependence on ε (i.e., ε^{-2}) by achieving $\widetilde{O}(m/\sqrt{T})$ space.

Separately, the aforementioned targeted sampling strategies of [32] and [5] provide 4-pass algorithms for estimating $T_3(G)$ using space $\widetilde{O}(m^{3/2}/T)$. When T is large enough – specifically, $T = \Omega(m)$ – this space bound is better than the $\widetilde{O}(m/\sqrt{T})$ bound obtained via heavy/light edge partitioning. By picking the better of the two algorithms, one obtains a space bound of $\widetilde{O}(\min\{m^{3/2}/T, m/\sqrt{T}\})$. Both portions of this bound are tight, thanks to lower bounds of $\Omega(m^{3/2}/T)$ and $\Omega(m/\sqrt{T})$ that follow by reducing from the SET-DISJOINTNESS communication problem [13, 5].

The algorithm of [20] is optimal in a different sense: it runs in a single pass and $\widetilde{O}((m/T)(\Delta_E + \sqrt{\Delta_V}))$ space, where Δ_E and Δ_V are as defined in Theorem 1. Each term in this bound is tight; reductions from DISJOINTNESS [5] and the BOOLEAN-HIDDEN-MATCHING problem [25] imply $\Omega(m\sqrt{\Delta_V}/T)$ lower bounds and a reduction from the INDEX communication problem implies an $\Omega(m\Delta_E/T)$ bound for 1-pass algorithms [6]. Note, however, that this result is incomparable to the multi-pass upper bounds noted above. It must be so: a lower bound of $\Omega(m^3/T^2)$ holds for 1-pass algorithms [5].

1.3 Other Related Work

This work is focused on streams that simply *list* the input hypergraph's hyperedges. This is sometimes called the *insert-only* streaming model, in contrast to the *dynamic* or *turnstile* streaming model where the stream describes a sequence of (hyper)edge insertions or deletions. A small subset of works mentioned in Section 1.2 do provide results in a turnstile model. Besides these, there is the recent seminal work [23] that fully settles the complexity of triangle counting in turnstile streams for *constant-degree* graphs. This work also considers the very

[2] To be perfectly honest, the terms "targeted sampling" and "oblivious sampling" do not have precise technical definitions, but we hope the conceptual distinction is helpful to the reader as it was to us.

general problem of counting copies of an arbitrary fixed-size hypergraph motif M inside a large input hypergraph H, again in a turnstile setting. Because of the way their upper bound results depend on the structure of M, they cannot obtain sublinear-space solutions for counting k-simplices without a strong constant-degree assumption on the input H.

A handful of works on triangle counting consider adjacency list streams [8, 28, 32], where the input stream provides all edges incident to each vertex contiguously. This setting can somewhat simplify algorithm design, though the basic framework is still sample-count-extrapolate. We do not consider adjacency list streams in this work.

There are important related algorithms that predate the now-vast literature on streaming algorithms. In particular, [1] gives the current best run-time for exact triangle counting in the RAM model and [10] gives time-efficient algorithms for listing all triangles (and more general motifs). A version of the heavy/light partitioning idea appears in these early works.

More recently, a handful of works [15, 2, 16] have designed *sublinear-time* algorithms for approximately counting triangles and other motifs given query access to a large input graph. Triangle detection, listing, and counting have connections to other important problems in the area of fine-grained complexity [18, 43]. Triangle counting has also been studied in distributed, parallel, and high-performance computing models [40, 35, 39, 19].

2 Preliminaries

We now define our key terminology, set up notation, and establish some basic facts that we shall refer to in the algorithms and analyses to come.

▶ **Definition 3** (Hypergraph, degrees, neighborhoods). *A* hypergraph *is a pair $H = (V, E)$ where V is a nonempty finite set of* vertices *and $E \subseteq 2^V$ is a set of* hyperedges. *In certain contexts, this structure is instead called a* set system *over V. If $|e| = k \geq 1$ for all $e \in E$, then H is said to be* k-uniform. *For simplicity, we shall often shorten "k-uniform hypergraph" to* k-graph *and "hyperedge" to "edge."*

For each $S \subseteq V$ with $|S| < k$, the joint degree *or* codegree $\operatorname{codeg}(S)$ *of S is the number of edges that strictly extend S. The* neighborhood $\mathrm{N}(S)$ *of S is the set of non-S vertices that share an edge with S. Formally,*

$$\operatorname{codeg}(S) := |\{e \in E : e \supsetneq S\}| ; \tag{1}$$

$$\mathrm{N}(S) := \{v \in V : v \notin S \text{ and } \exists\, e \in E \text{ such that } e \supseteq S \cup \{v\}\} . \tag{2}$$

For a singleton set $S = \{u\}$, we define $\deg(u) := \operatorname{codeg}(\{u\})$ *and* $\mathrm{N}(u) := \mathrm{N}(\{u\})$. *Further, given $S \subseteq V$, we define the* S-relative degrees *of vertices $x \in V \setminus S$ by*

$$\deg(x \mid S) := \operatorname{codeg}(S \cup \{x\}) . \tag{3}$$

Note that this is meaningful only when $|S| \leq k - 2$. Note, also, that $\deg(x) = \deg(x \mid \varnothing)$.

Throughout the paper, hypergraphs will be k-uniform unless qualified otherwise. We shall consistently use the notation $H = (V, E)$ for a generic k-graph and define $n := |V|$ and $m := |E|$. We shall assume that each vertex $v \in V$ has a unique ID, denoted $\mathrm{ID}(v)$, which is an integer in the range $[n] := \{1, \ldots, n\}$.

In general, there isn't an equation relating $|\mathrm{N}(S)|$ to $\operatorname{codeg}(S)$. However, all k-graphs satisfy the following useful lemmas.

▶ **Lemma 4.** *For $S \subseteq V$ with $|S| < k$,* $\operatorname{codeg}(S) \leq |\mathrm{N}(S)| \leq (k - |S|) \operatorname{codeg}(S)$.

Proof. Each edge that extends S adds ≥ 1 and $\leq k - |S|$ new neighbors to the sum. ◀

▶ **Lemma 5.** *For $1 \leq r < k$, we have $\sum_{S \subseteq V\colon |S|=r} \operatorname{codeg}(S) = \binom{k}{r} m = O(m)$.*

Proof. The sum counts each edge exactly $\binom{k}{r}$ times. ◀

▶ **Definition 6** (Simplices)**.** *A k-*simplex *is a k-graph on $k+1$ vertices such that all possible k-sized edges are present. If $H = (V, E)$ is a k-graph and $X \subseteq V$ with $|X| = k+1$, we say that H has a simplex at X if the induced subhypergraph $H[X] := (X, \{e \in E : e \subseteq X\})$ is a simplex. Abusing notation, we also use X to denote this simplex.*

We use $\mathcal{T}_k(H)$ to denote the set of all k-simplices in H and $T_k(H) := |\mathcal{T}_k(H)|$ to denote the number of such simplices. We define $\Delta_E = \Delta_E(H)$ and $\Delta_V = \Delta_V(H)$ as follows:

$$\Delta_E := \max_{e \in E} |\{X \in \mathcal{T}_k(H) : X \text{ contains edge } e\}| \,; \tag{4}$$

$$\Delta_V := \max_{v \in V} |\{X \in \mathcal{T}_k(H) : X \text{ contains vertex } v\}| \,. \tag{5}$$

Drawing inspiration from the *link* operation for abstract and simplicial complexes in topology, we define a couple of operations that derive $(k-1)$-graphs from k-graphs.

