On Some Estimation Problems through the Sub-linear Lens

By

Gopinath Mishra

A thesis submitted in partial fulfillment of the requirements for the degree of *Doctor of Philosophy* in *Computer Science* at *Indian Statistical Institute*

Supervisors

Arijit Bishnu and Arijit Ghosh

Advanced Computing and Microelectronics Unit

Indian Statistical Institute

203 B. T. Road, Kolkata-700108

December, 2021

ACKNOWLEDGEMENTS

First, I would like to thank my supervisors Dr. Arijit Bishnu and Dr. Arijit Ghosh for introducing me to the wonderful area of sub-linear algorithms. I express my sincere gratitude to my supervisors for their guidance, constant support, motivation, encouragement, and enthusiasm in all aspects of my Ph.D. The philosophical discussions about research and teaching, with my supervisors, helped me being motivated throughout the period, and certainly, it will help me in my future endeavors. Full freedom from my supervisors to work on whatever I liked and felt motivated to work on, along with their student friendly nature, helped me to enjoy my work and my entire Ph.D. days in general.

I thank my coauthors – Dr. Anup Bhattacharya, Dr. Sudeshna Kolay, Dr. Saket Saurabh, and Ms. Anannya Upasana for giving their consent to add the joint works with them as part of my thesis. Also, I thank my other coauthors – Dr. Sourav Chakraborty, Dr. Sameer Desai, Dr. Rogers Mathew, Ms. Subhra Mazumdar, Dr. Arindam Pal, Mr. Manaswi Paraashar, Dr. Subhabrata Paul, Dr. Sandeep Sen, and Mr. Sayantan Sen, for giving me opportunities to work with them. All of my coauthors played a unique role in my academic development.

I also thank all the faculty members of the Indian Statistical Institute, particularly of Advanced Computing and Microelectronics Unit, for their support, motivation, help, and encouragement. I would also like to thank Dr. Krishanu Maulick and Dr. Partha Pratim Mohanta for their personal help.

I am very thankful to the Indian Statistical Institute for funding me for the entire duration of Ph.D. Also, I thank the office staffs of the Indian Statistical Institute, particularly of Advanced Computing and Microelectronics Unit, for their help in making non-academic things very smooth. This in turn helped me to focus on my research. I also thank all my research scholar friends for many valuable discussions, arguments, and for making my days wonderful.

At last but not the least, I am thankful to my parents and family members for their constant support and having faith in me during my Ph.D.

(Gopinath Mishra)

Abstract

The way the data is processed or some computation done over it has changed drastically over time with the advent of the paradigm of *big data*. The study of *model-centric computation* gives the theoretical underpinning to the machine models that abstract such data processing or computation. Of the many such models, this thesis focuses on the following models: (i) sub-linear time and *query complexity*, (ii) sub-linear space or *streaming*. The main focus of this thesis is on estimating some parameters of graphs using the above mentioned models.

In the query complexity framework, this thesis starts with a query oracle called BIPARTITE INDEPENDENT SET (BIS) on graphs introduced by Beame *et al.*[ITCS'20] to estimate the number of edges in a graph with polylogarithmic BIS queries. In BIS, the input to the oracle is two disjoint subsets of vertices of the graph, and the oracle reports if there exists an edge with vertices in both of the sets. In this thesis, we first generalized BIS to TRIPARTITE INDEPENDENT SET oracle (TIS) to estimate the number of triangles using polylogarithmic TIS queries. In TIS, the input to the oracle is three pairwise disjoint subsets of vertices of the graph, and the oracle reports if there exists a triangle with vertices in each of the three sets. BIS oracle is further generalized to GENERALIZED D-PARTITE INDEPENDENT SET oracle (GPIS) for a d-uniform hypergraph, where we give d disjoint vertex sets of the hypergraph as input to the oracle, and the oracle reports if there exists a hyperedge having one vertex in each of the d sets. We show that the number of hyperedges can be estimated using polylogarithmically many queries to GPIS oracle. Note that BIS is a special case of GPIS when $d = 2$. Also, triangle estimation using TIS queries is essentially the same as that of hyperedge estimation in a 3-uniform hypergraph using GPIS queries when $d = 3$, where the vertex set of the hypergraph is same as that of the graph and three vertices form a hyperedge in the hypergraph if they form a triangle in the original graph. This thesis also considers a problem with a parameterization flavor in the query setting – the d-HITTING SET problem. Here, a parameter $k \in \mathbb{N}$ is given as input along with GPIS query oracle access to a d-uniform hypergraph; k is a parameter related to the hitting set. We show that

d-HITTING SET can be solved by using $\mathcal{O}\left(\min\{k^d \log n, k^{2d^2}\} \cdot \log k\right)$ GPIS queries, and $\Omega(k^d)$ GPIS queries are necessary to solve the problem.

In the streaming framework, we consider two problems in this thesis. The first one is about parameterized graph deletion problems in all standard graph streaming models like DYNAMIC EDGE ARRIVAL, EDGE ARRIVAL, VERTEX ARRIVAL and ADJACENCY LIST. The main upper bound result is that SUBGRAPH DELETION and MINOR DELETION problem can be solved by using $f(K)$ space in ADJACENCY LIST model, where the parameter $K \in \mathbb{N}$ is the vertex cover size of the input graph and f is a polynomial function. We complement the upper bound results by a set of lower bound results when we take solution size as our parameter for most of the streaming models.

The second problem, in the streaming framework, is a graph coloring problem that is otherwise easy in the RAM model but becomes quite non-trivial in the one-pass streaming model. In contrast to previous graph coloring problems in streaming that try to find an assignment of colors to vertices, our main work is on estimating the number of conflicting or monochromatic edges given a coloring function that is streaming along with the graph; we call the problem CONFLICT-EST. The coloring function on a vertex can be read or accessed only when the vertex is revealed in the stream. We provide algorithms for a graph that is streaming in different variants of the one-pass vertex arrival streaming model, viz. the VERTEX ARRIVAL (VA), Vertex Arrival With Degree Oracle (VADEG), VERTEX ARRIVAL IN RANDOM ORDER (VARAND) models, with special focus on the random order model and the model with degree oracle. We also provide matching lower bounds for most of the cases.

Contents

Chapter 1

Introduction

Contents

With the advent of the paradigm of *big data*, the way the data is processed or some computation is done over it, has also become important. The data may arrive in a sequence. In some cases, the data may be too big so that reading it even once may be infeasible. The study of *model-centric computation* gives the theoretical underpinning to the machine models that abstract such data processing or computation. In effect, while solving

a problem, not only the characterization of the problem is important, but also the machine model for computation becomes important. Of the many such models, this thesis focusses on the following models:

- sub-linear time and *query complexity*
- sub-linear space or *streaming*.

The main focus of this thesis is in solving some estimation problems on the above mentioned models.

1.1 Models of computation

While discussing models, we use the following terminologies and notations for *graphs* and *hypergraphs*. A simple and undirected graph is a tuple $(V(G), E(G))$, denoted as $G(V(G), E(G))$ or $G(V, E)$, where $V(G)$ denotes the set of vertices and $E(G) \subseteq \{\{u, v\} : u, v \in V(G)\}$ denotes the set of edges. An edge formed by two vertices u and v is denoted by $\{u, v\}$ or (u, v). The degree of a vertex is the number of edges incident on that vertex.

A *hypergraph* is a set system $(U(\mathcal{H}), \mathcal{F}(\mathcal{H}))$, denoted as $\mathcal{H}(U(\mathcal{H}), \mathcal{F}(\mathcal{H}))$ or $\mathcal{H}(U, \mathcal{F})$, where $U(\mathcal{H})$ denotes the set of vertices and $\mathcal{F}(\mathcal{H}) \subseteq 2^{U(\mathcal{H})}$ [1] denotes the set of hyperedges. For a hyperedge $F \in \mathcal{F}(\mathcal{H})$, $U(F)$ or simply F denotes the subset of vertices that form the hyperedge. A hypergraph is said to be a *d-uniform hypergraph* if each hyperedge has exactly d vertices. Note that a graph is a 2-uniform hypergraph.

1.1.1 Sub-linear time and query complexity

In situations where a graph cannot be accessed fully, the access is provided to the unknown graph through *query oracles* that can access the graph only through some prespecified queries. In this context, the complexity of the algorithm is measured in terms of the number of queries made to the oracle, which is known as the *query complexity*

[1] 2^S denotes the power set of set S.

of the algorithm. The time spent by the algorithm for the computational purposes (that do not involve any query to the oracle) is not counted in the query complexity of the algorithm [Gol17, Rub20]. The query complexity of a problem, using a particular query oracle $\mathcal{O}$, is the query complexity of the best algorithm to solve the problem at hand, using query oracle $\mathcal{O}$.

Estimation of graph parameters like the number of edges, triangles, cliques, etc., where the graph can be accessed through query oracles only, has been an active area of research in sub-linear time algorithms for a while [Fei06, GR08, ELRS15, ERS18, RSW18, AKK19, ABG$^+$18, CGR$^+$14, GRS11, ORRR12]. The query oracle can access the graph at different granularities – it can answer properties about the graph that are local or global in nature. By now, the LOCAL queries have been used for edge [GR08], triangle [ELRS15], clique estimation [ERS18] and have been widely accepted among researchers. The *local queries* for a graph $G = (V(G), E(G))$ are:

(i) DEGREE query: given $u \in V(G)$, the oracle reports the degree of u in $V(G)$;

(ii) NEIGHBOR query: given $u \in V(G)$ and an integer i, the oracle reports the i-th neighbor of u, if it exists; otherwise, the oracle reports that the degree of u is less than i [2];

(iii) ADJACENCY query: given $u, v \in V(G)$, the oracle reports whether $\{u, v\} \in E(G)$.

Apart from the above local queries, in the last few years, researchers have also used the RANDOM EDGE query [ABG$^+$18, AKK19], where the oracle returns an edge in the graph G uniformly at random. Notice that the randomness will be over the probability space of all edges, and hence, it is not completely justified to classify a random edge query as a local query. On the other hand, *global queries* come in different forms. The global queries considered in this thesis fall under the big umbrella of *subset* queries. Before discussing the global query oracles considered in this thesis, we briefly discuss the subset query oracle. In the *subset size estimation* problem using the *query model* of

[2] The ordering of the neighbors of the vertices are unknown to the algorithm.

computation, the subset query oracle is used to estimate the size of an unknown subset $S \subseteq U$, where U is a known universe of elements. A subset query with a subset $T \subseteq U$ asks whether $S \cap T$ is empty or not. At its core, a subset query essentially enquires about the existence of an intersection between two sets – a set chosen by the algorithm designer and an unknown set whose size we want to estimate. The study of subset queries was initiated in a breakthrough work by Stockmeyer [Sto83, Sto85] and later formalized by Ron and Tsur [RT16]. The following query oracles, which can be thought of as subset queries in the context of graphs, have been considered in the literature.

Bipartite independent set oracle (BIS) **[BHR$^+$18]:** Given two disjoint subsets A, B of the vertex set V of a graph $G(V, E)$, the BIS oracle reports whether there exists an edge with one endpoint in A and one in B.

Independent set oracle (IS) **[BHR$^+$18]:** Given a subset A of the vertex set V of a graph $G(V, E)$, the IS oracle answers whether A is an independent set.

The BIS and IS oracles have been used to estimate the size of the edge set of a graph $G(V, E)$ [BHR$^+$18, CLW20]. The edge set is the unknown set whose size one wants to estimate and the known set is formed out of subsets of the vertex set. The BIS and IS oracles report whether there is any interaction or intersection among the edge and vertex sets.

In this thesis, we focus on a generalization of the bipartite independent set oracle (BIS). BIS was introduced by Beame *et al.* [BHR$^+$18] to estimate the number of edges in the unknown graph. We first generalize BIS to *Tripartite independent set* oracle (TIS), to estimate the number of triangles in the (unknown) graph. It is defined as follows.

Tripartite independent set oracle (TIS) **[BBGM19b]:** Given three disjoint subsets A, B, C of the vertex set V of a graph $G(V, E)$, the TIS oracle reports whether there exists a triangle with one endpoint in A, one in B and one in C.

Then we further generalize BIS to *Generalized d-partite independent set* oracle (GPIS) to (i) estimate the number of hyperedges in a d-uniform hypergraph [BBGM19a],

and (ii) to study the optimization and decision version of the parameterized hitting set problem [BGK+18a].

Generalized d-partite independent set oracle (GPIS) **[BGK+18a]:** Given d pairwise disjoint subsets of vertices $A_1, \ldots, A_d$ of a d-uniform hypergraph $\mathcal{H}(U, \mathcal{F})$ as input, GPIS query oracle answers whether $m(A_1, \ldots, A_d) \neq 0$, where $m(A_1, \ldots, A_d)$ denotes the number of hyperedges in $\mathcal{H}$ having exactly one vertex in each A_i, $\forall i \in \{1, 2, \ldots, d\}$.

Note that GPIS oracle was introduced by Bishnu *et al.* [BGK+18a], and later considered by Bishnu *et al.* [BBGM19a] and Dell *et al.* [DLM20b]; though Dell *et al.* used it under the name of *colorful independence oracle*.

1.1.2 Streaming model

A *data stream* $\mathcal{P} = \{p_1, \ldots, p_i, \ldots, p_n\}$ is a sequence of data that can be read in increasing order of its indices i ($i = 1, \ldots, n$) in one or more passes. In this thesis, we consider the *one-pass, insertion only streaming model*. Here (i) one-pass means that we can access the data only once one by one, and (ii) insertion only stream means that there is no deletion of data, there is only insertion [3]. In streaming, only a sketch $\mathcal{S}$ of $\mathcal{P}$ is stored; $|\mathcal{S}| \ll |\mathcal{P}|$. A sketch is either a subset of $\mathcal{P}$ or some information derived from it. As a machine model, streaming has just the bare essentials. Thus, impossibility results, in terms of lower bounds on the sketch size, become extremely important. The seminal work of Alon *et al.* [AMS99] introduced the idea of lower bounds on space for approximating frequency moments. In this thesis, the focus is on *graph streaming*. In graph streaming, a graph is presented as a sequence of edges. In the simplest of this model, we have a stream of edge arrivals, where each edge in the stream is added to the graph seen so far, or the stream may include a dynamic mixture of arrivals and departures of edges. In either case, the primary objective is to quickly

[3] There are streaming models where we can make multiple passes over the data, and there is a streaming model (known as *turnstile model*) in which deletion is also allowed

answer some basic questions over the current state of the graph, such as finding a (maximal) matching over the current graph edges, or finding a (minimum) vertex cover, while storing only a small amount of information. In the most restrictive model, we only allow $\mathcal{O}(\log^{O(1)} n)$ bits of space for storage [4]. However, using standard techniques from communication complexity one can show that most problems do not admit such algorithms. Thus one relaxes this notion and defines what is called a *semi-streaming model*, which allows $\mathcal{O}(n \log^{O(1)} n)$ bits of space. This model has been extremely successful for graph streaming algorithms and many non-trivial algorithms have been designed in this model [AKL16, GVV17, KKSV17]. There is a vast literature on graph streaming and we refer to the survey by McGregor [McG14a] for more details. There is a wide range of different graph streaming models [CDK19, McG14a, MVV16]. In this thesis, we consider the following standard streaming models for graph problems. The general description of the following models allow self loops and multiple parallel edges between two vertices. But this thesis considers simple graphs with no self loops and no parallel edges.

(i) EDGE ARRIVAL (EA) model: The stream consists of edges of G in an arbitrary order.

(ii) DYNAMIC EDGE ARRIVAL (DEA) model: Each element of the input stream is a pair (e, state), where $e \in E(G)$ and state $\in \{\text{insert}, \text{delete}\}$ describes whether e is being inserted into or deleted from the current graph.

(iii) VERTEX ARRIVAL (VA) model: The vertices of $V(G)$ are exposed in an arbitrary order. After a vertex v is exposed, all the edges between v and neighbors of v that have already been exposed, are revealed. This set of edges are revealed one by one in an arbitrary order.

(iv) VERTEX ARRIVAL WITH DEGREE ORACLE (VADEG) [MVV16,BS20]: This model works the same as the VA model in terms of exposure of the vertex v; but we are allowed to know the *degree* of the currently exposed vertex v from a degree oracle on G.

[4] Here n denotes the number of vertices in a graph.

(v) VERTEX ARRIVAL IN RANDOM ORDER (VARAND) [SK12, TGRV14]: This model works in the same way as the VA model but the vertex sequence revealed is equally likely to be any one of the permutations of the vertices.

(vi) ADJACENCY LIST (AL) model: The vertices of $V(G)$ are exposed in an arbitrary order. When a vertex v is exposed, all the edges that are incident to v, are revealed one by one in an arbitrary order. Note that in this model each edge is exposed twice, once for each exposure of an endpoint.

1.2 Problems considered and our results

This thesis addresses several estimation problems across different computing models like streaming and query complexity. In this subsection, we describe the problems that we solve in this thesis and mention the models used for the specific problem. For ease of reading and proper contextualization, we will not put all relevant literature review in one place; rather we postpone it to relevant chapters.