▶ **Definition 7** (Neighborhood and shadow hypergraphs)**.** *Let $k \geq 3$ and let $H = (V, E)$ be a k-graph. For each $u \in V$, the* neighborhood hypergraph of H at u *is the $(k-1)$-graph $H \downarrow u = (\mathrm{N}(u), E_u)$, where*

$$E_u := \{\{x_1, \ldots, x_{k-1}\} : \{u, x_1, \ldots, x_{k-1}\} \in E\} = \{e \smallsetminus u : e \in E \text{ and } e \ni u\} \,.$$

The shadow hypergraph *of H is the $(k-1)$-graph $H' = (V', E') = \left(\bigcup_{u \in V} V'_u, \bigcup_{u \in V} E'_u\right)$, where*

$$V'_u := \left\{x^{\langle u \rangle} : x \in V\right\} \text{ is a copy of } V \text{ ``flavored'' with } u, \text{ and}$$

$$E'_u := \left\{\left\{x_1^{\langle u \rangle}, \ldots, x_{k-1}^{\langle u \rangle}\right\} : \{u, x_1, \ldots, x_{k-1}\} \in E \text{ and } \mathrm{ID}(u) \leq \mathrm{ID}(x_i) \;\forall\, i \in [k-1]\right\} .$$

Note that E'_u is subtly different from E_u in that it is induced by hyperedges incident on u in which u is the minimum-ID vertex. Observe that, as a result, $|E'| = |E|$.

We use neighborhood hypergraphs to analyze our first major algorithm, in Section 3. We use shadow hypergraphs in a crucial way to obtain our second major algorithm, in Section 4.2.

In the literature on triangle counting in graphs, the structure formed by two edges sharing a common vertex is called a *wedge*. It will be useful to define an analog of the notion for hypergraphs.

▶ **Definition 8** (Hyperwedge)**.** *A k-*hyperwedge *is the hypergraph obtained by deleting one hyperedge from a k-simplex. Thus, if its vertex set is X, then $|X| = k+1$ and it has k hyperedges with exactly one common vertex, which we call the* apex *of the hyperwedge. The only k-sized subset of X that is* not *present in the hyperwedge is called the* base *of the hyperwedge. Adding this base as a new hyperedge produces a k-simplex.*

Fix a k-graph H and a k-simplex X in H. Observe that X contains $k+1$ distinct hyperwedges; each such hyperwedge W corresponds to a $(k-1)$-simplex in the neighborhood hypergraph of H at the apex of W. On the other hand, X corresponds to exactly one $(k-1)$-simplex in the shadow hypergraph H', namely the simplex at $\{u_1^{\langle z \rangle}, \ldots, u_k^{\langle z \rangle}\}$, where z is the minimum-ID vertex in X and $\{u_1, \ldots, u_k\} = X \smallsetminus \{z\}$.

Our algorithm analyses often use the following standard boosting lemma from the theory of randomized algorithms. A random variable $\widetilde{Q}$ is said to be an (ε, δ)-*estimate* of a statistic Q if $\Pr[\widetilde{Q} \in (1 \pm \varepsilon) Q] \geq 1 - \delta$.

▶ **Lemma 9** (Median-of-Means improvement). *If $\hat{Q}$ is an unbiased estimator of some statistic, then one can obtain an (ε, δ)-estimate of that statistic by suitably combining*

$$K := \frac{C}{\varepsilon^2} \ln \frac{2}{\delta} \cdot \frac{\operatorname{Var}[\hat{Q}]}{\mathbb{E}[\hat{Q}]^2}$$

independent samples of $\hat{Q}$, where C is a universal constant. ⌟

3 An Optimal Algorithm Based on Targeted Sampling

In this section, we establish Subresult (1.1) of Theorem 1 by giving a formal description of the relevant algorithm (as Algorithm 1) followed by its analysis. This algorithm is based on "targeted sampling," as outlined in Section 1.2. It runs in $\widetilde{O}(m^{1+1/k}/T)$ space, which is optimal in view of Subresult (2.3) in Theorem 2. It is the method of choice for the "abundant" case, when T is large enough: specifically, in view of the guarantees of Algorithm 2 to follow (see Theorem 16), "large enough" should be defined as $T \geq m^{(k+1)/(k^2-k)}$.

The full version of the paper [9] additionally gives a more straightforward targeted-sampling algorithm, achieving a suboptimal space bound of $\widetilde{O}(m^{2-1/k})$. Although suboptimal, that algorithm's analysis contains novel ideas about the structure of hypergraphs that might be useful in subsequent research, such as the notion of "hyperabroricity."

3.1 The Algorithm

The basic setup is as in the algorithm of [5] for counting odd cycles in graphs (and an analogous algorithm of [32]). We use one pass to pick an edge $e \in E$ uniformly at random and, in subsequent passes, use further sampling to estimate the number of suitable apex vertices z at which there is a hyperwedge with base e, which would complete a simplex together with e. In order to control the variance of the resulting unbiased estimator, we want the vertices of a detected simplex $e \cup \{z\}$ to respect a certain ordering "by degree."

We let the sampled edge e determine the ordering, for which we use *relative* degrees, relative to certain subsets of e. Moreover, this ordering – which we call the e-relative ordering – is only partial, as we never need to compare two vertices that both lie outside e. Let $c_i(e)$ be the ith vertex of e according to this ordering that we shall soon define. In our algorithm, we shall look for a suitable apex z only in the neighborhood $\mathrm{N}(\{c_1(e), \ldots, c_{k-1}(e)\})$. Furthermore, we shall require that z have a larger degree than $c_1(e)$, a larger $\{c_1(e)\}$-relative degree than $c_2(e)$, a larger $\{c_1(e), c_2(e)\}$-relative degree than $c_3(e)$, and so on. As we shall see, the analysis of the resulting algorithm uses the recursive structure of k-graphs and k-simplices through the notion of neighborhood hypergraphs (Definition 7).

These next two, somewhat elaborate, definitions formalize the above ideas.

▶ **Definition 10** (e-relative ordering). *Fix a hyperedge $e \in E$. The e-relative ordering, $\prec_e$, is a partial order on V defined as follows. Set $S_0(e) = \varnothing$ and define the following, iteratively, for i running from 1 to k.*

> *Let $c_i(e)$ be the vertex in $e \smallsetminus S_{i-1}(e)$ that minimizes $\deg(c_i(e) \mid S_{i-1}(e))$, with ties resolved in favor of the vertex with smallest ID. Set $S_i(e) := S_{i-1}(e) \cup \{c_i(e)\} = \{c_1(e), \ldots, c_i(e)\}$.*

Then declare $c_1(e) \prec_e \cdots \prec_e c_k(e)$. *Further, for each* $z \in V \smallsetminus e$, *declare* $c_k(e) \prec_e z$ *if the following two things hold: (a) for* $1 \le i \le k-1$, *we have* $\deg(c_i(e) \mid S_{i-1}(e)) \le \deg(z \mid S_{i-1}(e))$, *and (b) we also have* $\deg(c_k(e) \mid S_{k-2}(e)) \le \deg(z \mid S_{k-2}(e))$. *Ties in degree comparisons are resolved by declaring the vertex with smaller ID to be smaller.*[3]

▶ **Definition 11** (Simplex labeling). *Suppose that* H *has a simplex at* X, *where* $X \subseteq V$. *Number the vertices in* X *as* $u_1, \ldots, u_{k+1}$ *iteratively, as follows. For* i *from* 1 *to* $k-1$, *let* u_i *be the vertex in* X *that minimizes* $\deg(u_i \mid \{u_1, \ldots, u_{i-1}\})$. *Let* u_k *be the vertex among* $\{u_k, u_{k+1}\}$ *that minimizes* $\deg(u_k \mid \{u_1, \ldots, u_{k-2}\})$. *This automatically determines* u_{k+1}. *Based on this, define the* label *of the simplex at* X *to be* $(e(X), z(X))$, *where* $e(X) := \{u_1, \ldots, u_k\}$ *and* $z(X) := u_{k+1}$.

Notice that if a simplex is labeled (e, z), then the e-relative ordering satisfies $\max_{\prec_e}(e) \prec_e z$. Conversely, if there is a simplex at X and an edge e of this simplex such that $X = e \cup \{z\}$ and $\max_{\prec_e}(e) \prec_e z$, then the label of this simplex is (e, z).

For each $e \in E$, let τ_e denote the number of simplices that have a label of the form $(e, \cdot)$. Thanks to the uniqueness of labels, we immediately have

$$\sum_{e \in E} \tau_e = T_k(H)\,. \tag{6}$$

Our algorithm is described in Algorithm 1. Our analysis will make use of the following two combinatorial lemmas at key moments. The proofs of these lemmas can be found in the full paper [9].

▶ **Lemma 12.** *For every edge* $e \in E$, *we have* $\tau_e \le k m^{1/k}$.

▶ **Lemma 13.** *If* H *is* k*-uniform and* $k \ge 3$, *then* $\sum_{e \in E} \operatorname{codeg}(S_{k-1}(e)) = O\left(m^{1+1/k}\right)$.

3.2 Analysis of the Algorithm

We begin by proving that the estimator Y computed by Algorithm 1 is unbiased. Let $\mathcal{E}_e$ denote the event that the edge sampled in pass 1 is e. For each $j \in [R]$, the vertex x_j is picked uniformly at random from $\mathrm{N}(S_{k-1}(e))$ and exactly τ_e choices cause Z_j to be set to the nonzero value $\operatorname{codeg}(S_{k-1}(e))$. Therefore,

$$\mathbb{E}[Z_j \mid \mathcal{E}_e] = \frac{\tau_e}{|\mathrm{N}(S_{k-1}(e))|} \cdot \operatorname{codeg}(S_{k-1}(e)) = \tau_e \tag{7}$$

because $\operatorname{codeg}(S_{k-1}(e)) = |N(S_{k-1}(e))|$, by Lemma 4. By the law of total expectation and Equation (6),

$$\mathbb{E}[Y] = \frac{m}{R} \sum_{j=1}^{R} \sum_{e \in E} \frac{1}{m} \mathbb{E}[Z_j \mid \mathcal{E}_e] = \frac{1}{R} \sum_{j=1}^{R} \sum_{e \in E} \tau_e = T_k(H)\,. \tag{8}$$

Next, we bound the variance of the estimator. Proceeding along similar lines, we have $\mathbb{E}[Z_j^2 \mid \mathcal{E}_e] = \operatorname{codeg}(S_{k-1}(e)) \cdot \tau_e$ and $\mathbb{E}[Z_{j_1} Z_{j_2} \mid \mathcal{E}_e] = \tau_e^2$ for all $j_1 \ne j_2$, because Z_{j_1} and Z_{j_2} are independent conditioned on $\mathcal{E}_e$. So,

[3] This is how ties are resolved from now on, even when not mentioned.

Algorithm 1 Counting k-simplices in k-uniform hypergraph streams: the “abundant” case.

```
procedure k-Simplex-Count-Abundant($\sigma$ : stream of edges of $k$-graph $H = (V, E)$)
    pass 1:                                                                  ▷ $O(1)$ space
        pick edge $e = \{u_1, u_2, \ldots, u_k\} \in E$ u.a.r. using reservoir sampling
    pass 2:                                                         ▷ $O(2^k) = O(1)$ space
        compute $\operatorname{codeg}(S)$ for all non-trivial $S \subset e$
    pass 3:                                                                  ▷ $O(R)$ space
        re-arrange (if needed) $u_1, \ldots, u_k$ so that $u_i = c_i(e)$ for $i \in [k]$, using co-degrees from pass 2
        $R \leftarrow \lceil \operatorname{codeg}(S_{k-1}(e)) \cdot m^{-1/k} \rceil$      ▷ note that $\operatorname{codeg}(S_{k-1}(e)) = |\mathrm{N}(S_{k-1}(e))|$
        for $j \leftarrow 1$ to $R$, in parallel, do
            $Z_j \leftarrow 0$
            pick vertex $x_j \in \mathrm{N}(S_{k-1}(e))$ u.a.r. using reservoir sampling from relevant substream of $\sigma$
    pass 4:                                                         ▷ $O(kR) = O(R)$ space
        compute the relative degrees $\deg(x_j \mid S_{i-1}(e))$ for all $i \in [k-1]$ and $j \in [R]$
        for $j \leftarrow 1$ to $R$, in parallel, do
            if $e \cup \{x_j\}$ determines a simplex with label $(e, x_j)$ then
                $Z_j \leftarrow \operatorname{codeg}(S_{k-1}(e))$
    return $Y \leftarrow (m/R) \sum_{j=1}^{R} Z_j$                   ▷ Average out the trials and scale
```

$$\mathbb{E}[Y^2 \mid \mathcal{E}_e] = \mathbb{E}\left[\left(\frac{m}{R}\sum_{j=1}^{R} Z_j\right)^2 \,\middle|\, \mathcal{E}_e\right] = \frac{m^2}{R^2}\sum_{j=1}^{R}\mathbb{E}[Z_j^2 \mid \mathcal{E}_e] + \frac{m^2}{R^2}\sum_{j_1 \neq j_2}\mathbb{E}[Z_{j_1} Z_{j_2} \mid \mathcal{E}_e]$$
$$\leq \frac{m^2 \cdot \operatorname{codeg}(S_{k-1}(e)) \cdot \tau_e}{R} + \frac{m^2 R(R-1)\tau_e^2}{R^2} \leq m^{2+1/k}\tau_e + m^2\tau_e^2\,,$$

since $\operatorname{codeg}(S_{k-1}(e)) \leq m^{1/k} R$ by the definition of R. Using $\sum_{e \in E} \tau_e = T_k(H)$, we get

$$\operatorname{Var}[Y] \leq \mathbb{E}[Y^2] = \sum_{e \in E}\frac{1}{m}\mathbb{E}[Y^2 \mid \mathcal{E}_e] \leq m^{1+1/k}\sum_{e \in E}\tau_e + m\sum_{e \in E}\tau_e^2$$
$$\leq m^{1+1/k} T_k(H) + m \cdot (\max_{e \in E}\tau_e) \cdot \sum_{e \in E}\tau_e = O\left(m^{1+1/k} T_k(H)\right)\,,$$

where we invoked Lemma 12 and used Equation (6) twice.