1.2.1 Triangle estimation using TIS queries

Estimating the number of triangles in a graph is one of the most fundamental problems in sub-linear algorithms and it has been solved in the local query model [ELRS15]. In this thesis, we extend non-trivially the algorithmic framework of Beame *et al.* [BHR$^+$18]. Beame *et al.* used the BIPARTITE INDEPENDENT SET (BIS) query to estimate the number of edges in a graph; our extension is to use the TRIPARTITE INDEPENDENT SET query to estimate the number of triangles in a graph. In particular, we provide an algorithm that approximately counts the number of triangles in a graph using only polylogarithmic TIS queries; we work under the assumption that *the number of triangles on any edge in the graph is polylogarithmically bounded* [5]. This problem has been dealt in Chapter 3, and it has been accepted in ISAAC'19 [BBGM19b] and TOCS'21 [BBGM21].

[5] Here, polylogarithmic refers to a polynomial in $\log n$ and $1/\varepsilon$, where n is the number of vertices in the graph and ε is the approximation parameter.

1.2.2 Hyperedge estimation using GPIS queries

In this work, we estimate the number of hyperedges in a d-uniform hypergraph $\mathcal{H}(U(\mathcal{H}), \mathcal{F}(\mathcal{H}))$, where $U(\mathcal{H})$ denotes the set of vertices and $\mathcal{F}(\mathcal{H})$ denotes the set of hyperedges. We assume a GPIS query oracle access to the hypergraph $\mathcal{H}$. Our randomized algorithm (for the hyperedge estimation problem using the GPIS query oracle) outputs $\widehat{m}$ for $m(\mathcal{H})$ satisfying $(1-\varepsilon)\cdot m(\mathcal{H}) \leq \widehat{m} \leq (1+\varepsilon)\cdot m(\mathcal{H})$, where $m(\mathcal{H}) = |\mathcal{F}(\mathcal{H})|$. Moreover, the number of queries made by our algorithm, assuming d to be a constant, is polylogarithmic in the number of vertices of the hypergraph. This work can be thought of as a full generalization of Beame *et al.* [BHR+18] and the previously mentioned work [BBGM19b] to estimate the number of edges and triangles in a graph using queries to the BIPARTITE INDEPENDENT SET (BIS) and the TRIPARTITE INDEPENDENT SET (TIS) oracles, respectively. We discuss this problem in Chapter 4. This work is in arXiv [BBGM19a] and under review. Note that, independent to us, there is a similar result by Dell *et al.* [DLM20b], and they have acknowledged that our work is independent of them. Though the final results in both the works are essentially the same, the algorithms and correctness are different. We elaborate on this more later.

1.2.3 Hitting Set estimation using GPIS queries

HITTING SET is a very fundamental problem. HITTING-SET of a hypergraph $\mathcal{H}(U, \mathcal{F})$ refers to the minimum cardinality subset U' of U that intersects with all hyperedges in $\mathcal{F}$. In this work, we focus on HITTING-SET of a d-uniform hypergraph through the lens of sub-linear time algorithms. Given access to the d-uniform hypergraph through GPIS query oracle, we show that sub-linear time algorithms for hitting set have almost tight parameterized query complexity. By *parameterized query complexity*, we mean that the number of queries to the oracle is computed based on the parameter k, the size of the HITTING-SET. We consider both the parameterized decision and optimization versions of the HITTING-SET problem. d-HITTING-SET, the hitting set problem for d-uniform hypergraphs, can be solved with $\widetilde{\mathcal{O}}_d(k^d \log n)$ [6] GPIS queries. d-

[6] Here $\widetilde{\mathcal{O}}(\cdot)$ hides a $\log k$ factor.

DECISION-HITTING-SET, the decision version of d-HITTING-SET can be solved with $\widetilde{\mathcal{O}}_d\left(\min\left\{k^d \log n, k^{2d^2}\right\}\right)$ GPIS queries. We use color coding and queries to the oracles to generate subsamples from the hypergraph, that retain some structural properties of the original hypergraph. We use the stability of the sunflowers in a non-trivial way to do so. We complement these parameterized upper bounds with an almost matching parameterized lower bound that states that any algorithm that solves d-DECISION-HITTING-SET requires $\Omega\left(\binom{k+d}{d}\right)$ GPIS queries. Chapter 5 discusses these problems, and the work has been accepted in ISAAC'18 [BGK$^+$18b].

1.2.4 Streaming algorithms for graph deletion problems

The study of parameterized streaming complexity on graph problems was initiated by Fafianie *et al.* [FK14], Chitnis *et al.* [CCHM15] and Chitnis *et al.* [CCE$^+$16]. Simply put, the main goal is to design streaming algorithms for parameterized problems such that $\mathcal{O}(f(k)\log^{O(1)} n)$ space suffices, where f is an arbitrary computable function depending only on the parameter k that depends on the problem at hand. However, in the past few years, very few positive results have been established. Most of the graph problems that do have streaming algorithms of the above nature are the ones where localized checking is required, like VERTEX COVER or MAXIMUM MATCHING parameterized by the size k of the solution we are seeking. Many important parameterized problems that form the backbone of traditional parameterized complexity are known to require $\Omega(n)$ bits of storage for any streaming algorithm; e.g. FEEDBACK VERTEX SET, EVEN CYCLE TRANSVERSAL, ODD CYCLE TRANSVERSAL, TRIANGLE DELETION or the more general $\mathcal{F}$-SUBGRAPH DELETION when parameterized by solution size k. The problems mentioned here are defined in the respective chapter.

To overcome the obstacles of $\Omega(n)$ lower bound to efficient parameterized streaming algorithms, we utilize the power of structural parameterization. We focus on the vertex cover size K as the parameter for the parameterized graph deletion problems we consider. At the same time, most of the previous works in parameterized streaming complexity were restricted to the EA (edge arrival) or DEA (dynamic edge arrival) models.

We consider the four most well-studied streaming models: the EA, DEA, VA (vertex arrival) and AL (adjacency list) models. We show that $\mathcal{F}$-SUBGRAPH DELETION and $\mathcal{F}$-MINOR DELETION do not admit streaming algorithms with space complexity $o(n)$ unless we consider AL model and vertex cover size K as the parameter. But both the problems can be solved using $f(K)$ space when we we consider AL model and vertex cover size K as the parameter. Note that TRIANGLE DELETION is a special case of $\mathcal{F}$-SUBGRAPH DELETION. CLUSTER VERTEX DELETION is very closely related to TRIANGLE DELETION, and both of them admit the same time complexity in the RAM model [CFK$^+$15]. However, CLUSTER VERTEX DELETION admits space efficient streaming algorithm in the DEA model when parameterized by vertex cover K. The details of the streaming algorithms for the above problems are discussed in Chapter 6, and the work has been accepted in COCOON'20 [BGK$^+$20].

1.2.5 Monochromatic edge estimation when both vertices and colors stream

We study a graph coloring problem that is otherwise easy in the RAM model but becomes quite non-trivial in the one-pass streaming model. In contrast to previous graph coloring problems in streaming that try to find an assignment of colors to vertices [ACK19, AA20a,BCG19], our main focus is on estimating the number of conflicting or monochromatic edges given a coloring function that is streaming along with the graph; we call the problem CONFLICT-EST. The coloring function on a vertex can be read or accessed only when the vertex is revealed in the stream. If we need the color on a vertex that has streamed past, then that color, along with its vertex, has to be stored explicitly. We provide algorithms for a graph that is streaming in different variants of the one-pass vertex arrival streaming model, viz. the VERTEX ARRIVAL (VA), Vertex Arrival With Degree Oracle (VADEG), VERTEX ARRIVAL IN RANDOM ORDER (VARAND) models, with special focus on the random order model and the model with degree oracle. We also provide matching lower bounds for most of the cases. The mainstay of our work is in showing that the properties of a random order stream can be exploited to design stream-

ing algorithms for estimating the number of conflicting edges. We have also obtained a lower bound, though not matching the upper bound, for the random order model. Among all the three models vis-a-vis this problem, we can show a clear separation of power in favor of the VARAND model. Chapter 7 discusses these problems. This work has been accepted in ITCS'21 [BBMU21].

1.3 Generic notations

For $x \in \mathbb{R}$, $\exp(x)$ denotes the standard exponential function e^x. We denote the sets $\{1, \ldots, n\}$ and $\{0, \ldots, n\}$ by $[n]$ and $[n^*]$, respectively. Without loss of generality, assume that n is as a power of 2 whenever required. Let $\mathbb{E}[X]$ and $\mathbb{V}[X]$ denote the expectation and variance of a random variable X. For an event $\mathcal{E}$, $\mathcal{E}^c$ denotes the complement of $\mathcal{E}$. Throughout the thesis, the statement that "event $\mathcal{E}$ occurs with high probability" is equivalent to $\mathbb{P}(\mathcal{E}) \geq 1 - \frac{1}{n^c}$, where c is an absolute constant, unless otherwise mentioned, where n is clear from the context. The statement "a is a $(1 \pm \varepsilon)$-multiplicative approximation of b" means $|b - a| \leq \varepsilon \cdot b$. By polylogarithmic, we mean $\mathcal{O}\left((\log n/\varepsilon)^{\mathcal{O}(1)}\right)$. Unless otherwise mentioned, the notation $\widetilde{\mathcal{O}}(\cdot)$ hides a polylogarithmic term in $\mathcal{O}(\cdot)$.

Chapter 2

Preliminaries

Contents

2.1 Some probability results

The upper bounds we prove in different chapters uses a number of probability results that are stated in this section.

Proposition 2.1. Let X be a random variable. Then $\mathbb{E}[X] \leq \sqrt{\mathbb{E}[X^2]}$.

Lemma 2.2. ([DP09, Theorem 7.1]). *Let f be a function of n random variables $X_1, \ldots, X_n$ such that*

(i) Each X_i takes values from a set A_i,

(ii) $\mathbb{E}[f]$ is bounded, i.e., $0 \leq \mathbb{E}[f] \leq M$,

(iii) $\mathcal{B}$ be any event satisfying the following for each $i \in [n]$.

$$|\mathbb{E}[f \mid X_1, \ldots, X_{i-1}, X_i = a_i, \mathcal{B}^c] - \mathbb{E}[f \mid X_1, \ldots, X_{i-1}, X_i = a_i', \mathcal{B}^c]| \leq c_i.$$

Then for any $\delta \geq 0$,

$$\mathbb{P}\left(|f - \mathbb{E}[f]| > \delta + M\mathbb{P}(\mathcal{B})\right) \leq e^{-\delta^2/\sum_{i=1}^{n} c_i^2} + \mathbb{P}(\mathcal{B}).$$

Lemma 2.3 (Hoeffding's inequality [DP09])**.** *Let $X_1, \ldots, X_n$ be n independent random variables such that $X_i \in [a_i, b_i]$. Then for $X = \sum_{i=1}^{n} X_i$, the following is true for any $\delta > 0$.*

$$\mathbb{P}\left(|X - \mathbb{E}[X]| \geq \delta\right) \leq 2 \cdot e^{-2\delta^2/\sum_{i=1}^{n}(b_i - a_i)^2}.$$

Lemma 2.4 (Chernoff-Hoeffding bound [DP09])**.** *Let $X_1, \ldots, X_n$ be independent random variables such that $X_i \in [0, 1]$. For $X = \sum_{i=1}^{n} X_i$ and $\mu_l \leq \mathbb{E}[X] \leq \mu_h$, the followings hold for any $\delta > 0$.*

(i) $\mathbb{P}\left(X > \mu_h + \delta\right) \leq e^{-2\delta^2/n}$.

(ii) $\mathbb{P}\left(X < \mu_l - \delta\right) \leq e^{-2\delta^2/n}$.

Lemma 2.5 ([DP09](Chernoff-Hoeffding bound))**.** *Let $X_1, \ldots, X_N$ be independent random variables such that $X_i \in [0, 1]$. For $X = \sum_{i=1}^{N} X_i$ and $\mu = \mathbb{E}[X]$, the following holds for any $0 \leq \delta \leq 1$:*

(i) $\mathbb{P}(X \geq (1+\delta)\mu) \leq \exp\left(\frac{-\mu\delta^2}{3}\right)$;

(ii) $\mathbb{P}(X \leq (1-\delta)\mu) \leq \exp\left(\frac{-\mu\delta^2}{3}\right)$;

(iii) Furthermore, if $\mu \leq t$, then the following holds.

$$\mathbb{P}(X \geq (1+\epsilon)t) \leq \exp\left(\frac{-t\delta^2}{3}\right).$$

Lemma 2.6 ([Mul18])**.** *Let $I = \{1, \ldots, N\}$, $r \in [N]$ be a given parameter. If we sample a subset R without replacement, then the following holds for any $J \subset I$ and $\delta \in (0, 1)$.*

(i) $\mathbb{P}\left(|J \cap R| \geq (1+\delta)\,|J|\,\frac{r}{N}\right) \leq \exp\left(-\frac{\delta^2|J|r}{3N}\right)$;

(ii) $\mathbb{P}\left(|J \cap R| \leq (1-\delta)\,|J|\,\frac{r}{N}\right) \leq \exp\left(-\frac{\delta^2|J|r}{3N}\right)$*;*

(iii) Further more, we have the following if $|J| \leq k$*, then the following holds.*

$$\mathbb{P}\left(|J \cap R| \geq (1+\delta)k\frac{r}{N}\right) \leq \exp\left(-\frac{\delta^2 kr}{3N}\right)$$

Lemma 2.7. ([DP09, Theorem 3.2])**.** Let $X_1, \ldots, X_n$ be random variables such that $a_i \leq X_i \leq b_i$ and $X = \sum_{i=1}^{n} X_i$. Let $\mathcal{D}$ be the *dependent* graph, where $V(\mathcal{D}) = \{X_1, \ldots, X_n\}$ and $E(\mathcal{D}) = \{(X_i, X_j) : X_i \text{ and } X_j \text{ are dependent}\}$. Then for any $\delta > 0$,

$$\mathbb{P}(|X - \mathbb{E}[X]| \geq \delta) \leq 2e^{-2\delta^2/\chi^*(\mathcal{D})\sum_{i=1}^{n}(b_i-a_i)^2},$$

where $\chi^*(\mathcal{D})$ denotes the *fractional chromatic number* of $\mathcal{D}$.

The following lemma directly follows from Lemma 2.7.

Lemma 2.8. *Let* $X_1, \ldots, X_n$ *be indicator random variables such that there are at most* d X_j*'s on which an* X_i *depends and* $X = \sum_{i=1}^{n} X_i$*. Then for any* $\delta > 0$*,*

$$\mathbb{P}(|X - \mathbb{E}[X]| \geq \delta) \leq 2e^{-2\delta^2/(d+1)n}.$$

Lemma 2.9 (Importance sampling [BHR$^+$18])**.** *Let* $(D_1, w_1, e_1), \ldots, (D_r, w_r, e_r)$ *are the given structures and each* D_i *has an associated weight* $c(D_i)$ *satisfying*

(i) $w_i, e_i \geq 1, \forall i \in [r]$*;*

(ii) $\frac{e_i}{\rho} \leq c(D_i) \leq e_i\rho$ *for some* $\rho > 0$ *and all* $i \in [r]$*; and*

(iii) $\sum_{i=1}^{r} w_i \cdot c(D_i) \leq M$*.*

Note that the exact values $c(D_i)$*'s are not known to us. Then there exists an algorithm that finds* $(D'_1, w'_1, e'_1), \ldots, (D'_s, w'_s, e'_s)$ *such that, with probability at least* $1 - \delta$*, all of*

the above three conditions hold and

$$\left|\sum_{i=1}^{t} w_i' \cdot c(D_i') - \sum_{i=1}^{r} w_i \cdot c(D_i)\right| \leq \lambda S,$$

where $S = \sum_{i=1}^{r} w_i \cdot c(D_i)$ *and* $\lambda, \delta > 0$. *The time complexity of the algorithm is* $\mathcal{O}(r)$ *and* $s = \mathcal{O}\left(\frac{\rho^4 \log M\left(\log\log M + \log\frac{1}{\delta}\right)}{\lambda^2}\right)$.

2.2 Communication complexity

The lower bounds we prove in different chapters are by reductions from problems in *communication complexity*. Here, we briefly review the results we need. We refer the reader to [KN97] for details.

Let $f : \Omega_1 \times \Omega_2 \to \Omega$. Usually, $\Omega_1, \Omega_2 \in \{0,1\}^n$ and $\Omega \in \{0,1\}$. In communication complexity, two players Alice and Bob get as inputs $\mathbf{x} \in \Omega_1$ and $\mathbf{y} \in \Omega_2$, respectively, and the goal for the players is to devise a protocol to compute $f(\mathbf{x}, \mathbf{y})$ by exchanging as few bits of information between themselves as possible. Since its introduction by Yao [Yao79], communication complexity has found many applications in different areas of computer science like streaming algorithms, property testing, sketching, data structure, circuit complexity and auction theory [KN97, Rou16].