Having bounded the variance, we may apply the median-of-means technique (Lemma 9) to the basic estimator of Algorithm 1 to obtain a final algorithm that provides an (ε, δ)-estimate. As noted in the comments within Algorithm 1, the basic estimator uses $O(R)$ space, where R is a random variable. Therefore, the final algorithm, which still uses four passes, has expected space complexity $\widetilde{O}(\mathbb{E}[R] \cdot \operatorname{Var}[Y]/\mathbb{E}[Y]^2) = \mathbb{E}[R] \cdot \widetilde{O}(m^{1+1/k}/T_k(H)) = \mathbb{E}[R] \cdot \widetilde{O}(m^{1+1/k}/T)$, thanks to the promise that $T_k(H) \geq T$.

Finally, we need to bound R in expectation. We find that

$$\mathbb{E}[R] = \frac{1}{m}\sum_{e \in E}\mathbb{E}[R \mid \mathcal{E}_e] = \frac{1}{m}\sum_{e \in E}\left\lceil\frac{\operatorname{codeg}(S_{k-1}(e))}{m^{1/k}}\right\rceil \leq 1 + m^{-1-1/k}\sum_{e \in E}\operatorname{codeg}(S_{k-1}(e))\,.$$

Invoking Lemma 13, we conclude that $\mathbb{E}[R] = O(1)$, giving our first main algorithmic result.

▶ **Theorem 14.** *There is a 4-pass algorithm that reads a stream of m hyperedges specifying a k-uniform hypergraph H and produces an (ε, δ)-estimate of $T_k(H)$; under the promise that $T_k(H) \geq T$, the algorithm runs in expected space $O(\varepsilon^{-2} \log \delta^{-1} \log n \cdot m^{1+1/k}/T)$.* ⌟

We conclude with two remarks about this algorithm. First, the space bound can easily be made worst-case, by applying the following modification to the median-of-means boosting process. Notice that in each instance of the basic estimator, at the end of pass 2, we know the value of R and thus the actual space that this instance would use. For each batch of estimators for which we compute a mean, we determine whether this batch would use more than A times its expected space bound, for some large constant A. If so, we abort this batch and don't use it in the median computation. By Markov's inequality, the probability that we abort a batch is at most $1/A$, which does not affect the rest of the standard analysis of the quality of the final median-of-means estimator.

Second, while we have not dwelt on *time* complexity in our exposition, it is not hard to convince oneself that the algorithm processes each incoming edge in $\widetilde{O}(R)$ time, which is $\widetilde{O}(1)$ in expectation.

4 Algorithms Based on Oblivious Sampling

We now turn to our second family of algorithms, which use "oblivious sampling," as outlined in Section 1.2. To recap, whereas the algorithms in the previous section used an initially-sampled edge to guide their subsequent sampling, an oblivious sampling strategy decides whether or not to store an edge based only coin tosses specific to that edge and perhaps its constituent vertices (and not on any edges in store). The main result of this section is an $\widetilde{O}(m/T^{2/(k+1)})$-space algorithm, given as Algorithm 2, which is the method of choice for the "meager" case, when T is not too large (specifically, $T < m^{(k+1)/(k^2-k)}$). Simpler versions of our ideas give algorithms with worse space guarantees: namely, $\widetilde{O}(m/T^{1/k})$ and $\widetilde{O}(m/T^{2/(k+2)})$. The full version of the paper [9] gives the details.

The full paper additionally describes a 1-pass algorithm – also based on oblivious sampling – whose space guarantee depends on additional structural parameters of the input hypergraph H, proving Subresult (1.3) of Theorem 1.

4.1 A Unifying Framework for Oblivious Sampling Strategies

We find it useful to set up a framework for analyzing algorithms based on oblivious sampling and view our eventual algorithm as an instantiation of the framework. Further, this framework unifies a number of approaches seen in previous work on triangle counting [32, 20, 25, 6].

Suppose that we are trying to estimate the number, $T(H)$, of copies of a target substructure (motif) Ξ in a streamed hypergraph H. Suppose we do this by collecting certain edges into a random sample Q, using some random process parameterized by a quantity p. The sample Q determines which of the $T(H)$ copies of Ξ get "detected" by the rest of the logic of the algorithm. Let $\Xi_1, \ldots, \Xi_{T(H)}$ be the copies of Ξ in H (under some arbitrary enumeration scheme) and let Λ_i be the indicator random variable for the event that Ξ_i is detected.

In our framework, we will want to identify parameters α, β, and γ such that the following conditions hold for all $i, j \in [T(H)]$ with $i \neq j$ and all $e \in E$:

$$\Pr[\Lambda_i = 1] = p^\alpha \,; \quad \Pr[\Lambda_i = \Lambda_j = 1] \leq p^\beta \,; \quad \Pr[e \in Q] = p^\gamma \,. \tag{9}$$

In instantiations of the framework, we will of course have $\beta > \alpha$ and further, we will have $\beta < 2\alpha$, which is natural because for certain pairs i, j, the variables Λ_i and Λ_j will be positively correlated. We shall further require that our detection process satisfy the *separation property*, where $\Xi \cap \Xi'$ denotes the set of *edges* common to Ξ and Ξ' and the notation "$X \perp Y$" means that random variables X and Y are independent.

$$\textsc{Separation Property: } \text{If } \Xi_i \cap \Xi_j = \varnothing, \text{ then } \Lambda_i \perp \Lambda_j\,. \tag{10}$$

The logic of the algorithm will count the number of detected copies of Ξ in an accumulator $A := \sum_{i=1}^{T(H)} \Lambda_i$. Defining the estimator $\widetilde{T} := A/p^\alpha$, we see that $\mathbb{E}[\widetilde{T}] = \sum_{i=1}^{T(H)} p^{-\alpha}\mathbb{E}[\Lambda_i] = T(H)$, which makes it unbiased. To establish concentration, we estimate

$$\begin{aligned}\operatorname{Var}[A] &= \operatorname{Var}\sum_{i=1}^{T(H)} \Lambda_i = \sum_{i=1}^{T(H)} \operatorname{Var}[\Lambda_i] + \sum_{i \neq j} \operatorname{Cov}[\Lambda_i, \Lambda_j] \le \sum_{i=1}^{T(H)} \mathbb{E}[\Lambda_i^2] + \sum_{i \neq j} \operatorname{Cov}[\Lambda_i, \Lambda_j] \\ &\le T(H)p^\alpha + \sum_{\Lambda_i \not\perp \Lambda_j} \mathbb{E}[\Lambda_i \Lambda_j] \le T(H)p^\alpha + \sum_{\Lambda_i \not\perp \Lambda_j} p^\beta = T(H)p^\alpha + p^\beta \cdot \underbrace{\sum_{e \in E} \sum_{\Xi_i \cap \Xi_j = \{e\}} 1}_{s_{\text{cor}}}\,, \end{aligned} \tag{11}$$

where the penultimate step uses (9) and the last step holds because of the separation property and the reasonable structural assumption (true of simplices) that two distinct copies of the motif Ξ can intersect in at most one edge.