The *deterministic communication complexity* $D(f)$ of a function f is the minimum number of bits Alice and Bob exchange in the worst case to compute the function f. Note that in the deterministic setting, the goal is to correctly compute $f(\mathbf{x}, \mathbf{y})$ for all $\mathbf{x} \in \Omega_1$ and $\mathbf{y} \in \Omega_2$. We will denote by $D^{\to}(f)$ the *one round deterministic communication complexity* of f where only Alice sends a single message consisting of possibly multiple bits to Bob, and Bob computes the output; there is no communication from Bob to Alice. In the randomized setting, both Alice and Bob share an infinite random source. For example, both Alice and Bob can share a uniformly random infinite string of zeros and ones. The goal in the randomized setting is to give the correct answer with a probability of at least $2/3$. Note that the number $2/3$ is arbitrary and any constant more

than $1/2$ will be good enough. The randomized communication complexity $R(f)$ of f denotes the minimum number of bits exchanged by the players for the worst case input by the best randomized protocol computing f. We will denote by $R^{\rightarrow}(f)$ the *one round randomized communication complexity* of f where only Alice sends a single message to Bob, and Bob computes the output. In the randomized communication complexity setting, it is also usually assumed that both the players have an infinite source of (free) common random bits termed as *public coin random bits*, and the algorithm or protocol is termed as *public coin randomized protocol*. Moreover, the number of public coin random bits used by the players is not counted as the communication complexity of the function of interest. Note that the protocol where the players do not have access to public randomness (but each player has his or her own randomness) is known as *private coin randomized protocol*. Unless otherwise mentioned, randomized protocols in communication complexity are always public coin protocols. Communication complexity has found numerous applications in areas like streaming algorithms and property testing etc. for proving lower bounds [Rou16, RY20, Gol17].

Now let us discuss and define the following three fundamental problems and their communication complexity.

DISJOINTNESS$_n$**:** Here Alice gets a string $\mathbf{x} \in \{0,1\}^n$ and Bob gets a string $\mathbf{y} \in \{0,1\}^n$. Their objective is to decide whether there exists an $i \in [n]$ such that $x_i = y_i = 1$ [1]. Formally, DISJ$_n : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}$, and DISJ$_n(\mathbf{x}, \mathbf{y}) = 0$ if and only if there exists an $i \in [n]$ such that $x_i = y_i = 1$.

The $\{0,1\}^n$ vector can be thought of as representing the characteristic vector of a set formed from an universe $[n]$.

INDEX$_n$**:** Here Alice gets $\mathbf{x} \in \{0,1\}^n$ and Bob gets an index $j \in [n]$ and the objective of Bob is to determine if the value of $x_j = 1$. Formally, INDEX$_n : \{0,1\}^n \times [n] \rightarrow \{0,1\}$, and INDEX$_n(\mathbf{x}, j) = x_j$.

[1] x_i denotes the i-th bit of $\mathbf{x} \in \{0,1\}^n$. Also, $[n]$ denotes $\{1, \ldots, n\}$

EQUALITY$_n$**:** Here Alice gets a string $\mathbf{x} \in \{0,1\}^n$ and Bob gets a string $\mathbf{y} \in \{0,1\}^n$. Their objective is to decide whether $\mathbf{x} = \mathbf{y}$. Formally, EQ$_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, and EQ$_n(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$.

The following theorem describes the communication complexities of DISJOINTNESS$_n$.

Theorem 2.10. *[KN97]* $R(\text{DISJ}_n) = D(\text{DISJ}_n) = R^{\to}(\text{DISJ}_n) = D^{\to}(\text{DISJ}_n) = \Omega(n)$.

Though there is no difference between one way and two way communication complexities of DISJOINTNESS$_n$, there are problems in communication complexity (for example INDEX$_n$) where there is a difference between one way and two way communication complexities.

Theorem 2.11. *[KN97]* $R(\text{INDEX}_n) = D(\text{INDEX}_n) = \Theta(\log n)$. ***But*** $R^{\to}(\text{INDEX}_n) = D^{\to}(\text{INDEX}_n) = \Omega(n)$.

Note that the randomized and deterministic communication of both DISJOINTNESS$_n$ and INDEX$_n$ are the same, but there are some problems (for example EQUALITY$_n$) where the randomized complexity is much less than the deterministic counterpart.

Theorem 2.12. *[KN97]* $D(\text{EQ}_n) = D^{\to}(\text{EQ}_n) = \Omega(n)$. ***But*** $R(\text{EQ}_n) = R^{\to}(\text{EQ}_n) = \Theta(1)$.

Chapter 3

Triangle Estimation using TIS Queries

Contents

3.1 Brief description of the problem

Counting or estimating the number of triangles in a graph is a fundamental algorithmic problem in the RAM model [AYZ97, BPWZ14, IR78], streaming [ADNK14, AGM12, BKS02, BFL+06, CJ17, JSP13, JG05, KP17, KMSS12, PTTW13, TPT13] and the query model [ELRS15, GRS11]. We name this problem as TRIANGLE-ESTIMATION problem,

where we are given an $\varepsilon \in (0,1)$ as input and the objective is to report a $(1 \pm \varepsilon)$-multiplicative approximation of the number of triangles.

Notations, the query model, the problem and the result

Let $V(G)$, $E(G)$ and $T(G)$ denote the set of vertices, edges and triangles in the input graph G, respectively. When the graph G is explicit, we may write only V, E and T for the set of vertices, edges and triangles. Let $t(G) = |T(G)|$. The statement A, B, C are disjoint, means A, B, C are pairwise disjoint. For three non-empty disjoint sets A, B, $C \subseteq V(G)$, $G(A, B, C)$, termed as a *tripartite subgraph* of G, denotes the induced subgraph of $A \cup B \cup C$ in G minus the edges having both endpoints in A or B or C. The number of triangles in $G(A, B, C)$ is denoted as $t(A, B, C)$. We use the triplet (a, b, c) to denote the triangle having a, b, c as its vertices. Let Δ_u denote the number of triangles having u as one of its vertices. Let $\Delta_{(u,v)}$ be the number of triangles having (u, v) as one of its edges and $\Delta_E = \max_{(u,v)\in E(G)} \Delta_{(u,v)}$, the maximum number of triangles on any edge of G. For a set $\mathcal{U}$, "$\mathcal{U}$ is COLORED with [n]", means that each member of $\mathcal{U}$ is assigned a color out of $[n]$ colors independently and uniformly at random.

In this work, we provide the first algorithm for TRIANGLE-ESTIMATION that uses only polylogarithmic queries to query oracle TRIPARTITE INDEPENDENT SET (TIS). Recall that TIS functions as follows.

Definition 3.1 (Tripartite independent set oracle (TIS))**.** Given three non-empty disjoint subsets $V_1, V_2, V_3 \subseteq V(G)$ of a graph G, TIS query oracle answers 'YES' if and only if $t(V_1, V_2, V_3) \neq 0$.

Notice that the query oracle looks at only those triangles that have vertices in all of these sets V_1, V_2, V_3.

The result we prove in this chapter is formally stated in the following theorem. Note that the query complexity of the algorithm depends on Δ_E, where Δ_E denotes the maximum number of triangles on an edge.

Theorem 3.2 (Triangle Estimation using TIS)**.** *Let G be a graph with $\Delta_E \leq \wp$, $|V(G)| = n \geq 64$. For any $\varepsilon > 0$,* TRIANGLE-ESTIMATION *can be solved using $\mathcal{O}\left(\frac{\wp^2 \log^{18} n}{\varepsilon^4}\right)$* TIS *queries with probability at least $1 - \frac{\mathcal{O}(1)}{n^2}$.*

Note that the query complexity stated in Theorem 3.2 is poly$(\log n, \frac{1}{\varepsilon})$, even if $\wp$ is $\mathcal{O}(\log^c n)$, where c is a positive constant. We reiterate that the only bound we require is on the number of triangles on an edge; neither do we require any bound on the maximum degree of the graph, nor do we require any bound on the number of triangles incident on a vertex. It is also worth to note that, the dependecy on $\wp$ will be removed in Chapter 4. Moreover, in Chapter 4, we generalize TRIANGLE-ESTIMATION problem to HYPEREDGE-ESTIMATION problem. However, the TRIANGLE ESTIMATION problem considered in this chapter stands in its own right as there are some scenarios where the number of triangles sharing an edge is bounded. An obvious example for such graphs are graphs with bounded degree. The followings are some of the other scenarios:

(i) Consider a graph $G(P, E)$ such that the vertex set P corresponds to a subset of (points in) $\mathbb{R}^2$ and $(u, v) \in E$ if and only if the distance between u and v is exactly 1. The objective is to compute the number of triples of points from P forming an equilateral triangle having side length 1, that is, the number of triangles in G. Observe that there can be at most two triangles sharing an edge in G, that is, $\Delta_E \leq 2$.

(ii) Consider a graph $G(P, E)$ such that the vertex set P corresponds to a set of points inside an $N \times N$ square in $\mathbb{R}^2$ and $(u, v) \in E$ if and only if the distance between u and v is at most 1. The objective is to compute the number of triples of points from P forming a triangle having each side length at most 1, that is, the number of triangles in G. For *large* enough N there can be bounded number of triangles sharing an edge in G with high probability.

(iii) Consider a graph $G(V, E)$ representing a community sharing information. Each node has some information and two nodes are connected if and only if there exists an edge between the nodes. Nodes increase their information by sharing information among their neighbors in G. Observe that the information of a node is derived

by the set of neighbors. So, if two nodes have *large* number of common neighbors in G, then there is no need of an edge between the two nodes. So, the number of triangles on any edge in the graph is bounded. The objective is to compute the number of triangles in G, that is, the number of triples of nodes in G such that each pair of vertices are connected.

In (i) and (ii), TIS oracle can be implemented very efficiently. We can report a TIS query by just running a standard plane sweep algorithm in Computational Geometry that takes $\mathcal{O}(n \log n)$ running time.

Organization of the chapter

To start with, we review relevant literature in Section 3.2. We give a broad overview of the algorithm in Section 3.3. Section 3.4 gives the details of sparsification. In Section 3.5, we give exact/approximate estimation algorithm with respect to a threshold. Section 3.6 discusses about the algorithm for *coarse* estimation of the number of triangles. The final algorithm is given in Section 3.7. Section 3.8 concludes the chapter with some discussions about future improvements.

3.2 Related works

The following literature review is for both Chapters 3 and 4 as these two chapters consider related problems.

Graph parameter estimation, where one wants to estimate the number of edges, triangles or small subgraphs in a graph, is a well-studied area of research in sub-linear algorithms. Feige [Fei06], and Goldreich and Ron [GR08] gave algorithms to estimate the number of edges in a graph using degree, and degree and neighbour queries, respectively. Eden *et al.* [ELRS15] estimated the number of triangles in a graph using degree, neighbour and edge existence queries, and gave almost matching lower bound on the query complexity. This result was generalized for estimating the number of cliques of size k in [ERS18]. Since the information revealed by degree, neighbour and edge existence

queries is limited to the locality of the queried vertices, these queries are known as local queries [Gol17]. The subset queries, used in [BHR$^+$18, BBGM18, BBGM19b, DLM19], are examples of global queries, where a global query can reveal information of the graph at a much broader level.

Goldreich and Ron [GR08] solved the edge estimation problem in undirected graphs using $\tilde{O}(n/\sqrt{m})$ local queries. Dell and Lapinskas [DL18] used the INDEPENDENT SET (IS) oracle to estimate the number of edges in bipartite graphs, where an IS oracle takes a subset S of the vertex set as input and outputs whether S is an independent set or not. Their algorithm for edge estimation in bipartite graphs makes polylogarithmic IS queries and $\mathcal{O}(n)$ edge existence queries. Beame et al. [BHR$^+$18] extended the above result for the edge estimation problem in bipartite graphs to general graphs, and showed that the edge estimation problem in general graphs can be solved using $\widetilde{\mathcal{O}}\left(\min\{\sqrt{m}, n^2/m\}\right)$ [1] IS queries. Recently, Chen et al. [CLW20] improved this result to solve the edge estimation problem using only $\widetilde{\mathcal{O}}\left(\min\{\sqrt{m}, n/\sqrt{m}\}\right)$ IS queries.

3.3 Overview of the algorithm

We start with a brief overview of the algorithmic framework of Beame *et al.* [BHR$^+$18, BHR$^+$20]. It consists of subroutines for sparsifying a graph into a number of subgraphs each with reduced number of edges, and exactly or approximately counting the number of edges in these subgraphs. Sparsification constitutes the main building block of this framework. The graph is sparsified by coloring the vertices of the graph, and by looking at only those edges that exist between those pairs that are a matching of the color classes. It can be proved, by a suitable Chernoff bound, that counting the edges between the matched color classes suffice with a suitable scaling. We can ignore the other edges. Therefore, the original problem reduces to the problem of counting the number of edges in bipartite subgraphs. The next step of coarse estimation involves coarsely estimating the number of edges in each colored subgraph. Next, these subgraphs are grouped based on their coarse estimates, and subsampling is done from the groups with a relatively

[1] $\widetilde{\mathcal{O}}(\cdot)$ hides a polylogarithmic term.

large number of edges. As these bipartite subgraphs become manageable in their sizes, the edges are exactly counted in the sparse subgraphs. On the dense subgraphs, the recursive procedure continues.

Our algorithmic framework is inspired by [BHR+18] but the detailed analysis is markedly different, like the use of a relatively new concentration inequality, due to Janson [Jan04], for handling sums of random variables with bounded dependency. Apart from Lemmas 3.6 and 3.9, all other proofs require different ideas.

In Figure 3.1, we give a *flowchart* of the algorithm and show the corresponding lemmas that support the steps of the algorithm. We would like to note that the sparsification (Lemma 3.3) and the algorithm (Lemma 3.4) to estimate the number of triangles in a graph when the number of triangles is less than a threshold, are the main novel contributions.

The main idea of our algorithm is as follows. We can figure out for a given G, if the number of triangles $t(G)$ is greater than a threshold τ ((Lemma 3.4)). If $t(G) \leq \tau$, i.e., G is sparse in triangles, we compute a $(1 \pm \varepsilon)$-approximation of $t(G)$ (Lemma 3.4). Otherwise, we sparsify G to get a disjoint union of tripartite subgraphs of G that maintain $t(G)$ up to a scaling factor (Lemma 3.3). For each tripartite subgraph, if the subgraph is sparse (decided by Lemma 3.5), we count the number of triangles exactly (Lemma 3.6). Otherwise, we again sparsify (Lemma 3.7). This repeated process of sparsification may create a huge number of tripartite subgraphs. Counting the number of triangles in them is managed by doing a coarse estimation (Lemma 3.8) and taking a sample of the subgraph that maintains the number of triangles *approximately*. Each time we sparsify, we ensure that the sum of the number of triangles in the subgraphs generated by sparsification is a constant fraction of the number of triangles in the graph before sparsification, making the number of iterations $O(\log n)$.

We sparsify G by considering the partition obtained when $V(G)$ is COLORED with $[3k]$. This sparsification is done such that: (i) the sparsified graph is a union of a set of vertex disjoint tripartite subgraphs and (ii) a proper scaling of the number of triangles in the sparsified graph is a *good* estimate of $t(G)$ with high probability[2]. The proof of

[2]High probability means that the probability of success is at least $1 - \frac{1}{n^c}$ for some constant c.

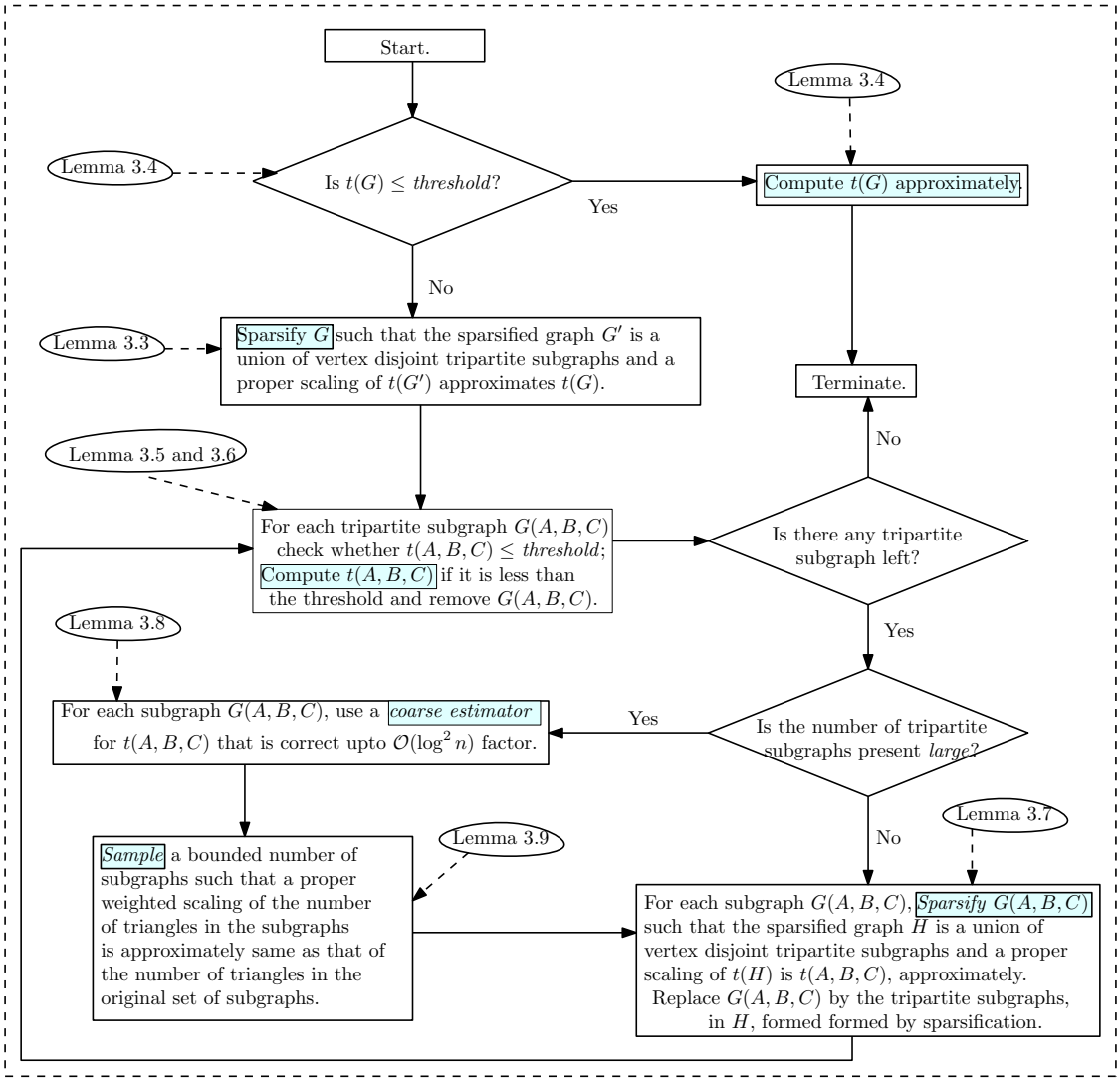

Figure 3.1: Flow chart of the algorithm. The highlighted texts indicate the basic building blocks of the algorithm. We also indicate the corresponding lemmas that support the building blocks.

the sparsification result stated next uses the *method of averaged bounded differences* and *Chernoff-Hoeffding type* inequality in bounded dependency setting by Janson [Jan04]. The detailed proof is in Section 3.4. Recall that Δ_E is the maximum number of triangles on a particular edge.