The term s_{cor} captures the amount of correlation in the sampling process. Let us now specialize the discussion to the problem at hand, where the motif Ξ is a k-simplex and so, $T(H) = T_k(H)$. Let M_e be the number of simplices that share hyperedge e (note that this is subtly different from τ_e in Section 3). Recalling the definition of Δ_E in Equation (4), we see that $\max_{e \in E} M_e = \Delta_E$. We can therefore upper bound s_{cor} as follows:

$$s_{\text{cor}} \le \sum_{e \in E} \binom{M_e}{2} \le \sum_{e \in E} \frac{M_e^2}{2} \le \frac{\Delta_E}{2} \sum_{e \in E} M_e = \frac{(k+1)\Delta_E T_k(H)}{2}\,. \tag{12}$$

Plugging this bound into (11) and applying Chebyshev's inequality gives the following "error probability" bound, where $B > 0$ will be chosen later.

$$P_{\text{err}}^{(B)}\big[\,\widetilde{T}\,\big] := \Pr\left[|\widetilde{T} - T(H)| \ge \varepsilon B\right] \le \frac{\operatorname{Var}[A]}{p^{2\alpha}\varepsilon^2 B^2} \le \frac{T_k(H)}{\varepsilon^2 p^\alpha B^2} + \frac{(k+1)\Delta_E T_k(H)}{2\varepsilon^2 p^{2\alpha-\beta} B^2}\,. \tag{13}$$

Heavy/Light Edge Partitioning via an Oracle. Using the above estimator $\widetilde{T}$ directly on the input hypergraph H does not give a good bound in general, because the quantity Δ_E appearing in Equation (13) could be as large as $T_k(H)$, making the upper bound on P_{err} useless. One remedy is to apply the technique so far to a subgraph of H where Δ_E is *guaranteed* to be a fractional power of $T_k(H)$ and deal separately with simplices that don't get counted in such a subgraph. Such a subgraph can be obtained by taking only the "light" edges in H, where an edge e is considered to be light if $M_e = O(T_k(H)^\theta)$, for some parameter $\gamma < 1$ that we will soon choose. We remark that this kind of heavy/light partitioning is a standard technique in this area: it occurs in the triangle counting algorithms of [12, 32] and in fact in the much earlier triangle listing algorithm of [10].

More precisely, we create an *oracle* to label each edge as either HEAVY or LIGHT. Thus:

$$T_k(H) = T_k^L(H) + T_k^H(H)\,, \tag{14}$$

where $T_k^L(H)$ is the number of simplices containing only light edges, and $T_k^H(H)$ is the number of simplices containing at least one heavy edge. The key insight in the partitioning technique is that we can estimate the two terms in Equation (14) separately, using the estimator $\widetilde{T}$ described above for $T_k^L(H)$ and a different estimator for $T_k^H(H)$ that we shall soon describe. In what follows, we allow an additive error of $\pm\varepsilon T_k(H)$ in the estimate of each term, leading to a multiplicative error of $1 \pm 2\varepsilon$ in the estimation of $T_k(H)$.

The oracle is implemented by running the following one-pass randomized streaming algorithm implementing a function $\mathrm{orcl}\colon E \to \{\textsc{heavy}, \textsc{light}\}$ to label the edges of H. Form a random subset $Z \subseteq V$ by picking each vertex $v \in V$, independently, with probability

$$q := \xi\varepsilon^{-2} \ln n \cdot T^{-\theta}\,, \tag{15}$$

where T is the promised lower bound on $T_k(H)$ given to the algorithm, and $\xi \geq 12k(k+1)$ is a constant. Then, in one pass over the input stream, collect all hyperedges e containing at least one vertex from Z. Let S denote the random sample of edges thus collected. Note that this is distinct from (and independent of) the sample Q collected by the oblivious sampling scheme discussed above. Now, for each hyperedge $e = \{u_1, \ldots, u_k\} \in E$, let $\widetilde{M}_e := |\{z \in Z : e \cup \{z\} \in \mathcal{T}_k(H)\}|$ be the number of simplices that e completes with respect to Z. Then we define:

$$\mathrm{orcl}(e) = \begin{cases} \textsc{heavy}\,, & \text{if } \widetilde{M}_e < qT^\theta\,, \\ \textsc{light}\,, & \text{otherwise.} \end{cases} \tag{16}$$

The next lemma (whose proof, via Chernoff bounds, we omit) says that the oracle's predictions correspond, with high probability, to our intuitive definition of heavy/light edges, up to a factor of 2.

▶ **Lemma 15.** *W.h.p., the oracle given by* (16) *is "correct," meaning that it satisfies*

$$\forall\, e \in E: \ \mathrm{orcl}(e) = \textit{LIGHT} \ \Rightarrow \ M_e < 2T^\theta \ \wedge \ \mathrm{orcl}(e) = \textit{HEAVY} \ \Rightarrow \ M_e > \tfrac{1}{2}T^\theta\,.$$

The space required to implement the oracle is that required to store the sample S. Since

$$\mathbb{E}[|S|] = O\left(\sum_{v \in V} q \deg(v)\right) = O(qm)\,, \tag{17}$$

the expected space is $O(qm \log n) = O(\varepsilon^{-2} \log^2 n \cdot T^{-\theta} m) = \widetilde{O}(m/T^\theta)$, by Equation (15).

Counting the Light and Heavy Simplices. We can now describe our overall algorithm. We use two streaming passes. In the first pass, we collect the samples S and Q. At the end of the pass, we have the information necessary to prepare the heavy/light labeling oracle and using it, we remove all heavy edges from Q. In the second pass, for each edge e encountered, we use the oracle to classify it. If e is light, we feed it into the detection logic for the oblivious sampling scheme (which uses Q). If e is heavy, we use S to estimate M_e, the number of simplices containing e; this requires a little care, as we explain below.

In analyzing the algorithm, we start by assuming that the oracle is correct (which happens w.h.p., by Lemma 15). Let $\widetilde{T}^L$ be the estimator for $T_k^L(H)$ produced by the oblivious sampling scheme. We analyze its error probability using Equation (13), noting that $\Delta_E \leq 2T^\theta$ in the light-edges subgraph and choosing $B = T_k(H)$. This leads to

$$
\begin{aligned}
P_{\mathrm{err}}[\widetilde{T}^L] &:= \Pr\left[|\widetilde{T}^L - T_k^L(H)| \geq \varepsilon T_k(H)\right] \\
&\leq \frac{T_k^L(H)}{\varepsilon^2 p^\alpha T_k(H)^2} + \frac{(k+1)\cdot 2T^\theta \cdot T_k^L(H)}{2\varepsilon^2 p^{2\alpha-\beta} T_k(H)^2} \leq \frac{1}{\varepsilon^2 p^\alpha T} + \frac{k+1}{\varepsilon^2 p^{2\alpha-\beta} T^{1-\theta}}\,, \qquad (18)
\end{aligned}
$$

where we used the inequalities $T_k^L(H) \leq T_k(H)$ and $T_k(H) \geq T$. This bound on the error probability will inform our choices of parameters p, α, β, and θ, below.