Lemma 3.3 (General Sparsification)**.** *Let $k, \wp \in \mathbb{N}$. There exists a constant κ_1 such that*

for any graph G with $\Delta_E \leq \wp$, if $V_1, \ldots, V_{3k}$ is a random partition of $V(G)$ obtained by $V(G)$ being COLORED *with $[3k]$, then*

$$\mathbb{P}\left(\left|\frac{9k^2}{2}\sum_{i=1}^{k} t(V_i, V_{k+i}, V_{2k+i}) - t(G)\right| > \kappa_1 \wp k^2 \sqrt{t(G)}\log n\right) \leq \frac{2}{n^4}.$$

We apply the sparsification corresponding to Lemma 3.3 only when $t(G)$ is above a threshold[3] to ensure that the relative error is bounded. We can decide whether $t(G)$ is at most the threshold and if it is so, we *estimate* the value of $t(G)$, using the following lemma, whose proof is given in Section 3.5.

Lemma 3.4 (Estimation with respect to a threshold)**.** *There exists an algorithm that for any graph G, a threshold parameter $\tau \in \mathbb{N}$ and an $\varepsilon \in (0,1)$, determines whether $t(G) > \tau$. If $t(G) \leq \tau$, the algorithm gives a $(1 \pm \varepsilon)$-approximation to $t(G)$ by using $\mathcal{O}(\frac{\tau \log^2 n}{\varepsilon^2})$* TIS *queries with probability at least $1 - n^{-10}$.*

Assume that $t(G)$ is *large* [4] and G has undergone sparsification. We initialize a data structure with a set of vertex disjoint tripartite graphs that are obtained after the sparsification step. For each tripartite graph $G(A, B, C)$ in the data structure, we check whether $t(A, B, C)$ is less than a threshold using the algorithm corresponding to Lemma 3.5. If it is less than a threshold, we compute the exact value of $t(A, B, C)$ using Lemma 3.6 and remove $G(A, B, C)$ from the data structure. The proofs of Lemmas 3.5 and 3.6 are given in Section 3.5.

Lemma 3.5 (Threshold for Tripartite Graph)**.** *There exists a deterministic algorithm that given any disjoint subsets $A, B, C \subset V(G)$ of any graph G and a threshold parameter $\tau \in \mathbb{N}$, can decide whether $t(A, B, C) \leq \tau$ using $\mathcal{O}(\tau \log n)$* TIS *queries.*

Lemma 3.6 (Exact Counting in Tripartite Graphs)**.** *There exists a deterministic algorithm that given any disjoint subsets $A, B, C \subset V(G)$ of any graph G, can determine the exact value of $t(A, B, C)$ using $\mathcal{O}(t(A, B, C) \log n)$* TIS *queries.*

[3] The threshold is a fixed polynomial in $\wp, \log n$ and $\frac{1}{\varepsilon}$.

[4] Large refers to a fixed polynomial in $\wp, \log n$ and $\frac{1}{\varepsilon}$

Now we are left with some tripartite graphs such that the number of triangles in each graph is more than a threshold. If the number of such graphs is not large, then we sparsify each tripartite graph $G(A, B, C)$ in a fashion almost similar to the earlier sparsification. This sparsification result formally stated in the following Lemma, has a proof similar to Lemma 3.3. We replace $G(A, B, C)$ by a constant (say, k) [5] number of tripartite subgraphs formed after sparsification.

Lemma 3.7 (Sparsification for Tripartite Graphs)**.** *Let $k, \wp \in \mathbb{N}$. There exists a constant κ_2 such that*

$$\mathbb{P}\left(\left|k^2 \sum_{i=1}^{k} t(A_i, B_i, C_i) - t(A, B, C)\right| > \kappa_2 \wp k^2 \sqrt{t(G)} \log n\right) \leq \frac{1}{n^8}$$

where A, B and C are disjoint subsets of $V(G)$ for any graph G with $\Delta_E \leq \wp$, and $A_1, \ldots, A_k$, $B_1, \ldots, B_k$ and $C_1, \ldots, C_k$ are the partitions of A, B, C formed uniformly at random, respectively.

If we have a large number of vertex disjoint tripartite subgraphs of G and each subgraph contains a large number of triangles, then we *coarsely* estimate the number of triangles in each subgraph which is correct up to $\mathcal{O}(\log^2 n)$ factor by using the algorithm corresponding to the following Lemma, whose proof is in Section 3.6.

Lemma 3.8 (Coarse Estimation)**.** *There exists an algorithm that given disjoint subsets $A, B, C \subset V(G)$ of any graph G, returns an estimate $\widetilde{t}$ satisfying*

$$\frac{t(A, B, C)}{64 \log^2 n} \leq \widetilde{t} \leq 64 t(A, B, C) \log^2 n$$

with probability at least $1 - n^{-9}$. Moreover, the query complexity of the algorithm is $\mathcal{O}(\log^4 n)$.

Remark 3.1. The coarse estimation algorithm for the number of edges by Beame *et al.* [BHR+18] takes two disjoint sets A and B and reports $\widetilde{e}$ such that $\frac{e(A,B)}{c \log n} \leq \widetilde{e} \leq$

[5] In our algorithm, k is a constant. However, Lemma 3.7 and Lemma 3.3 holds for any $k \in \mathbb{N}$.

$c \log n$, where c is a constant and $e(A, B)$ denotes the number of edges with one endpoint in A and one in B. Our COARSE-ESTIMATE-TRIANGLE algorithm is similar in structure to the coarse estimation algorithm for edge estimation, but a suitable extension for triangle via delicate parameter setting and careful analysis. The difficulty in extension is inherent as triangles are more complicated structures than that of edges. One may think that we are estimating triangle using TIS which is powerful than that of BIS. However, the relative power hierarchy of BIS and TIS is not at all clear.

After estimating the number of triangles in each subgraph coarsely, we approximately maintain the triangle count using the following sampling result which is a direct consequence of the *Importance Sampling Lemma* of [BHR+18]. [6]

Lemma 3.9 ([BHR+18]). *Let $(A_1, B_1, C_1, w_1), \ldots, (A_r, B_r, C_r, w_r)$ be the tuples present in the data structure and e_i be the corresponding coarse estimation for $t(A_i, B_i, C_i), i \in [r]$, such that*

(i) *$\forall i \in [r]$, we have w_i, $e_i \geq 1$;*

(ii) *$\forall i \in [r]$, we have $\frac{e_i}{\rho} \leq t(A_i, B_i, C_i) \leq e_i \rho$ for some $\rho > 0$; and*

(iii) *$\sum_{i=1}^{r} w_i \cdot t(A_i, B_i, C_i) \leq M$.*

Note that the exact values $t(A_i, B_i, C_i)$'s are not known to us. Then there exists an algorithm that finds $(A'_1, B'_1, C'_1, w'_1), \ldots, (A'_s, B'_s, C'_s, w'_s)$ such that all of the above three conditions hold and

$$\left| \sum_{i=1}^{s} w'_i \cdot t(A'_i, B'_i, C'_i) - \sum_{i=1}^{r} w_i \cdot t(A_i, B_i, C_i) \right| \leq \lambda S,$$

with probability at least $1 - \delta$, where $S = \sum_{i=1}^{r} w_i \cdot t(A_i, B_i, C_i)$ and $\lambda, \delta > 0$. Also,

$$s = \mathcal{O}\left(\lambda^{-2} \rho^4 \log M \left(\log \log M + \log \frac{1}{\delta} \right) \right).$$

[6] For the exact statement of the Importance Sampling Lemma see Lemma 2.9 in Section 2.1.

Now again, for each tripartite graph $G(A, B, C)$, we check whether $t(A, B, C)$ is less than a threshold using the algorithm corresponding to Lemma 3.5. If yes, then we can compute the exact value of $t(A, B, C)$ using Lemma 3.6 and remove $G(A, B, C)$ from the data structure. Otherwise, we iterate on all the required steps discussed above as shown in Figure 3.1. Observe that each iteration uses polylogarithmically [7] many queries. Now, note that the number of triangles reduces by a constant factor after each sparsification step. So, the number of iterations is bounded by $\mathcal{O}(\log n)$. Hence, the query complexity of our algorithm is polylogarithmic. This completes the high level description of our algorithm.

3.4 Sparsification step

In this Section, we prove Lemma 3.3. The proof of Lemma 3.7 is similar.

Lemma 3.10 (Lemma 3.3 restated)**.** *Let $k, \wp \in \mathbb{N}$. There exists a constant κ_1 such that for any graph G with $\Delta_E \leq \wp$, if $V_1, \ldots, V_{3k}$ is a random partition of $V(G)$ obtained by $V(G)$ being* COLORED *with $[3k]$, then*

$$\mathbb{P}\left(\left|\frac{9k^2}{2}\sum_{i=1}^{k} t(V_i, V_{k+i}, V_{2k+i}) - t(G)\right| > \kappa_1 \wp k^2 \sqrt{t(G)}\log n\right) \leq \frac{2}{n^4}.$$

Proof. $V(G)$ is COLORED with $[3k]$. Let $V_1, \ldots, V_{3k}$ be the resulting partition of $V(G)$. Let Z_i be the random variable that denotes the color assigned to the i^{th} vertex. For $i \in [3k]$, $\pi(i)$ is a set of three colors defined as follows: $\pi(i) = \{i, (1 + (i + k - 1) \bmod 3k), (1 + (i + 2k - 1) \bmod 3k)\}$.

Definition 3.11. A triangle (a, b, c) is said to be *properly colored* if there exists a bijection in terms of coloring from $\{a, b, c\}$ to $\pi(i)$.

Let $f(Z_1, \ldots, Z_n) = \sum_{i=1}^{k} t(V_i, V_{k+i}, V_{2k+i})$. Note that f is the number of triangles that are properly colored. The probability that a triangle is properly colored is $\frac{2}{9k^2}$. So, $\mathbb{E}[f] = \frac{2t(G)}{9k^2}$.

[7] Polylogarithmic refers to a polynomial in $\wp, \log n$ and $\frac{1}{\varepsilon}$

Let us focus on the instance when vertices $1, \ldots, t-1$ are already colored and we are going to color vertex t. Let S_ℓ (resp., S_r) be the set of triangles in G having t as one of the vertices and other two vertices are from $[t-1]$ (resp., $[n] \setminus [t]$). $S_{\ell r}$ be the set of triangles in G such that t is a vertex and the second and third vertices are from $[t-1]$ and $[n] \setminus [t]$, respectively.

Given that the vertex t is colored with color $c \in [3k]$, let $N_\ell^c, N_r^c, N_{\ell r}^c$ be the random variables that denote the number of triangles in S_ℓ, S_r and $S_{\ell r}$ that are properly colored, respectively. Also, let $\mathbb{E}_f^t$ be a random variable that is a function of f, random variables $Z_1, \ldots, Z_{t-1}$, and $a_t, a_t' \in [3k]$. Moreover, $\mathbb{E}_f^t$ denotes the absolute difference in the conditional expectation of the number of triangles that are properly colored when vertex t is colored with color a_t and when vertex t is colored with color a_t'. By considering the vertices in S_ℓ, S_r and $S_{\ell r}$ separately, we can bound $\mathbb{E}_f^t$.

$$\begin{aligned}
\mathbb{E}_f^t &= |\mathbb{E}[f \mid Z_1, \ldots, Z_{t-1}, Z_t = a_t] - \mathbb{E}[f \mid Z_1, \ldots, Z_{t-1}, Z_t = a_t']| \\
&= \left| N_\ell^{a_t} - N_\ell^{a_t'} + \mathbb{E}\left[N_r^{a_t} - N_r^{a_t'}\right] + \mathbb{E}\left[N_{\ell r}^{a_t} - N_{\ell r}^{a_t'}\right] \right| \\
&\leq \left| N_\ell^{a_t} - N_\ell^{a_t'} \right| + \mathbb{E}\left[\left| N_r^{a_t} - N_r^{a_t'} \right|\right] + \mathbb{E}\left[\left| N_{\ell r}^{a_t} - N_{\ell r}^{a_t'} \right|\right]
\end{aligned}$$

Now, consider the following claim, which we prove later.

Claim 3.12. *(a)* $\mathbb{P}(\mid N_\ell^{a_t} - N_\ell^{a_t'} \mid < 8\sqrt{\wp \Delta_t \log n}) \geq 1 - 4n^{-8}$*;*

(b) $\mathbb{E}[\mid N_r^{a_t} - N_r^{a_t'} \mid] \leq \sqrt{\wp \Delta_t}/k$*;*

(c) $\mathbb{E}[\mid N_{\ell r}^{a_t} - N_{\ell r}^{a_t'} \mid] < 6\wp\sqrt{\Delta_t \log n}$.

Note that Δ_t is the number of triangles having t as one of its vertices and we are not assuming any bound on Δ_t. We assume Δ_E, that is the number of triangles on any edge, is bounded.

Let $c_t = 15\wp\sqrt{\Delta_t \log n}$. From the above claim, we have

$$\mathbb{E}_f^t < 8\sqrt{\wp \Delta_t \log n} + \frac{\sqrt{\wp \Delta_t}}{k} + 6\wp\sqrt{\Delta_t \log n} \leq 15\wp\sqrt{\Delta_t \log n} = c_t$$

with probability at least $1 - \frac{4}{n^8}$. Let $\mathcal{B}$ be the event that there exists $t \in [n]$ such that $\mathbb{E}_f^t > c_t$. By the union bound over all $t \in [n]$, $\mathbb{P}(\mathcal{B}) \leq \frac{4}{n^7}$.

Using the method of *averaged bounded difference* [DP09] (See Lemma 2.2 in Appendix 2.1), we have

$$\mathbb{P}\left(|f - \mathbb{E}[f]| > \delta + t(G)\mathbb{P}(\mathcal{B})\right) \leq e^{-\delta^2/\sum_{t=1}^{n} c_t^2} + \mathbb{P}(\mathcal{B}).$$

We set $\delta = 60\wp\sqrt{t(G)}\log n$. Observe that $\sum_{t=1}^{n} c_t^2 = 225\wp^2 \log n \sum_{t=1}^{n} \Delta_t = 675\wp^2 t(G)\log n$. Hence,

$$\mathbb{P}\left(\left|f - \frac{2t(G)}{9k^2}\right| > 60\wp\sqrt{t(G)}\log n + t(G)\mathbb{P}(\mathcal{B})\right) \leq \frac{1}{n^4} + \frac{1}{n^7},$$

that is,

$$\mathbb{P}\left(\left|\frac{9k^2}{2}f - t(G)\right| > 270\wp k^2\sqrt{t(G)}\log n + \frac{9k^2}{2}\cdot\frac{t(G)}{n^7}\right) \leq \frac{1}{n^4} + \frac{1}{n^7}.$$

Since, $\frac{9k^2}{2}\cdot\frac{t(G)}{n^7} < \wp k^2\sqrt{t(G)}\log n$, we get

$$\mathbb{P}\left(\left|\frac{9k^2}{2}f - t(G)\right| > 271\wp k^2\sqrt{t(G)}\log n\right) \leq \frac{2}{n^4}.$$

□

To finish the proof of Lemma 3.3, we need to prove Claim 3.12. For that, we need the following definition and intermediate result (Lemma 3.14) that is stated in terms of objects, which in the current context can be thought of as vertices.