Finally, we estimate $T_k^H(H)$. Define a simplex in $\mathcal{T}_k(H)$ to be *i-heavy* if it has exactly i heavy edges. For each edge e and $1 \leq i \leq k+1$, let M_e^i be the number of i-heavy simplices containing e. Notice that

$$
\sum_{e \text{ heavy}} M_e^i = i \cdot |\{\Xi \in \mathcal{T}_k(H) : \Xi \text{ is } i\text{-heavy}\}|\,, \quad \text{whence } T_k^H(H) = \sum_{e \text{ heavy}} \sum_{i=1}^{k+1} \frac{M_e^i}{i}\,. \qquad (19)
$$

During our second streaming pass, we compute unbiased estimators for each M_e^i as follows. Recall that S is the set of all edges incident to some vertex in the set Z of sampled vertices. Define $\widetilde{M_e^i}$ to be the number of vertices in Z that combine with e to form an i-heavy simplex; we consult the oracle to test whether a simplex in question is indeed i-heavy. Notice that $\widetilde{M_e^i} \sim \mathrm{Bin}(M_e^i, q)$. Therefore, by Equation (19),

$$
\widetilde{T}^H := \frac{1}{q} \sum_{e \text{ heavy}} \sum_{i=1}^{k+1} \frac{\widetilde{M_e^i}}{i} \qquad (20)
$$

is an unbiased estimator for $T_k^H(H)$. To establish concentration, consider the inner sum corresponding to any particular edge e. The sum can be rewritten as a sum of M_e mutually independent indicator variables scaled by factors in $\{1, 1/2, \ldots, 1/(k+1)\}$. Since the oracle is correct, the heaviness of e implies that $M_e \geq \frac{1}{2}T^\theta$ and then a Chernoff bound gives

$$
\Pr\left[\sum_{i=1}^{k+1} \frac{\widetilde{M_e^i}}{i} \notin (1 \pm \varepsilon) q \sum_{i=1}^{k+1} \frac{M_e^i}{i}\right] \leq 2\exp\left(-\frac{1}{3}\varepsilon^2 q \cdot \frac{1}{2}T^\theta \cdot \frac{1}{k+1}\right) \leq n^{-2k}\,,
$$

where the final step uses $\xi \geq 12k(k+1)$. By a union bound over the $\leq n^k$ heavy edges,

$$
\begin{aligned}
P_{\mathrm{err}}[\widetilde{T}^H] &:= \Pr\left[|\widetilde{T}^H - T_k^H(H)| \geq \varepsilon T_k(H)\right] \leq \Pr\left[\widetilde{T}^H \notin (1 \pm \varepsilon) T_k^H(H)\right] \\
&\leq \Pr\left[\bigvee_{e \text{ heavy}} \left\{\sum_{i=1}^{k+1} \frac{\widetilde{M_e^i}}{i} \notin (1 \pm \varepsilon) q \sum_{i=1}^{k+1} \frac{M_e^i}{i}\right\}\right] \leq n^{-O(1)}\,. \qquad (21)
\end{aligned}
$$

Determining the Parameters and Establishing a Space Bound. We have now described the overall logic of a generic algorithm in our framework. To instantiate it, we must plug in a specific oblivious sampling scheme, whose logic will determine the parameters α, β, and γ as given by Equation (9). Now we choose p to ensure that $P_{\mathrm{err}}[\widetilde{T}^L]$ is less than a small constant. Thanks to Equation (18), it suffices to ensure that

$$
\frac{1}{\varepsilon^2 p^\alpha T} + \frac{k+1}{\varepsilon^2 p^{2\alpha-\beta} T^{1-\theta}} = o(1)\,.
$$

Setting $p = (\log n/(\varepsilon^2 T))^{1/\alpha}$ reduces the first term to $1/\log n$. To make the second term small as well, we set the threshold parameter θ used by the heavy/light partitioning oracle to $\theta := \beta/\alpha - 1$; as we remarked after Equation (9), we will have $\alpha < \beta < 2\alpha$, which implies $0 < \theta < 1$. The second term now reduces to

$$\frac{(k+1)p^{\beta-\alpha}T^{\theta}}{\log n} = \frac{k+1}{(\log n)^{1-\theta}\varepsilon^{2\theta}} = o(1)\,,$$

for n sufficiently large. We therefore obtain $P_{\text{err}}[\widetilde{T}^L] = o(1)$ and, putting this together with Equation (21) and Lemma 15, we conclude that the overall algorithm computes a $(1 \pm 2\varepsilon)$-approximation to $T_k(H)$ with probability at least $1 - o(1)$.

With these parameters now set, the space complexity analysis is straightforward. The space usage is dominating by the storing of the samples S and Q. By Equation (17) and Equation (15), we have $\mathbb{E}[|S|] = \widetilde{O}(m/T^{\theta})$ and by Equation (9), we have

$$\mathbb{E}[|Q|] = p^{\gamma}m = \left(\frac{\log n}{\varepsilon^2}\right)^{\gamma/\alpha} \frac{m}{T^{\gamma/\alpha}}\,.$$

It is natural that the probability of detecting a particular simplex is at most the probability of storing one particular edge, so $\gamma \le \alpha$. Therefore, $\mathbb{E}[|Q|] \le \varepsilon^{-2}\log n \cdot m/T^{\gamma/\alpha}$. Altogether, since storing a single edge uses $O(\log n)$ bits, the total space usage is $\widetilde{O}(m/T^{\lambda})$, where

$$\lambda = \min\left\{\frac{\gamma}{\alpha},\, \theta\right\} = \min\left\{\frac{\gamma}{\alpha},\, \frac{\beta}{\alpha} - 1\right\}\,. \tag{22}$$

This completes the description of our framework for designing and analyzing simplex counting algorithms based on oblivious sampling.

4.2 Instantiation: Oblivious Sampling Using the Shadow Hypergraph

It is time to instantiate our framework with a specific sampling and simplex-detection strategy. The simplest instantiation samples edges at rate p and detects light simplices composed of sampled edges, leading to a $\widetilde{O}(m/T^{1/k})$ space bound. We get a better bound of $\widetilde{O}(m/T^{2/(k+2)})$ algorithm by adapting the vertex-coloring-based graph algorithm of [35]: a key insight is that we color $(k-1)$-sized *subsets* of V and then pick fully monochromatic edges in the sample. Coloring single vertices would have run into the technical problem that the separation property would not hold. Our full paper [9] describes these ideas in detail.

Our best algorithm combines ideas from these initial approaches, eventually achieving a space complexity of $\widetilde{O}(m/T^{2/(k+1)})$, which is our best bound of the form $\widetilde{O}(m/T^{\lambda})$. It is therefore our method of choice for the "meager" case. Comparing with Theorem 14, we see that this bound wins when $T < m^{(k+1)/(k^2-k)}$. We continue to use the coloring idea with two added twists: we color appropriately-sized *subsets* of vertices, and we do so with vertices of the *shadow hypergraph* $H' = (V', E')$, defined in Definition 7.

To be more specific, we uniformly and independently color $(k-2)$-sized subsets of vertices V' of H' using N colors. Since each vertex in V' is essentially an ordered pair of vertices in V, we need a coloring function that maps $\binom{V \times V}{k-2}$ to $[N]$. Recall that $H' = (V', E')$ is a $(k-1)$-uniform hypergraph. During the first pass, when an edge $e = \{u_1, \ldots, u_k\}$ arrives in the stream, it implies the arrival of an edge $e' = E'$ which we store in Q iff all $(k-2)$-sized subsets within e' receive the same color. We call such an edge e' *monochromatic.*

We detect simplices using this sample Q as follows. As observed after Definition 8, each k-simplex in H corresponds to exactly one $(k-1)$-simplex in H'; we try to detect this "shadow simplex." In greater detail, let Ξ be a particular simplex in H. Let z be its minimum-ID vertex, let e_z be the edge of Ξ not incident to z and let W_z be the hyperwedge of Ξ with apex z. Note that the edges of W_z correspond to edges in H' that lie in E'_z (see Definition 7) and, since W_z is a hyperwedge, these edges form a $(k-1)$-simplex in the z-flavored component of H'. In the second pass, when e_z arrives in the stream, we detect Ξ iff this entire $(k-1)$-simplex has been stored in Q, i.e., iff all its edges are monochromatic. Algorithmically, when we see an edge e in the second pass, we go over all vertices $z \in V$, detecting a simplex whenever this e plays the role of e_z for a simplex on the vertices in $e \cup \{z\}$ in the manner just described.

The remaining details are as described in Section 4.1: we apply the heavy/light edge partitioning technique on top of the above sampling scheme and proceed as in our framework. We formalize the overall algorithm, with full details, as Algorithm 2. Note that the coloring function need not be fully random and can be chosen from a d-wise independent hash family for an appropriate constant d. The update logic for the accumulator A_L (Section 4.2) implements the detection method just described. The update logic for the accumulator A_H (Section 4.2) implements the estimator in Equation (20). Note that the heavy-simplex estimator does not use the shadow hypergraph.

Algorithm 2 Simplex counting in the “meager” case using $\widetilde{O}\big(m/T^{2/(k+1)}\big)$ bits of space.

```
procedure k-SIMPLEX-COUNT-MEAGER($\sigma$ : stream of edges of $k$-graph $H = (V, E)$)
    $d \leftarrow 2\binom{k}{k-2}$;  $r \leftarrow \binom{n^2}{k-2}$;  $\alpha \leftarrow \binom{k}{k-2} - 1$;  $p \leftarrow (\varepsilon^{-2}T^{-1}\log n)^{1/\alpha}$;  $N \leftarrow \lceil 1/p \rceil$
    $\theta \leftarrow \big(2\binom{k}{2} - k\big)/\big(\binom{k}{2} - 1\big)$;  $q \leftarrow \xi\varepsilon^{-2}\log n \cdot T^{-\theta}$
                                                  ▷ $\xi$ is a suitable constant; see Equation (15)
    $Z \leftarrow$ sample of $V$, picking each vertex independently with probability $q$
    $\mathcal{F} \subseteq [N]^{[r]} \leftarrow d$-wise independent family of hash functions of the form $h\colon [r] \to [N]$
    select a function col( ) uniformly at random from $\mathcal{F}$ to color the elements of $\binom{V'}{k-2}$
    pass 1:
        $(Q, S) \leftarrow (\varnothing, \varnothing)$
        for each hyperedge $e = \{u_1, \dots, u_k\}$ in the stream do
            let $i \in [k]$ be such that $u_i$ is the vertex in $e$ with the smallest ID
            if all $(k-2)$-sized subsets of $e \smallsetminus \{u_i\} \in E_{u_i}$ are colored the same then
                $Q \leftarrow Q \cup \Big\{\{u_1^{\langle u_i\rangle}, \dots, u_{i-1}^{\langle u_i\rangle}, u_{i+1}^{\langle u_i\rangle}, \dots, u_k^{\langle u_i\rangle}\}\Big\}$
            if $e \cap Z \neq \varnothing$ then
                $S \leftarrow S \cup \{e\}$
        create heavy/light oracle based on $S$
    pass 2:
        $(A_L, A_H) \leftarrow (0, 0)$
        let $Q^L$ be the set of hyperedges in $Q$ corresponding to light hyperedges in $E$
        for each hyperedge $e = \{u_1, \dots, u_k\}$ in the stream do
            if orcl($e$) = LIGHT then
                $A_L \leftarrow A_L + \Big|\Big\{z \in V : \bigwedge_{i=1}^{k} \Big\{\{u_1^{\langle z\rangle}, \dots, u_{i-1}^{\langle z\rangle}, u_{(i+1)}^{\langle z\rangle}, \dots, u_k^{\langle z\rangle}\} \subseteq Q^L\Big\}\Big\}\Big|$
            else
                Let $\hat{M}_e^i := \big|\{z \in Z : e \cup \{z\}$ a simplex with $i$ heavy edges$\}\big|$
                $A_H \leftarrow A_H + \sum_{i=1}^{k+1} \hat{M}_e^i / i$
    return $A_L/p^\alpha + A_H/q$
```

Analysis. We proceed in the now-established manner within our framework: we need to work out the parameters α, β, and γ.

Consider a particular k-simplex Ξ in H and let the $(k-1)$-simplex Ξ' be its "shadow simplex" in H', as in the above discussion. Then, if Ξ is light, it is detected at most once, when the hyperedge opposite to its lowest-ID vertex arrives in the stream. For this detection to actually happen, all edges of Ξ' must be monochromatic, i.e., all $\binom{k}{k-2}$ possible $(k-2)$-sized subsets of vertices in Ξ' must receive the same color. Let $p = 1/N$. The probability of this event is p^α, where $\alpha = \binom{k}{k-2} - 1 = \binom{k}{2} - 1$

Next, we observe that for two distinct simplices Ξ_1 and Ξ_2 in H, the events of their detection have nonzero correlation iff they have an edge in common. Thus, the separation property (10) holds. Moreover, when $\Xi_1 \cap \Xi_2 = \{e\}$, the correlation is in fact nonzero only if both Ξ_1 and Ξ_2 have the same minimum-ID vertex and that vertex lies in e. When this happens, the simultaneous detection event occurs iff all $(k-2)$-sized subsets of shadow vertices arising from vertices of Ξ_1 and Ξ_2 receive the same color. Counting the number of such subsets shows that the probability of simultaneous detection is p^β, for

$$\beta = 2\binom{k}{k-2} - (k-1) - 1 = 2\binom{k}{2} - k$$

Finally, an edge in E' is made monochromatic (and thus, stored in Q) with probability p^{k-2} because $k-1$ different subsets must all be colored identically. This gives $\gamma = k-2$, so

$$\lambda = \min\left\{ \frac{2\binom{k}{2} - k}{\binom{k}{2} - 1} - 1, \frac{k-2}{\binom{k}{2} - 1} \right\} = \frac{k-2}{\binom{k}{2} - 1} = \frac{2}{k+1}$$

for $k \geq 3$. Thus, our algorithm runs in $\widetilde{O}(m/T^\lambda) = \widetilde{O}\left(m/T^{2/(k+1)}\right)$ bits of memory in the worst case. We have therefore proved the following result.

▶ **Theorem 16.** *There is a 2-pass algorithm that reads a stream of m hyperedges specifying a k-uniform hypergraph H and produces an $(\varepsilon, o(1))$-estimate of $T_k(H)$; under the promise that $T_k(H) \geq T$, the algorithm runs in expected space $O(\varepsilon^{-2} \log^2 n \cdot m/T^{2/(k+1)})$.* ◢

5 Concluding Remarks

We initiated a systematic study of the simplex counting problem in a streamed hypergraph, a natural generalization of the much-studied triangle counting problem. We obtained several sublinear-space algorithms: which of them is best depends on some combinatorial parameters of the problem instance. Overall, we learned that established methods for triangle and substructure counting in graph streams are not by themselves enough and considerable effort is required to deal with the more intricate structures occurring in hypergraphs.