Definition 3.13. Let $\mathcal{X}$ be a set of u objects COLORED with $[3k]$. Let $\alpha, \beta \in [3k]$ and $\alpha \neq \beta$. A pair of objects $\{a, b\}$ is said to be colored with $\{\alpha, \beta\}$ if there is a bijection in terms of coloring from $\{a, b\}$ to $\{\alpha, \beta\}$. For a single object $o \in \mathcal{X}$, o is colored with $\{\alpha, \beta\}$ means o is either colored with α or β.

Example: Let us take $u = 5$, $\chi = \{o_1, \ldots, o_5\}$ and $k = 1$. Let $c : \chi \to [3]$ be the coloring function assigned to the objects in χ such that $c(o_1) = 2, c(o_2) = 3, c(o_3) = 3, c(o_4) = 1$ and $c(o_5) = 2$. So, we can say that, $\{o_2, o_5\}$ is colored with $\{2, 3\}$ and $\{o_1, o_4\}$ is colored with $\{1, 2\}$, but $\{o_4, o_5\}$ is not colored with $\{1, 3\}$, etc. For a single object, say o_3, we can say that o_3 is colored with $\{3, 2\}$.

Recall Definition 3.11. A triangle incident on t is properly colored if the pair of vertices in the triangle other than t, is colored with $\pi(Z_t) \setminus \{Z_t\}$. Note that, Claim 3.12 bounds the difference in the number of properly colored triangles incident on t when $Z_t = a_t$ and when $Z_t = a'_t$, that is, the difference in the number of triangles whose pair of vertices other than t is colored with $\pi(a_t) \setminus \{a_t\}$ and that is colored with $\pi(a'_t) \setminus \{a'_t\}$. As, a vertex can be present in many pairs, proper coloring of one triangle, incident on t, is dependent on the proper coloring of another triangle. However, this dependency is bounded due to our assumption $\Delta_E \leq \wp$. Now, let us consider the following Lemma.

Lemma 3.14. *Let $\mathcal{X}$ be a set of u objects* COLORED *with $[3k]$. $\mathcal{F}$ be a set of v pairs of objects such that an object is present in at most $\wp$ ($\wp \leq v$) pairs and $\mathcal{P} \subseteq \mathcal{X}$ be a set of w objects, $\mathcal{F}_{\{\alpha,\beta\}} \subseteq \mathcal{F}$ be a set of pairs of objects that are colored with $\{\alpha, \beta\}$, $M_{\{\alpha,\beta\}} = \left|\mathcal{F}_{\{\alpha,\beta\}}\right|$, and $\mathcal{P}_{\{\alpha,\beta\}} \subseteq \mathcal{P}$ be the set of objects that are colored with $\{\alpha, \beta\}$ and $N_{\{\alpha,\beta\}} = \left|\mathcal{P}_{\{\alpha,\beta\}}\right|$. Then, we have*

(i) $\mathbb{P}\left(\left|M_{\{\alpha,\beta\}} - M_{\{\alpha',\beta'\}}\right| \geq 8\sqrt{\wp v \log u}\right) \leq \frac{4}{u^8}$,

(ii) $\mathbb{E}\left[\left|M_{\{\alpha,\beta\}} - M_{\{\alpha',\beta'\}}\right|\right] \leq \frac{\sqrt{\wp v}}{k}$, *and*

(iii) $\mathbb{P}\left(\left|N_{\{\alpha,\beta\}} - N_{\{\alpha',\beta'\}}\right| \geq 4\sqrt{w \log u}\right) \leq \frac{4}{u^8}$.

Proof. (i) Let $\mathcal{F} = \{\{a_1, b_1\}, \ldots, \{a_v, b_v\}\}$. Let X_i be the indicator random variable such that $X_i = 1$ if and only if $\{a_i, b_i\}$ is colored with $\{\alpha, \beta\}$, where $i \in [v]$. Note that $M_{\{\alpha,\beta\}} = \sum_{i=1}^{v} X_i$. Also, $\mathbb{E}[X_i] = \frac{2}{9k^2}$, hence $\mathbb{E}[M_{\{\alpha,\beta\}}] = \frac{2v}{9k^2}$.

X_i and X_j are dependent if and only if $\{a_i, b_i\} \cap \{a_j, b_j\} \neq \emptyset$. As each object can be present in at most $\wp$ pairs of objects, there are at most $2\wp$ X_j's on which an X_i

depends. Now using *Chernoff-Hoeffding's type bound in the bounded dependent setting* [DP09] (see Lemma 2.8 in Section 2.1), we have

$$\mathbb{P}\left(\left|M_{\{\alpha,\beta\}} - \frac{2v}{9k^2}\right| \geq 4\sqrt{\wp v \log u}\right) \leq \frac{2}{u^8}.$$

Similarly, one can also show that $\mathbb{P}\left(\left|M_{\{\alpha',\beta'\}} - \frac{2v}{9k^2}\right| \geq 4\sqrt{\wp v \log u}\right) \leq \frac{2}{u^8}$. Note that

$$\left|M_{\{\alpha,\beta\}} - M_{\{\alpha',\beta'\}}\right| \leq \left|M_{\{\alpha,\beta\}} - \frac{2v}{9k^2}\right| + \left|M_{\{\alpha',\beta'\}} - \frac{2v}{9k^2}\right|.$$

Hence,

$$\begin{aligned}
&\mathbb{P}\left(\left|M_{\{\alpha,\beta\}} - M_{\{\alpha',\beta'\}}\right| \geq 8\sqrt{\wp v \log u}\right) \\
&\leq \mathbb{P}\left(\left|M_{\{\alpha,\beta\}} - \frac{2v}{9k^2}\right| + \left|M_{\{\alpha',\beta'\}} - \frac{2v}{9k^2}\right| \geq 8\sqrt{\wp v \log u}\right) \\
&\leq \mathbb{P}\left(\left|M_{\{\alpha,\beta\}} - \frac{2v}{9k^2}\right| \geq 4\sqrt{\wp v \log u}\right) + \mathbb{P}\left(\left|M_{\{\alpha',\beta'\}} - \frac{2v}{9k^2}\right| \geq 4\sqrt{\wp v \log u}\right) \\
&\leq 4u^{-8}.
\end{aligned}$$

(ii) Let $X_i, i \in [v]$, be the random variable such that $X_i = 1$ if $\{a_i, b_i\}$ is colored with $\{\alpha, \beta\}$; $X_i = -1$ if $\{a_i, b_i\}$ is colored with $\{\alpha', \beta'\}$; $X_i = 0$, otherwise. Let $X = \sum_{i=1}^{v} X_i$. Note that $M_{\{\alpha,\beta\}} - M_{\{\alpha',\beta'\}} = X = \sum_{i=1}^{v} X_i$.

So, we need to bound $\mathbb{E}[|X|]$ to prove the claim.

The random variables X_i and X_j are dependent if and only if $\{a_i, b_i\} \cap \{a_j, b_j\} \neq \emptyset$. As each object can be present in at most $\wp$ pairs of objects, there are at most $2\wp$ X_j's on which an X_i depends. Observe that

$$\mathbb{P}(X_i = 1) = \mathbb{P}(X_i = -1) = \frac{2}{9k^2}.$$

So, $\mathbb{E}[X_i] = 0$ and $\mathbb{E}[X_i^2] = \frac{4}{9k^2}$.

If X_i and X_j are independent, then

$$\mathbb{E}[X_i X_j] = \mathbb{E}[X_i] \cdot \mathbb{E}[X_j] = 0.$$

If X_i and X_j are dependent, then

$$\mathbb{E}[X_i X_j] \leq \mathbb{P}(X_i X_j = 1).$$

$$\begin{aligned}
\mathbb{P}(X_i X_j = 1) &= \mathbb{P}(X_i = 1, X_j = 1) + \mathbb{P}(X_i = -1, X_j = -1) \\
&= \mathbb{P}(X_i = 1) \cdot \mathbb{P}(X_j = 1 \mid X_i = 1) \\
&\quad + \mathbb{P}(X_i = -1) \cdot \mathbb{P}(X_j = -1 \mid X_i = -1) \\
&= \frac{2}{9k^2} \cdot \frac{1}{3k} + \frac{2}{9k^2} \cdot \frac{1}{3k} = \frac{4}{27k^3}
\end{aligned}$$

Using the expression $\mathbb{E}[X^2] = \sum_{i=1}^{v} \mathbb{E}[X_i^2] + 2 \cdot \sum_{1 \leq i < j \leq v} \mathbb{E}[X_i X_j]$ and recalling the fact that each X_i depends on at most $2\wp$ other X_j's, we get

$$\mathbb{E}[X^2] \leq v \cdot \frac{4}{9k^2} + 2\wp v \cdot \frac{4}{27k^3} \leq \frac{8\wp v}{9k^2}.$$

Now, using $\mathbb{E}[|X|] \leq \sqrt{\mathbb{E}[X^2]}$, we get $\mathbb{E}[|X|] < \frac{\sqrt{\wp v}}{k}$.

(iii) Let $\mathcal{P} = \{o_1, \ldots, o_w\}$ be the set of w objects. Let $X_i, i \in [w]$, be the indicator random variable such that $X_i = 1$ if and ony if o_i is colored with $\{\alpha, \beta\}$. Note that $N_{\{\alpha,\beta\}} = \sum_{i=1}^{w} X_i$. Observe that $\mathbb{E}[X_i] = \frac{2}{3k}$ and hence, $\mathbb{E}\left[N_{\{\alpha,\beta\}}\right] = \frac{2w}{3k}$. Note that X_i and X_j are independent. Applying Hoeffding's inequality (See Lemma 2.3 in Section 2.1), we get

$$\mathbb{P}\left(\left|N_{\{\alpha,\beta\}} - \frac{2w}{3k}\right| \geq 2\sqrt{w \log u}\right) \leq \frac{2}{u^8}.$$

Similarly, we can aso show that $\mathbb{P}\left(\left|N_{\{\alpha',\beta'\}} - \frac{2w}{3k}\right| \geq 2\sqrt{w\log u}\right) \leq \frac{2}{u^8}$. Hence,

$$\begin{aligned}
&\mathbb{P}(\left|N_{\{\alpha,\beta\}} - N_{\{\alpha',\beta'\}}\right| \geq 4\sqrt{w\log u}) \\
&\leq \mathbb{P}\left(\left|N_{\{\alpha,\beta\}} - \frac{2w}{3k}\right| + \left|N_{\{\alpha',\beta'\}} - \frac{2w}{3k}\right| \geq 4\sqrt{w\log u}\right) \\
&\leq \mathbb{P}\left(\left|N_{\{\alpha,\beta\}} - \frac{2w}{3k}\right| \geq 2\sqrt{w\log u}\right) + \mathbb{P}\left(\left|N_{\{\alpha',\beta'\}} - \frac{2w}{3k}\right| \geq 2\sqrt{w\log u}\right) \\
&\leq \frac{4}{u^8}.
\end{aligned}$$

□

We will now give the proof of Claim 3.12.

Proof of Claim 3.12. (a) Let $S_\ell = \{(a_1, b_1, t), \ldots, (a_v, b_v, t)\}$. Note that $v \leq \Delta_t$. As $\Delta_E \leq \wp$, each vertex in $[n]$ can be present in at most $\wp$ pairs of S_ℓ. Now we apply Lemma 3.14. Set $\mathcal{X} = [n]$ and $\mathcal{F} = S_\ell$ in Lemma 3.14. Observe that $N_\ell^{a_t} = M_{\pi(a_t)\setminus\{a_t\}}$ and $N_\ell^{a'_t} = M_{\pi(a'_t)\setminus\{a'_t\}}$. So, by of Lemma 3.14 (i),

$$\mathbb{P}\left(\left|N_\ell^{a_t} - N_\ell^{a'_t}\right| \geq 8\sqrt{\wp v\log n}\right) \leq \frac{4}{n^8}.$$

This implies $\mathbb{P}\left(\left|N_\ell^{a_t} - N_\ell^{a'_t}\right| \geq 8\sqrt{\wp\Delta_t\log n}\right) \leq \frac{4}{n^8}$.

(b) Let $S_r = \{(t, a_1, b_1), \ldots, (t, a_v, b_v)\}$. Note that $v \leq \Delta_t$, the number of triangles incident on vertex t. As $\Delta_E \leq \wp$, each vertex in $[n]$ can be present in at most $\wp$ pairs of S_r. Now we apply Lemma 3.14. Set $\mathcal{X} = [n]$ and $\mathcal{F} = S_r$ in Lemma 3.14. Observe that $N_r^{a_t} = M_{\pi(a_t)\setminus\{a_t\}}$ and $N_r^{a'_t} = M_{\pi(a'_t)\setminus\{a'_t\}}$. By Lemma 3.14 (ii), we get

$$\mathbb{E}\left[\left|N_r^{a_t} - N_r^{a'_t}\right|\right] \leq \frac{\sqrt{\wp v}}{k} \leq \frac{\sqrt{\wp\Delta_t}}{k}.$$

(c) Let $S_{\ell r} = \{(a_1, t, b_1), \ldots, (a_w, t, b_w)\}$. Without loss of generality, assume that $a_i \in [t-1]$ and $b_i \in [n]\setminus[t]$. Note that $w \leq \Delta_t$. Given that the vertex t is colored

with color c and we know $Z_1, \ldots, Z_{t-1}$, define the set P_c as

$$P_c := \{(a,t,b) \in S_{\ell r} \ : t \text{ is colored with } c \text{ and } \mathbb{P}((a,t,b) \text{ is properly colored}) > 0\}.$$

Let $Q_c = |P_c|$. Observe that for $(a,t,b) \in S_{\ell r}$, $\mathbb{P}((a,t,b)$ is properly colored$) > 0$ if and only if a is colored with some color in $\pi(c) \setminus \{c\}$. Now we apply Lemma 3.14. Set $\mathcal{X} = [n]$, $\mathcal{P} = \{a_1, \ldots, a_w\}$. Observe that $\mathcal{P}_{\pi(a_t)\setminus a_t} = P_{a_t}$ and $\mathcal{P}_{\pi(a'_t)\setminus a'_t} = P_{a'_t}$. By (iii) of Lemma 3.14,

$$\mathbb{P}\left(\left|Q_{a_t} - Q_{a'_t}\right| \geq 4\sqrt{w \log n}\right) \leq \frac{4}{n^8}.$$

Let $\mathcal{E}$ be the event that $\left|Q_{a_t} - Q_{a'_t}\right| \geq 4\sqrt{w \log n}$. So, $\mathbb{P}(\mathcal{E}) \leq \frac{4}{n^8}$. Assume that $\mathcal{E}$ has not occurred. Let $P = P_{a_t} \cap P_{a'_t} = \{(x_1, t, y_1), \ldots, (x_q, t, y_q)\}$. Note that $q \leq w \leq \Delta_t$. Recall that Z_x is the random variable that denotes the color assigned to vertex $x \in [n]$. Let $X_i, i \in [q]$, be the random variable such that $X_i = 1$ if y_i is colored with $\pi(a_t) \setminus \{Z_{x_i}, a_t\}$; $X_i = -1$ if y_i is colored with $\pi(a'_t) \setminus \{Z_{x_i}, a'_t\}$; $X_i = 0$, otherwise. Let $X = \sum_{i=1}^{q} X_i$. Observe that X_i and X_j are dependent if and only if $y_i = y_j$. As $\Delta_E \leq \wp$, there can be at most $\wp$ y_j's such that $y_i = y_j$. So, an X_i depends on at most $\wp$ other X_j's.