In some parameter regimes (what we called the "abundant" case), we obtained provably space-optimal algorithms. However, in the "meager" case, seeking algorithms with space complexity of the form m/T^λ, we established an upper bound with $\lambda = 2/(k+1)$ and a lower bound with $\lambda = 1 - 1/k$. Closing this gap seems to us to be the most compelling follow-up research question.

The simplex counting problem has the potential to impact research on pattern detection and enumeration. As triangle counting has done in the past two decades, simplex counting may offer new insights on how sampling techniques can exploit the structure of graphs and hypergraphs to extract meaningful information.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. Preliminary version in *Proc. 2nd Annual European Symposium on Algorithms*, pages 354–364, 1994.

2 Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. In *Proc. 10th Conference on Innovations in Theoretical Computer Science*, volume 124 of *LIPIcs*, pages 6:1–6:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

3 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 623–632, 2002.

4 Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proc. 14th Annual SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 16–24, 2008.

5 Suman K. Bera and Amit Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *Proc. 34th International Symposium on Theoretical Aspects of Computer Science*, pages 11:1–11:14, 2017.

6 Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *Proc. 40th International Colloquium on Automata, Languages and Programming*, pages 244–254. Springer, 2013.

7 Laurent Bulteau, Vincent Froese, Konstantin Kutzkov, and Rasmus Pagh. Triangle counting in dynamic graph streams. *Algorithmica*, 76(1):259–278, 2016. doi:10.1007/s00453-015-0036-4.

8 Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proc. 25th ACM Symposium on Principles of Database Systems*, pages 253–262, 2006.

9 Amit Chakrabarti and Themistoklis Haris. Counting simplices in hypergraph streams. *arXiv preprint*, 2021. arXiv:2112.11016.

10 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.

11 Nicholas A Christakis and James H Fowler. Social network sensors for early detection of contagious outbreaks. *PLoS ONE*, 5(9):e12948, 2010.

12 Graham Cormode and Hossein Jowhari. A second look at counting triangles in graph streams. *Theor. Comput. Sci.*, 552:44–51, 2014.

13 Graham Cormode and Hossein Jowhari. A second look at counting triangles in graph streams (corrected). *Theor. Comput. Sci.*, 683:22–30, 2017.

14 Lech Duraj, Krzysztof Kleiner, Adam Polak, and Virginia Vassilevska Williams. Equivalences between triangle and range query problems. In Shuchi Chawla, editor, *Proc. 31st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 30–47. SIAM, 2020.

15 Talya Eden, Amit Levi, Dana Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 614–633, 2015.

16 Talya Eden, Dana Ron, and C. Seshadhri. On approximating the number of k-cliques in sublinear time. *SIAM J. Comput.*, 49(4):747–771, 2020.

17 David C. Fisher. Lower bounds on the number of triangles in a graph. *J. Graph Th.*, 13(4):505–512, 1989.

18 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. *J. ACM*, 65(4):22:1–22:25, 2018.

19 Chuangyi Gui, Long Zheng, Pengcheng Yao, Xiaofei Liao, and Hai Jin. Fast triangle counting on GPU. In *2019 IEEE High Performance Extreme Computing Conference, HPEC 2019, Waltham, MA, USA, September 24-26, 2019*, pages 1–7. IEEE, 2019.

20 Rajesh Jayaram and John Kallaugher. An optimal algorithm for triangle counting in the stream. In *Proc. 24th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, volume 207 of *LIPIcs*, pages 11:1–11:11, 2021.

21 Madhav Jha, C. Seshadhri, and Ali Pinar. A space-efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox. *ACM Trans. Knowl. Discov. Data*, 9(3):15:1–15:21, 2015.

22 Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In Lusheng Wang, editor, *Proc. 11th Annual International Conference on Computing and Combinatorics (COCOON)*, pages 710–716, 2005.

23 John Kallaugher, Michael Kapralov, and Eric Price. The sketching complexity of graph and hypergraph counting. In *Proc. 59th Annual IEEE Symposium on Foundations of Computer Science*, pages 556–567. IEEE Computer Society, 2018.

24 John Kallaugher, Andrew McGregor, Eric Price, and Sofya Vorotnikova. The complexity of counting cycles in the adjacency list streaming model. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proc. 38th ACM Symposium on Principles of Database Systems*, pages 119–133. ACM, 2019.

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

26 Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Proc. 39th International Colloquium on Automata, Languages and Programming*, volume 7392 of *Lecture Notes in Computer Science*, pages 598–609. Springer, 2012.

27 Jeong Han Kim and Van H. Vu. Divide and conquer martingales and the number of triangles in a random graph. *Rand. Struct. Alg.*, 24(2):166–174, 2004.

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

29 Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Predicting positive and negative links in online social networks. In *Proc. 19th International Conference on World Wide Web (WWW)*, pages 641–650, 2010.

30 Madhusudan Manjunath, Kurt Mehlhorn, Konstantinos Panagiotou, and He Sun. Approximate counting of cycles in streams. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Proc. 19th Annual European Symposium on Algorithms*, volume 6942 of *Lecture Notes in Computer Science*, pages 677–688. Springer, 2011.

31 Willem Mantel. Problem 28 (solution by H. Gouweniak, W. Mantel, J. Texeira de Mattes, F. Schuh, and WA Whythoff). *Wiskundige Opgaven*, 10:60–61, 1907.

32 Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In *Proc. 35th ACM Symposium on Principles of Database Systems*, pages 401–411, 2016.

33 Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

34 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3):16:1–16:40, 2018.

35 Rasmus Pagh and Charalampos E. Tsourakakis. Colorful triangle counting and a MapReduce implementation. *Information Processing Letters*, 112(7):277–281, 2012.

36 A. Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proc. VLDB Endowment*, 6(14):1870–1881, 2013.

37 Alexander A. Razborov. On the minimal density of triangles in graphs. *Comb. Probab. Comput.*, 17(4):603–618, 2008.

38 Jacques Rougemont and Pascal Hingamp. Dna microarray data and contextual analysis of correlation graphs. *BMC bioinformatics*, 4(1):1–11, 2003.

39 C. Seshadhri, Ali Pinar, and Tamara G. Kolda. Wedge sampling for computing clustering coefficients and triangle counts on large graphs. *Stat. Anal. Data Min.*, 7(4):294–307, 2014.

40 Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proc. 20th International Conference on World Wide Web (WWW)*, pages 607–614. ACM, 2011.

41 Charalampos E. Tsourakakis, Mihail N. Kolountzakis, and Gary L. Miller. Triangle sparsifiers. *J. Graph Algorithms Appl.*, 15(6):703–726, 2011.

42 Sofya Vorotnikova. Improved 3-pass algorithm for counting 4-cycles in arbitrary order streaming. *CoRR*, abs/2007.13466, 2020. `arXiv:2007.13466`.

43 Virginia Vassilevska Williams. 3sum and related problems in fine-grained complexity (invited talk). In *Proc. 37th ACM Symposium on Computational Geometry*, volume 189 of *LIPIcs*, pages 2:1–2:2. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

44 Zhiang Wu, Jie Cao, Yaqiong Wang, Youquan Wang, Lu Zhang, and Junjie Wu. hPSD: a hybrid pu-learning-based spammer detection model for product reviews. *IEEE transactions on cybernetics*, 50(4):1595–1606, 2018.