Observe that $\mathbb{P}(X_i = 1) = \mathbb{P}(X_i = -1) = \frac{1}{3k}$. So, $\mathbb{E}[X_i = 0]$ and $\mathbb{E}[X_i^2] = \frac{2}{3k}$. If X_i and X_j are independent, then $\mathbb{E}[X_i X_j] = 0$. If X_i and X_j are dependent, then

$$\begin{aligned}
\mathbb{E}[X_i X_j] &\leq \mathbb{P}(X_i = 1, X_j = 1) + \mathbb{P}(X_i = -1, X_j = -1) \\
&\leq \mathbb{P}(X_i = 1) + \mathbb{P}(X_j = -1) \\
&= \frac{2}{3k}.
\end{aligned}$$

Using the expression $\mathbb{E}[X^2] = \sum_{i=1}^{v} \mathbb{E}[X_i^2] + 2 \cdot \sum_{1 \leq i < j \leq v} \mathbb{E}[X_i X_j]$ and the fact that each X_i depends on at most $\wp$ other X_j's, we get

$$\mathbb{E}[X^2] \leq v \cdot \frac{2}{3k} + \wp v \cdot \frac{2}{3k} \leq \frac{\wp v}{k} \leq \frac{\wp \Delta_t}{k}.$$

Since, $\mathbb{E}[|X|] \leq \sqrt{\mathbb{E}[X^2]}$, we get $\mathbb{E}[|X|] \leq \sqrt{\frac{\wp\Delta_t}{k}}$. Using $\Delta_E \leq \wp$, we have

$$\begin{aligned}\mathbb{E}[\mid N_{\ell r}^{a_t} - N_{\ell r}^{a'_t} \mid \mid \mathcal{E}^c] &= \wp\cdot \mid Q_{a_t} - Q_{a'_t} \mid + \mathbb{E}[|X|] \\ &< 4\wp\sqrt{\Delta_t \log n} + \sqrt{\frac{\wp\Delta_t}{k}} < 5\wp\sqrt{\Delta_t \log n}.\end{aligned}$$

Observe that $\mathbb{E}[\mid N_{\ell r}^{a_t} - N_{\ell r}^{a'_t} \mid \mid \mathcal{E}] \leq w \leq \Delta_t$. Putting everything together,

$$\begin{aligned}\mathbb{E}\left[\mid N_{\ell r}^{a_t} - N_{\ell r}^{a'_t} \mid\right] &= \mathbb{P}(\mathcal{E})\cdot\mathbb{E}\left[\mid N_{\ell r}^{a_t} - N_{\ell r}^{a'_t} \mid \mid \mathcal{E}\right] + \mathbb{P}(\mathcal{E}^c)\cdot\mathbb{E}\left[\mid N_{\ell r}^{a_t} - N_{\ell r}^{a'_t} \mid \mid \mathcal{E}^c\right] \\ &< \frac{4}{n^8}\cdot\Delta_t + 1\cdot 5\wp\sqrt{\Delta_t \log n} \leq 6\wp\sqrt{\Delta_t \log n}\end{aligned}$$

□

3.5 Estimation: exact and approximate

In this Section, we prove Lemmas 3.4 (restated as Lemma 3.17), 3.5 (restated as Lemma 3.16) and 3.6 (restated as Lemma 3.15). We first prove Lemmas 3.5 and 3.6, whose proofs are very similar. Then we prove Lemma 3.4 that in turn uses Lemma 3.5.

Lemma 3.15 (Lemma 3.6 restated)**.** *There exists a deterministic algorithm that given any disjoint subsets $A, B, C \subset V(G)$ of any graph G, can determine the exact value of $t(A, B, C)$ using $\mathcal{O}(t(A, B, C)\log n)$* TIS *queries.*

Proof. We initialize a tree $\mathcal{T}$ with (A, B, C) as the root. We build the tree such that each node is labeled with either 0 or 1. If $t(A, B, C) = 0$, we label the root with 0 and terminate. Otherwise, we label the root with 1 and do the following as long as there is a leaf node (U, V, W) labeled with 1.

(i) If $t(U, V, W) = 0$, then we label (U, V, W) with 0 and go to other leaf node labeled as 1 if any. Otherwise, we label (U, V, W) as 1 and do the following.

(ii) If $|U| = |V| = |W| = 1$, then we add one node (U, V, W) as a child of (U, V, W) and label the new node as 0. Then we go to other leaf node labeled as 1 if any.

(iii) If $|U| = 1, |V| = 1$ and $|W| > 1$, then we partition the set W into W_1 and W_2 such that $|W_1| = \left\lceil \frac{|W|}{2} \right\rceil$ and $|W_2| = \left\lfloor \frac{|W|}{2} \right\rfloor$; and we add (U, V, W_1) and (U, V, W_2) as two children of (U, V, W). The case $|U| = 1, |V| > 1$, $|W| = 1$ and $|U| > 1, |V| = 1$, $|W| = 1$ are handled similarly.

(iv) If $|U| = 1, |V| > 1$ and $|W| > 1$, then we partition the set V into V_1 and V_2 (similarly, W into W_1 and W_2) such that $|V_1| = \left\lceil \frac{|V|}{2} \right\rceil$ and $|V_2| = \left\lfloor \frac{|V|}{2} \right\rfloor$ ($|W_1| = \left\lceil \frac{|W|}{2} \right\rceil$ and $|W_2| = \left\lfloor \frac{|W|}{2} \right\rfloor$); and we add (U, V_1, W_1), (U, V_1, W_2), (U, V_2, W_1) and (U, V_2, W_2) as four children of (U, V, W). The case $|U| > 1, |V| > 1$, $|W| = 1$ and $|U| > 1, |V| = 1$ $|W| > 1$ are handled similarly.

(v) If $|U| > 1, |V| > 1$ and $|W| > 1$, then we partition the sets U, V, W into U_1 and U_2; V_1 and V_2; W_1 and W_2, respectively, such that $|U_1| = \left\lceil \frac{|U|}{2} \right\rceil$ and $|U_2| = \left\lfloor \frac{|U|}{2} \right\rfloor$; $|V_1| = \left\lceil \frac{|V|}{2} \right\rceil$ and $|V_2| = \left\lfloor \frac{|V|}{2} \right\rfloor$; $|W_1| = \left\lceil \frac{|W|}{2} \right\rceil$ and $|W_2| = \left\lfloor \frac{|W|}{2} \right\rfloor$. We add (U_1, V_1, W_1), (U_1, V_1, W_2), (U_1, V_2, W_1), (U_1, V_2, W_2) (U_2, V_1, W_1), (U_2, V_1, W_2), (U_2, V_2, W_1) and (U_2, V_2, W_2) as eight children of (U, V, W).

Let $\mathcal{T}'$ be the tree after deleting all the leaf nodes in $\mathcal{T}$. Observe that $t(A, B, C)$ is the number of leaf nodes in $\mathcal{T}'$; and

- the height of $\mathcal{T}$ is bounded by $\max\{\log|A|, \log|B|, \log|C|\} + 1 \leq 2\log n$,
- the query complexity of the above procedure is bounded by the number of nodes in $\mathcal{T}$ as we make at most one query per node of $\mathcal{T}$.

The number of nodes in $\mathcal{T}'$, equal to the number of internal nodes of $\mathcal{T}$, is bounded by $2t(A, B, C)\log n$. So, the number of leaf nodes in $\mathcal{T}$ is at most $16t(A, B, C)\log n$ and hence the total number of nodes in $\mathcal{T}$ is at most $16t(U, V, W)\log n$. Putting everything together, the required query complexity is $\mathcal{O}(t(A, B, C)\log n)$. □

Lemma 3.16 (Lemma 3.5 restated)**.** *There exists a deterministic algorithm that given any disjoint subsets $A, B, C \subset V(G)$ of any graph G and a threshold parameter $\tau \in \mathbb{N}$, can decide whether $t(A, B, C) \leq \tau$ using $\mathcal{O}(\tau \log n)$* TIS *queries.*

Proof. The algorithm proceeds similar to the one presented in the Proof of Lemma 3.6 by initializing a tree $\mathcal{T}$ with (A, B, C) as the root. If $t(A, B, C) \leq \tau$, then we can find $t(A, B, C)$ by using $16t(A, B, C) \log n$ queries and the number of nodes in $\mathcal{T}$ is bounded by $16t(A, B, C) \log n$. So, if the number of nodes in $\mathcal{T}$ is more than $16\tau \log n$ at any instance during the execution of the algorithm, we report $t(G) > \tau$ and terminate. Hence, the query complexity is bounded by the number of nodes in $\mathcal{T}$, which is $\mathcal{O}(\tau \log n)$. □

Algorithm 3.1: THRESHOLD-APPROX-TRIANGLE-ESTIMATE(G, τ, ε)

Input: A parameter τ and an $\varepsilon \in (0, 1)$.
Output: Either report $t(G) > \tau$ or find a $(1 \pm \varepsilon)$-approximation of $t(G)$.

1 **for** *($i = 1$ to $\mathcal{N} = \frac{18 \log n}{\varepsilon^2}$)* **do**
2 Partition $V(G)$ into three parts such that each vertex is present in one of A_i, B_i, C_i with probability $1/3$ independent of the other vertices.
3 Run the algorithm corresponding to Lemma 3.5 to determine if $t(A_i, B_i, C_i) > \tau$. If yes, we report $t(G) > \tau$ and QUIT. Otherwise, we have the exact vale of $t(A_i, B_i, C_i)$
4 Report $\widehat{t} = \frac{9 \sum_{i}^{\mathcal{N}} t(A_i, B_i, C_i)}{2\mathcal{N}}$ as the output.

Lemma 3.17 (Lemma 3.4 restated)**.** *There exists an algorithm that for any graph G, a threshold parameter $\tau \in \mathbb{N}$ and an $\varepsilon \in (0, 1)$, determines whether $t(G) > \tau$. If $t(G) \leq \tau$, the algorithm gives a $(1 \pm \varepsilon)$-approximation to $t(G)$ by using $\mathcal{O}(\frac{\tau \log^2 n}{\varepsilon^2})$* TIS *queries with probability at least $1 - n^{-10}$.*

Proof. We show that Algorithm 3.1 satisfies the given condition in the statement of Lemma 3.5. Note that THRESHOLD-APPROX-TRIANGLE-ESTIMATE calls the algorithm corresponding to Lemma 3.5 at most $\mathcal{N} = \frac{18 \log n}{\varepsilon^2}$ times, where each call can be executed by $\mathcal{O}(\tau \log n)$ TIS queries. So, the total query complexity of THRESHOLD-APPROX-TRIANGLE-ESTIMATE is $\mathcal{O}(\mathcal{N} \cdot \tau \log n) = \mathcal{O}\left(\frac{\tau \log^2 n}{\varepsilon^2}\right)$.

Now, we show the correctness of THRESHOLD-APPROX-TRIANGLE-ESTIMATE. If there exists an $i \in [\mathcal{N}]$, such that $t(A_i, B_i, C_i) > \tau$, then we report $t(G) > \tau$ and QUIT. Otherwise, by Lemma 3.5, we have the exact values of $t(A_i, B_i, C_i)$'s. We will be done

by showing that $\widehat{t}$ is a $(1 \pm \varepsilon)$-approximation to $t(G)$ with probability at least $1 - n^{-10}$. From the description of the algorithm, each triangle in G will be counted in $t(A_i, B_i, C_i)$ with probability $\frac{2}{9}$. We have $\mathbb{E}[t(A_i, B_i, C_i)] = \frac{2}{9}\, t(G)$, and the expectation of the sum and the estimate $\widehat{t}$ is

$$\mathbb{E}\left[\sum_{i=1}^{\mathcal{N}} t(A_i, B_i, C_i)\right] = \frac{2}{9}\,\mathcal{N} \cdot t(G) \quad \text{and} \quad \mathbb{E}[\,\widehat{t}\,] = t(G).$$

Therefore, we have

$$\mathbb{P}\left(\left|\widehat{t} - t(G)\right| \geq \varepsilon \cdot t(G)\right) = \mathbb{P}\left(\left|\sum_{i=1}^{\mathcal{N}} t(A_i, B_i, C_i) - \frac{2}{9}\,\mathcal{N} \cdot t(G)\right| \geq \frac{2\varepsilon}{9}\,\mathcal{N} \cdot t(G)\right)$$

To bound the above probability, we apply Hoeffding's inequality (See Lemma 2.3 in Section 2.1) along with the fact that $0 \leq t(A_i, B_i, C_i) \leq \tau$ for all $i \in [\mathcal{N}]$, and we get

$$\mathbb{P}\left(\left|\widehat{t} - t(G)\right| \geq \varepsilon \cdot t(G)\right) \leq \frac{1}{n^{10}}.$$

□

3.6 Coarse estimation

We now prove Lemma 3.8. Algorithm 3.3 corresponds to Lemma 3.8. Algorithm 3.2 is a subroutine in Algorithm 3.3. Algorithm 3.2 determines whether a given estimate $\widehat{t}$ is correct upto a $\mathcal{O}(\log^2 n)$ factor. Lemmas 3.18 and 3.19 are intermediate results needed to prove Lemma 3.8.

Lemma 3.18. *If* $\widehat{t} \geq 64 t(A, B, C) \log^3 n$*,*

$$\mathbb{P}(\textsc{Verify-Estimate-Triangle}\ (A, B, C, \widehat{t})\ \textit{accepts}) \leq \frac{1}{20}$$

.

Proof. Let $T(A, B, C)$ denote the set of triangles having vertices $a \in A$, $b \in B$ and $c \in$

Algorithm 3.2: VERIFY-ESTIMATE-TRIANGLE $(A, B, C, \widehat{t})$

Input: Three pairwise disjoint set $A, B, C \subseteq V(G)$ and $\widehat{t}$.
Output: If $\widehat{t}$ is a good estimate, then ACCEPT. Otherwise, REJECT.

```
1 begin
2     for (i = 2 log n to 0) do
3         Find A_i ⊆ A by sampling each element of A with probability
          min{2^i / t̂, 1}.
4         for (j = log n to 0) do
5             Set A_ij = A_i.
6             Find B_ij ⊆ B and C_ij ⊆ C by sampling each element of B and C
              with probability min{(2^j / 2^i) log n, 1} and 1/2^j, respectively.
7             if (t(A_ij, B_ij, C_ij) ≠ 0) then
8                 ACCEPT
9     REJECT
```

C, where A, B and C are disjoint subsets of $V(G)$. For $(a, b, c) \in T(A, B, C)$ such that $a \in A, b \in B, c \in C$, let $X^{ij}_{(a,b,c)}$ denote the indicator random variable such that $X^{ij}_{(a,b,c)} = 1$ if and only if $(a, b, c) \in T(A_{ij}, B_{ij}, C_{ij})$ and $X_{ij} = \sum_{(a,b,c)\in T(A,B,C)} X^{ij}_{(a,b,c)}$. Here, A_{ij}, B_{ij}, C_{ij} are as in the Algorithm 3.2. Note that the number of triangles $t(A_{ij}, B_{ij}, C_{ij}) = X_{ij}$. The triangle (a, b, c) is present in $T(A_{ij}, B_{ij}, C_{ij})$ if $a \in A_{ij}$, $b \in B_{ij}$ and $c \in C_{ij}$. So,

$$\mathbb{P}\left(X^{ij}_{(a,b,c)} = 1\right) \leq \frac{2^i}{\widehat{t}} \cdot \frac{2^j}{2^i} \log n \cdot \frac{1}{2^j} = \frac{\log n}{\widehat{t}} \text{ and } \mathbb{E}\left[X_{ij}\right] \leq \frac{t(A, B, C)}{\widehat{t}} \log n.$$

As $X_{ij} \geq 0$,

$$\mathbb{P}\left(X_{ij} \neq 0\right) = \mathbb{P}(X_{ij} \geq 1) \leq \mathbb{E}\left[X_{ij}\right] \leq \frac{t(A, B, C)}{\widehat{t}} \log n.$$

Now using the fact that $\widehat{t} \geq 64t(A, B, C) \log^3 n$, we have $\mathbb{P}\left(X_{ij} \neq 0\right) \leq \frac{1}{64 \log^2 n}$. Observe that VERIFY-ESTIMATE-TRIANGLE accepts if and only if there exists $i, j \in$

$\{0, \ldots, \log n\}$ such that $X_{ij} \neq 0$. Using the union bound, we get

$$\begin{aligned}
\mathbb{P}\left(\text{VERIFY-ESTIMATE-TRIANGLE accepts}\right) &\leq \sum_{0 \leq i \leq 2\log n} \sum_{0 \leq j \leq \log n} \mathbb{P}\left(X_{ij} \neq 0\right) \\
&\leq \frac{(2\log n + 1)(\log n + 1)}{32 \log^2 n} \\
&\leq \frac{1}{20}.
\end{aligned}$$

□

Lemma 3.19. *If* $\widehat{t} \leq \frac{t(A,B,C)}{32 \log n}$,

$$\mathbb{P}(\text{VERIFY-ESTIMATE-TRIANGLE } (A, B, C, \widehat{t}) \text{ accepts}) \geq \frac{1}{5}.$$

Proof. For $p \in \{0, \ldots, 2\log n\}$, let $A^p \subseteq A$ be the set of vertices such that for each $a \in A_p$, the number of triangles of the form (a, b, c) with $(b, c) \in B \times C$, lies between 2^p and $2^{p+1} - 1$.

For $a \in A^p$ and $q \in \{0, \ldots, \log n\}$, let $B^{pq}(a) \subseteq B$ is the set of vertices such that for each $b \in B$, the number of triangles of the form (a, b, c) with $c \in C$ lies between 2^q and $2^{q+1} - 1$ We need the following Claim to proceed further.

Claim 3.20. *(i) There exists* $p \in \{0, \ldots, 2\log n\}$ *such that* $|A^p| > \frac{t(A,B,C)}{2^{p+1}(2\log n+1)}$.

(ii) For each $a \in A^p$*, there exists* $q \in \{0, \ldots, \log n\}$ *such that* $|B^{pq}(a)| > \frac{2^p}{2^{q+1}(\log n+1)}$.

Proof. (i) Observe that $t(A, B, C) = \sum_{p=0}^{2\log n} t(A^p, B, C)$ as the sum takes into account all incidences of vertices in A. So, there exists $p \in \{0, \ldots, 2\log n\}$ such that $t(A^p, B, C) \geq \frac{t(A,B,C)}{2\log n+1}$. From the definition of A^p, $t(A^p, B, C) < |A^p| \cdot 2^{p+1}$. Hence, there exists $p \in \{0, \ldots, 2\log n\}$ such that

$$|A^p| > \frac{t(A^p, B, C)}{2^{p+1}} \geq \frac{t(A, B, C)}{2^{p+1}(2\log n + 1)}.$$

(ii) Observe that $\sum_{q=0}^{\log n} t(\{a\}, B^{pq}(a), C) = t(\{a\}, B, C)$. So, there exists $q \in \{0,$

$\ldots, \log n\}$ such that $t(\{a\}, B^{pq}(a), C) \geq \frac{t(\{a\},B,C)}{\log n+1}$. From the definition of $B^{pq}(a)$, $t(\{a\}, B^{pq}(a), C) < |B^{pq}(a)| \cdot 2^{q+1}$. Hence, there exists $q \in \{0, \ldots, \log n\}$ such that

$$|B^{pq}(a)| > \frac{t(\{a\}, B^{pq}(a), C)}{2^{q+1}} \geq \frac{t(\{a\}, B, C)}{2^{q+1}(\log n + 1)} \geq \frac{2^p}{2^{q+1}(\log n + 1)}.$$

□

We come back to the proof of Lemma 3.19. We will show that VERIFY-ESTIMATE-TRIANGLE accepts with probability at least $\frac{1}{5}$ when loop executes for $i = p$, where p is such that $|A^p| > \frac{t(A,B,C)}{2^{p+1}(2\log n+1)}$. The existence of such a p is evident from Claim 3.20 (i).

Recall that $A_{pq} \subseteq A$, $B_{pq} \subseteq B$ and $C_{pq} \subseteq C$ are the samples obtained when the loop variables i and j in Algorithm 3.2 attain values p and q, respectively. Observe that

$$\mathbb{P}\left(A_{pq} \cap A^p = \emptyset\right) \leq \left(1 - \frac{2^p}{\widehat{t}}\right)^{|A^p|} \leq e^{-\frac{2^p}{\widehat{t}}|A^p|} \leq e^{-\frac{2^p}{\widehat{t}}\frac{t(A,B,C)}{2^{p+1}\log n}} = e^{-\frac{t(A,B,C)}{2\widehat{t}(2\log n+1)}}.$$

Now using the fact that $\widehat{t} \leq \frac{t(A,B,C)}{32\log n}$ and $n \geq 64$,

$$\mathbb{P}\left(A_{pq} \cap A^p = \emptyset\right) \leq \frac{1}{e^6}.$$

Assume that $A_{pq} \cap A^p \neq \emptyset$ and $a \in A_{pq} \cap A^p$. By Claim 3.20 (ii), there exists $q \in \{0, \ldots, \log n\}$, such that $B^{pq}(a) \geq \frac{2^p}{2^{q+1}(\log n+1)}$. Note that q depends on a. Observe that we will be done, if we can show that VERIFY-ESTIMATE-TRIANGLE accepts when loop executes for $i = p$ and $j = q$. Now,

$$\mathbb{P}\left(B_{pq} \cap B^{pq}(a) = \emptyset \mid A_{pq} \cap A^p \neq \emptyset\right) \leq \left(1 - \frac{2^q}{2^p}\log n\right)^{|B^{pq}(a)|} \leq \frac{1}{e^{3/7}}.$$

Assume that $A_{pq} \cap A^p \neq \emptyset$, $B_{pq} \cap B^{pq}(a) \neq \emptyset$ and $b \in B_{pq} \cap B^{pq}(a)$. Let S be the set such that (a, b, s) is a triangle in G for each $s \in S$. Note that $|S| \geq 2^q$. So,

$$\mathbb{P}\left(C_{pq} \cap S = \emptyset \mid A_{pq} \cap A^p \neq \emptyset \text{ and } B_{pq} \cap B^{pq}(a) \neq \emptyset\right) \leq \left(1 - \frac{1}{2^q}\right)^{2^q} \leq \frac{1}{e}.$$

Observe that VERIFY-ESTIMATE-TRIANGLE accepts if $t(A_{pq}, B_{pq}, C_{pq}) \neq 0$. Also, $t(A_{pq}, B_{pq}, C_{pq}) \neq 0$ if $A_{pq} \cap A^p \neq \emptyset$, $B_{pq} \cap B^{pq}(a) \neq \emptyset$ and $C_{pq} \cap S \neq \emptyset$. Hence,

$$\begin{aligned}
& \mathbb{P}(\text{VERIFY-ESTIMATE-TRIANGLE accepts}) \\
\geq\ & \mathbb{P}\left(A_{pq} \cap A^p \neq \emptyset,\, B_{pq} \cap B^{pq}(a) \neq \emptyset \text{ and } C_{pq} \cap S \neq \emptyset\right) \\
=\ & \mathbb{P}\left(A_{pq} \cap A^p \neq \emptyset\right) \cdot \mathbb{P}\left(B_{pq} \cap B^{pq}(a) \neq \emptyset \mid A_{pq} \cap A^p \neq \emptyset\right) \\
& \cdot \mathbb{P}\left(C_{pq} \cap S \neq \emptyset \mid A_{pq} \cap A^p \neq \emptyset \text{ and } B_{pq} \cap B^{pq}(a) \neq \emptyset\right) \\
>\ & \left(1 - \frac{1}{e^6}\right)\left(1 - \frac{1}{e^{3/7}}\right)\left(1 - \frac{1}{e}\right) > \frac{1}{5}.
\end{aligned}$$

□

Algorithm 3.3: COARSE-ESTIMATE-TRIANGLE (A, B, C)

Input: Three pairwise disjoint sets $A, B, C \subset V(G)$.
Output: An estimate $\widehat{t}$ for $t(A, B, C)$.

```
begin
    for ( t̂ = n^3, n^3/2, ..., 1) do
        Repeat VERIFY-ESTIMATE-TRIANGLE (A, B, C, t̂) for Γ = 2000 log n
          times. If at least Γ/10 runs of VERIFY-ESTIMATE-TRIANGLE accepts,
          then output t̃ = t̂/log n.
```

Lemma 3.21 (Lemma 3.8 restated)**.** *There exists an algorithm that given disjoint subsets $A, B, C \subset V(G)$ of any graph G, returns an estimate $\widetilde{t}$ satisfying*

$$\frac{t(A,B,C)}{64\ \log^2 n} \leq \widetilde{t} \leq 64\, t(A,B,C)\ \log^2 n$$

with probability at least $1 - n^{-9}$. Moreover, the query complexity of the algorithm is $\mathcal{O}(\log^4 n)$.

Proof. Note that an execution of COARSE-ESTIMATE-TRIANGLE for a particular $\widehat{t}$, repeats VERIFY-ESTIMATE-TRIANGLE for $\Gamma = 2000 \log n$ times and gives output $\widehat{t}$ if at least $\frac{\Gamma}{10}$ runs of VERIFY-ESTIMATE-TRIANGLE accepts. For a particular $\widehat{t}$, let

X_i be the indicator random variable such that $X_i = 1$ if and only if the i^{th} execution of VERIFY-ESTIMATE-TRIANGLE accepts. Also take $X = \sum_{i=1}^{\Gamma} X_i$. COARSE-ESTIMATE-TRIANGLE gives output $\widehat{t}$ if $X > \frac{\Gamma}{10}$.

Consider the execution of COARSE-ESTIMATE-TRIANGLE for a particular $\widehat{t}$. If $\widehat{t} \geq 32t(A,B,C)\log^3 n$, we first show that COARSE-ESTIMATE-TRIANGLE accepts with probability at least $1-\frac{1}{n^5}$. Recall Lemma 3.18. If $\widehat{t} \geq 64t(A,B,C)\log^3 n$, $\mathbb{P}(X_i = 1) \leq \frac{1}{20}$ and hence $\mathbb{E}[X] \leq \frac{\Gamma}{20}$. By using Chernoff-Hoeffding's inequality (See Lemma 2.4 (i) in Section 2.1),

$$\mathbb{P}\left(X > \frac{\Gamma}{10}\right) = \mathbb{P}\left(X > \frac{\Gamma}{20} + \frac{\Gamma}{20}\right) \leq \frac{1}{n^{10}}.$$

By using the union bound for all $\widehat{t}$, the probability that COARSE-ESTIMATE-TRIANGLE outputs some $\widetilde{t} = \frac{\widehat{t}}{\log n} \geq 16t(A,B,C)\log^2 n$, is at most $\frac{3\log n}{n^{10}}$.

Now consider the instance when the for loop in COARSE-ESTIMATE-TRIANGLE executes for a $\widehat{t}$ such that $\widehat{t} \leq \frac{t(A,B,C)}{32\log n}$. In this situation, $\mathbb{P}(X_i = 1) \geq \frac{1}{5}$. So, $\mathbb{E}[X] \geq \frac{\Gamma}{5}$. By using Chernoff-Hoeffding's inequality (Lemma 2.4 (ii) in Section 2.1),

$$\mathbb{P}\left(X \leq \frac{\Gamma}{10}\right) \leq \mathbb{P}\left(X < \frac{3\Gamma}{20}\right) = \mathbb{P}\left(X < \frac{\Gamma}{5} - \frac{\Gamma}{20}\right) \leq \frac{1}{n^{10}}.$$

By using the union bound for all $\widehat{t}$, the probability that COARSE-ESTIMATE-TRIANGLE outputs some $\widetilde{t} = \frac{\widehat{t}}{\log n} \leq \frac{t(A,B,C)}{16\log^2 n}$, is at most $\frac{3\log n}{n^{10}}$.

Observe that, the probability of COARSE-ESTIMATE-TRIANGLE giving an output of $\widetilde{t}$, such that $\widetilde{t} \notin \left[\frac{t(A,B,C)}{32\ \log^2 n}, 64\ t(A,B,C)\log^2 n\right]$, is at most $\frac{3\log n}{n^{10}} + \frac{3\log n}{n^{10}} \leq \frac{1}{n^9}$.

Putting everything together, COARSE-ESTIMATE-TRIANGLE gives some $\widetilde{t}$ as output with probability at least $1 - \frac{1}{n^9}$ satisfying

$$\frac{t(A,B,C)}{64\ \log^2 n} \leq \widetilde{t} \leq 64\ t(A,B,C)\ \log^2 n.$$

From the description of VERIFY-ESTIMATE-TRIANGLE and COARSE-ESTIMATE-TRIANGLE, the query complexity of VERIFY-ESTIMATE-TRIANGLE is $\mathcal{O}(\log^2 n)$ and COARSE-ESTIMATE-TRIANGLE calls VERIFY-ESTIMATE-TRIANGLE $\mathcal{O}(\log^2 n)$ times.

Hence, COARSE-ESTIMATE-TRIANGLE makes $\mathcal{O}(\log^4 n)$ queries. □

3.7 The final triangle estimation algorithm: Proof of Theorem 3.2

Now we design an algorithm for a $(1 \pm \varepsilon)$-multiplicative approximation of $t(G)$. If $\varepsilon \leq \frac{\sqrt{\wp}\log^{9/2} n}{n^{3/4}}$, we query for $t(\{a\},\{b\},\{c\})$ for all distinct $a, b, c \in V(G)$ and compute the exact value of $t(G)$. So, we assume that $\varepsilon > \frac{\sqrt{\wp}\log^{9/2} n}{n^{3/4}}$.

We build a data structure such that it maintains two things at any point of time.

(i) An accumulator ψ for the number of triangles. We initialize $\psi = 0$.

(ii) A set of tuples $(A_1, B_1, C_1, w_1), \ldots, (A_\zeta, B_\zeta, C_\zeta, w_\zeta)$, where tuple (A_i, B_i, C_i) corresponds to the tripartite subgraph $G(A_i, B_i, C_i)$ and w_i is the weight associated to $G(A_i, B_i, C_i)$. Initially, there is no tuple in our data structure.

Before discussing the steps of our algorithm, some remarks about our sparsification lemmas (Lemmas 3.3 and 3.7) are in order.

Remark 3.2. (i) In Lemma 3.3, $\frac{9k^2}{2}\sum_{i=1}^{k} t(V_i, V_{k+i}, V_{2k+i})$ is a $(1 \pm \lambda)$-approximation of $t(G)$ when

$$\kappa_1 \wp k^2 \sqrt{t(G)} \log n \leq \lambda\, t(G) \Leftrightarrow t(G) \geq \frac{\kappa_1^2 \wp^2 k^4 \log^2 n}{\lambda^2}.$$

In our algorithm, we apply Lemma 3.3 for $k = 1$. Also, we require $\lambda = \frac{\varepsilon}{6\log n}$. So, Lemma 3.3 gives useful result in our algorithm when $t(G) \geq \frac{36\kappa_1^2 \wp^2 \log^4 n}{\varepsilon^2}$.

(ii) In Lemma 3.7, $k^2 \sum_{i=1}^{k} t(V_i, V_{k+i}, V_{2k+i})$ is a $(1 \pm \lambda)$-approximation of $t(A, B, C)$ when

$$\kappa_2 \wp k^2 \sqrt{t(G)} \log n \leq \lambda t(G) \Leftrightarrow t(G) \geq \frac{\kappa_2^2 \wp^2 k^4 \log^2 n}{\lambda^2}.$$

In our algorithm, we apply Lemma 3.7 for $k = 3$. Also, we will require $\theta = \frac{\varepsilon}{6 \log n}$. So, the above sparsification lemma gives useful result in our algorithm when $t(A, B, C) \geq \frac{324\kappa_2^2 \wp^2 \log^4 n}{\varepsilon^2}$.

The algorithm sets a threshold $\tau = \max\left\{\frac{36\kappa_1^2 \wp^2 \log^4 n}{\varepsilon^2}, \frac{324\kappa_2^2 \wp^2 \log^4 n}{\varepsilon^2}\right\}$ and will proceed as follows:

Step 1: (Threshold-Approx-Triangle-Estimation) Run the algorithm THRESHOLD-APPROX-TRIANGLE-ESTIMATION, presented in Section 3.5 with parameters τ and ε. By Lemma 3.4, we either decide $t(G) > \tau$ or we have $\widehat{t}$ which is a $(1 \pm \varepsilon)$-approximation to $t(G)$. If $t(G) > \tau$, we go to Step 2. Otherwise, we terminate by reporting an estimate $\widehat{t}$. The query complexity of Step 1 is $\mathcal{O}\left(\frac{\tau \log^2 n}{\varepsilon^2}\right)$.

Step 2: (General Sparsification) $V(G)$ is COLORED with $[3k]$ for $k = 1$. Let A, B, C be the partition generated by the coloring of $V(G)$. We initialize the data structure by setting $\psi = 0$ and adding the tuple $(A, B, C, 9/2)$ to the data structure. Note that no query is required in this step. The constant $9/2$ is obtained by putting $k = 1$ in Lemma 3.3.

Step 3: We repeat Steps 4 to 7 until there is no tuple left in the data structure. We maintain an invariant that the number of tuples stored in the data structure, is $\mathcal{O}(N)$, where $N = \frac{\kappa_3 \log^{12} n}{\varepsilon^2}$. Note that κ_3 is a constant to be fixed later.

Step 4: (Threshold for Tripartite Graph and Exact Counting in Tripartite Graphs) For each tuple (A, B, C, w) in the data structure, we determine whether $t(A, B, C) \leq \tau$, the threshold, by using the deterministic algorithm corresponding to Lemma 3.5 with $\mathcal{O}(\tau \log n)$ queries. If yes, we find the exact value of $t(A, B, C)$ by using the deterministic algorithm corresponding to Lemma 3.6 with $\mathcal{O}(\tau \log n)$ queries. Then the algorithm adds $w \cdot t(A, B, C)$ to ψ. We remove all (A, B, C)'s for which the algorithm found that $t(A, B, C)$ is below the threshold. As there are $\mathcal{O}(N)$ triples at any time, the number of queries made in each iteration of the algorithm is $\mathcal{O}(\tau \log n \cdot N) = \mathcal{O}(\tau N \log n)$.

Step 5: Note that each tuple (A, B, C, w) in this step is such that $t(A, B, C) > \tau$. Let $(A_1, B_1, C_1, w_1), \ldots, (A_r, B_r, C_r, w_r)$ be the set of tuples stored at the current instant. If $r > 10N$ [8], we go to Step 6. Otherwise, we go to Step 7.

Step 6 (Coarse Estimation and Sampling) For each tuple $(A,\ B,\ C,\ w)$ in the data structure, we find an estimate $\widetilde{t}$ such that $\frac{t(A,B,C)}{64\log^2 n} < \widetilde{t} < 64t(A,\ B,\ C)\log^2 n$. This can be done due to Lemma 3.8 and the number of queries is $\mathcal{O}\left(\log^4 n\right)$ per tuple. As the algorithm executes the current step, the number of tuples in our data structure is more than $10N$. We take a sample from the set of tuples such that the sample maintains the required estimate *approximately* by using Lemma 3.9. We use the algorithm corresponding to Lemma 3.9 with $\lambda = \frac{\varepsilon}{6\log n}$, $\rho = 64\log^2 n$ and $\delta = \frac{1}{n^{10}}$ to find a new set of tuples $(A'_1,\ B'_1,\ C'_1,\ w'_1), \ldots, (A'_s,\ B'_s,\ C'_s,\ w'_s)$ such that

$$\left| S - \sum_{i=1}^{s} w'_i\, t(A',\ B',\ C') \right| \leq \lambda S$$

with probability $1 - \frac{1}{n^{10}}$, where $S = \sum_{i=1}^{r} w_i t(A_i,\ B_i,\ C_i)$ and $s = \frac{\kappa_3 \log^{12} n}{\varepsilon^2}$ [9] for some constant $\kappa_3 > 0$. This κ_3 is same as the one mentioned in Step 3. Also, note that, $N = s = \frac{\kappa_3 \log^{12} n}{\varepsilon^2}$. No query is required to execute the algorithm of Lemma 3.9. Recall that the number of tuples present at any time is $\mathcal{O}(N)$. Also, the coarse estimation for each tuple can be done by using $\mathcal{O}(\log^4 n)$ queries (Lemma 3.8). Hence, the number of queries in this step in each iteration, is $\mathcal{O}(N \cdot \log^4 n)$.

Step 7: (Sparsification for Tripartite Graphs) We partition each of $A,\ B$ and C into 3 parts uniformly at random. Let $A = U_1 \uplus U_2 \uplus U_3$; $V = V_1 \uplus V_2 \uplus V_3$ and $W = W_1 \uplus W_2 \uplus W_3$. We delete (A, B, C, w) from the data structure and add $(U_i, V_i, W_i, 9w)$ for each $i \in [3]$ to our data structure. Note that no query is made in this step.

Step 8: Report ψ as the estimate for the number of triangles in G, when no tuples are

[8] The constant 10 is arbitrary. Any absolute constant more than 1 would have been good enough.
[9] s is set according to Lemma 3.9.

left.

First, we prove that the above algorithm produces a $(1\pm\varepsilon)$ multiplicative approximation to $t(G)$ for any $\varepsilon > 0$ with high probability. Recall the description of Step 1 of the algorithm. If the algorithm terminates in Step 1, then we have a $(1 \pm \varepsilon)$ approximation to $t(G)$ by Lemma 3.4. Otherwise, we decide that $t(G) > \tau$ and proceed to Step 2. In Step 2, the algorithm colors $V(G)$ using three colors and incurs a multiplicative error of $1 \pm \varepsilon_0$ to $t(G)$, where $\varepsilon_0 = \frac{\kappa_1 \wp \log n}{\sqrt{t(G)}}$. This is because of Remark 3.2 and our choice of τ. As $t(G) > \tau$ and $n \geq 64$, $\varepsilon_0 \leq \lambda = \frac{\varepsilon}{6 \log n}$. Note that the algorithm possibly performs Step 4 to Step 7 multiple times, but not more than $O(\log n)$ times, as explained below.

Let $(A_1,\, B_1,\, C_1,\, w_1), \ldots, (A_\zeta,\, B_\zeta,\, C_\zeta,\, w_\zeta)$ are the set of tuples present in the data structure currently. We define $\sum_{i=1}^{\zeta} t(A_i,\, B_i,\, C_i)$ as the number of *active* triangles. Let $\textsc{Act}_i$ be the number of triangles that are active in the i^{th} iteration. Note that $\textsc{Act}_1 \leq t(G) \leq n^3$. By Lemma 3.7 and Step 7, observe that $\textsc{Act}_{i+1} \leq \frac{\textsc{Act}_i}{2}$. So, after $3 \log n$ iterations there will be at most constant number of active triangles and then we can compute the exact number of active triangles and add it to ψ. In each iteration, there can be a multiplicative error of $1 \pm \lambda$ in Step 5 and $1 \pm \varepsilon_0$ due to Step 4. So, using the fact that $\varepsilon_0 \leq \lambda$, the multiplicative approximation factor lies between $(1-\lambda)^{3 \log n + 1}$ and $(1+\lambda)^{3 \log n + 1}$. As $\lambda = \frac{\varepsilon}{6 \log n}$, the required approximation factor is $1 \pm \varepsilon$.

The query complexity of Step 1 is $\mathcal{O}\left(\frac{\tau \log n}{\varepsilon^2}\right)$. Steps 2, 3, 5, 7 and 8 do not make any query to the oracle. The query complexity of Step 4 is $\mathcal{O}\left(\tau N \log n\right)$ in each iteration and that of Step 6 is $\mathcal{O}(N \log^4 n)$ in each iteration. The total number of iterations is $\mathcal{O}(\log n)$. Hence, the total query complexity of the algorithm is

$$\mathcal{O}\left(\varepsilon^{-2}\tau \log n + (\tau \log n + \tau N \log n + N \log^4 n) \log n\right) = \mathcal{O}(\varepsilon^{-4}\wp^2 \log^{18} n).$$

In the above expression, we have put $\tau = \max\left\{\frac{36\kappa_1^2\wp^2 \log^4 n}{\varepsilon^2}, \frac{324\kappa_2^2\wp^2 \log^4 n}{\varepsilon^2}\right\}$ and $N = \frac{\kappa_3 \log^{12} n}{\varepsilon^2}$.

Now, we bound the failure probability of the algorithm. The algorithm can fail in Step 1 with probability at most $\frac{1}{n^{10}}$, Step 2 with probability at most $\frac{2}{n^4}$, Step 6 with probability at most $\frac{10\kappa_3 \log^{12} n}{\varepsilon^4} \cdot \frac{1}{n^9} + \frac{1}{n^{10}}$, and Step 7 with probability at most $\frac{10\kappa_3 \log^{12} n}{\varepsilon^4} \cdot \frac{1}{n^8}$.

As the algorithm might execute Steps 4 to 6 for $3 \log n$ times, the total failure probability is bounded by

$$\frac{1}{n^{10}} + \frac{2}{n^4} + 3 \log n \left(\frac{10\kappa_3 \log^{12} n}{\varepsilon^4} \cdot \frac{1}{n^8} + \frac{10\kappa_3 \log^{12} n}{\varepsilon^4} \cdot \frac{1}{n^9} + \frac{1}{n^{10}} \right) \leq \frac{c}{n^2}.$$

Note that the above inequality holds because $\varepsilon > \frac{\sqrt{\wp} \log^{9/2} n}{n^{3/4}}$ and $n \geq 64$.

We end this Section by restating our main result.

Theorem 3.22 (Restatement of Theorem 3.2)**.** *Let G be a graph with $\Delta_E \leq \wp$, $|V(G)| = n \geq 64$. For any $\varepsilon > 0$,* TRIANGLE-ESTIMATION *can be solved using $\mathcal{O}\left(\frac{\wp^2 \log^{18} n}{\varepsilon^4}\right)$* TIS *queries with probability at least $1 - \frac{\mathcal{O}(1)}{n^2}$.*

3.8 Discussion

In this work, we generalize the framework of Beame et al [BHR+18] of EDGE ESTIMATION to solve TRIANGLE-ESTIMATION by using TIS queries. Our algorithm makes $O(\varepsilon^{-4} \wp^2 \log^{18} n)$ TIS queries and returns a $(1 \pm \varepsilon)$-approximation to the number of triangles with high probability, where $\wp$ is the upper bound on Δ_E. The downside of our work is the assumption $\Delta_E \leq \wp$. Note that Beame et al. [BHR+18] had no such assumption. Removing the assumption is non-trivial mainly due to the fact that, unlike the case for edges where two edges can share a common vertex, two triangles can share an edge. Our sparsification algorithm crucially uses the assumption on Δ_E and that remains the main barrier to cross. Recall our sparsification lemma (Lemma 3.3) and the definition of properly colored triangles (Definition 3.11). Roughly speaking, our sparsification algorithm first colors the vertices of the graph, then counts the number of properly colored triangles, and finally scales it to have an estimation of the total number of triangles in the graph. Consider the situation when all the triangles in the graph have a common edge e. If e is not properly colored, then we can not keep track of any triangle in G. As a follow up to this work, Dell et al. [DLM20b] and Bhattacharya et al. [BBGM19a] [10], inde-

[10] This work is to be discussed in Chapter 4.

pendently, generalized our result to d-uniform hypergraphs, where $d \in \mathbb{N}$ is a constant. Note that TRIANGLE-ESTIMATION can also be thought of as HYPEREDGE ESTIMATION problem in a 3-uniform hypergrah. Their results showed that the bound on Δ_E is not necessary to solve TRIANGLE-ESTIMATION by using polylogarithmically many TIS queries. The main technical result in both the works is to come up with a sparsification algorithm that can take care of the case when Δ_E is not necessarily bounded. Note the sparsification algorithms in both the works are completely different and give different insights.

Bhattacharya et al. [BBGM19a] and Dell et al. [DLM20b] refer the generalized oracle as GENERALISED PARTITE INDEPENDENT SET (GPIS) oracle and COLORFUL DECISION (CD) oracle, respectively. Bhattacharya et al. [BBGM19a] showed that HYPEREDGE ESTIMATION can be solved by using $\mathcal{O}_d\left(\varepsilon^{-4}\log^{5d+5} n\right)$ GPIS queries and Dell et al. [DLM20b] showed that it can be solved by using $\mathcal{O}_d\left(\varepsilon^{-2}\log^{4d+8} n\right)$ CD queries [11], with high probability. Substituting $d = 3$ in their algorithm, we can have two different algorithms for TRIANGLE-ESTIMATION. Let us compare our result (stated in Theorem 3.22) with the results of [BBGM19a] and Dell et al. [DLM20b] in the context of TRIANGLE-ESTIMATION. If $\Delta_E = o(\log n)$, our algorithm for TRIANGLE-ESTIMATION have less query complexity than that of Bhattacharya et al. [BBGM19a] for any given $\varepsilon > 0$. Also, when $\Delta_E = o(\log n)$ and $\varepsilon > 0$ is a fixed constant, our algorithm for TRIANGLE-ESTIMATION has smaller query complexity than that of Dell et al. [DLM20b].

[11] The constant in $\mathcal{O}_c(\cdot)$ is a function of d. The result of Bhattacharya et al. is a high probability result. The exact bound in the paper of Dell et al. is $\mathcal{O}_d\left(\varepsilon^{-2}\log^{4d+7} n \log\frac{1}{\delta}\right)$, where the probability of success of their algorithm is $1-\delta$.

Chapter 4

Hyperedge Estimation Using GPIS Queries

Contents

4.1 Brief description of the problem

In Chapter 3, we discussed the problem of TRIANGLE ESTIMATION in a graph when we have TRIPARTITE INDEPENDENT SET (TIS) oracle access to the graph. Here we generalize the problem of TRIANGLE ESTIMATION in a graph using TIS oracle to HYPEREDGE-ESTIMATION problem in a d-uniform hypergraph using GENERALIZED PARTITE INDEPENDENT SET oracle, a *suitable* generalization of TIS.

Recall that a hypergraph $\mathcal{H}$ is a *set system* $(U(\mathcal{H}), \mathcal{F}(\mathcal{H}))$, where $U(\mathcal{H})$ denotes a set of n vertices and $\mathcal{F}(\mathcal{H})$, a set of subsets of $U(\mathcal{H})$, denotes the set of hyperedges. A hypergraph $\mathcal{H}$ is said to be d-uniform if every hyperedge in $\mathcal{H}$ consists of exactly d vertices. The cardinality of the hyperedge set is denoted as $m(\mathcal{H}) = |\mathcal{F}(\mathcal{H})|$. Also, recall the formal definition of GPIS oracle.

Definition 4.1. Generalized d-partite independent set oracle (GPIS) [BGK+18a]: Given d pairwise disjoint subsets of vertices $A_1, \ldots, A_d \subseteq U(\mathcal{H})$ of a hypergraph $\mathcal{H}$ as input, GPIS query oracle answers YES if and only if $m(A_1, \ldots, A_d) \neq 0$, where $m(A_1, \ldots, A_d)$ denotes the number of hyperedges in $\mathcal{H}$ having exactly one vertex in each A_i, $\forall i \in \{1, 2, \ldots, d\}$.

We now state the precise problem that we solve in the GPIS oracle framework and present our main result in Theorem 4.2.

HYPEREDGE-ESTIMATION
Input: A set of n vertices $U(\mathcal{H})$ of a hypergraph $\mathcal{H}$, a GPIS oracle access to $\mathcal{H}$, and $\varepsilon \in (0, 1)$.
Output: An estimate $\widehat{m}$ of $m(\mathcal{H})$ such that $(1 - \epsilon) \cdot m(\mathcal{H}) \leq \widehat{m} \leq (1 - \epsilon) \cdot m(\mathcal{H})$.

Theorem 4.2. *Let $\mathcal{H}$ be a hypergraph with $|U(\mathcal{H})| = n$. For any $\varepsilon \in (0, 1)$,* HYPEREDGE-ESTIMATION *can be solved using $\mathcal{O}_d\left(\frac{\log^{5d+5} n}{\varepsilon^4}\right)$* GPIS *queries with high probability[1], where the constant in $\mathcal{O}_d(\cdot)$ is a function of d.*

[1] high probability means a probability of at least $1 - n^{-O(1)}$

Here, we note that concurrently and independently, Dell et al. [DLM19] obtained polylogarithmic query complexity for the hyperedge estimation problem using a similar oracle. They called the query oracle a *colorful decision oracle*. We will discuss their result shortly.

Beame *et al.* [BHR+18] used a subset query oracle, named BIPARTITE INDEPENDENT SET (BIS) query oracle to estimate the number of edges in a graph using polylogarithmic query complexity[2]. The BIS query oracle answers a YES/NO question on the existence of an edge between two disjoint subsets of vertices of a graph G. Having estimated the number of edges in a graph using BIS queries, a very natural question was to estimate the number of hyperedges in a hypergraph using an appropriate query oracle. The answer to the above question is not obvious as two edges in a graph can intersect in at most one vertex but the intersection between two hyperedges in a hypergraph can be an arbitrary set. As a first step towards resolving this generalized question, Bhattacharya et al. [BBGM18,BBGM19b] considered the hyperedge estimation problem using a TRIPARTITE INDEPENDENT SET (TIS) oracle in 3-uniform hypergraphs. Recall that a TIS query oracle takes three disjoint subsets of vertices as input and reports whether there exists a hyperedge having a vertex in each of the three sets. It was shown that when the number of hyperedges having two vertices in common is bounded above (polylogarithmic in n), then the number of hyperedges in a 3-uniform hypergraph can be estimated using polylogarithmic TIS queries. This leads us to ask the next set of questions given as follows.

- **Question 1***: For a 3-uniform hypergraph, is the dependence of the* TIS *query complexity on the number of hyperedges with two common vertices inherent as in Bhattacharya et al. [BBGM18, BBGM19b]?*
- **Question 2***: Can the subset query oracle framework of Beame et al. be extended to estimate the number of hyperedges in a d-uniform hypergraph using only polylogarithmically many queries?*

In this work, we give positive answers to both these questions. We show that the

[2]query complexity means the number of queries used by the corresponding query oracle

number of hyperedges in a d-uniform hypergraph can be estimated using polylogarithmically [3] many GPIS queries.

Setup and notations

We denote the sets $\{1, \ldots, n\}$ and $\{0, \ldots, n\}$ by $[n]$ and $[n^*]$, respectively. A hypergraph $\mathcal{H}$ is a *set system* $(U(\mathcal{H}), \mathcal{F}(\mathcal{H}))$, where $U(\mathcal{H})$ denotes the set of vertices and $\mathcal{F}(\mathcal{H})$ denotes the set of hyperedges. The set of vertices present in a hyperedge $F \in \mathcal{F}(\mathcal{H})$ is denoted by $U(F)$ or simply F. A hypergraph $\mathcal{H}$ is said to be d-uniform if all the hyperedges in $\mathcal{H}$ consist of exactly d vertices. The cardinality of the hyperedge set is $m(\mathcal{H}) = |\mathcal{F}(\mathcal{H})|$. For $u \in U(\mathcal{H})$, $\mathcal{F}(u)$ denote the set of hyperedges that are incident on u. For $u \in U(\mathcal{H})$, the degree of u in $\mathcal{H}$, denoted as $\deg_{\mathcal{H}}(u) = |\mathcal{F}(u)|$ is the number of hyperedges incident on u. For a set A and $a \in \mathbb{N}$, $A, \ldots, A$ (a times) will be denoted as $A^{[a]}$. Let $A_1, \ldots, A_d \subseteq U(\mathcal{H})$ be such that for every $i, j \in [d]$ either $A_i = A_j$ or $A_i \cap A_j = \emptyset$. This has a bearing on the GPIS oracle queries we make; either the sets we query with are disjoint, or are the same. Consider the following d-partite sub-hypergraph of $\mathcal{H}$: $(U(A_1, \ldots, A_d), \mathcal{F}(A_1, \ldots, A_d))$ where the vertex set is $U(A_1, \ldots, A_d) = \bigcup_{i=1}^{d} A_i$ and the hyperedge set is $\mathcal{F}(A_1, \ldots, A_d) = \{\{i_1, \ldots, i_d\} \mid i_j \in A_j\}$; we will denote this d-partite sub-hypergraph of $\mathcal{H}$ as $\mathcal{H}(A_1, \ldots, A_d)$. With this notation, $\mathcal{H}\left(U^{[d]}\right)$ makes sense as a d-partite sub-hypergraph on a vertex set U. The number of hyperedges in $\mathcal{H}(A_1, \ldots, A_d)$ is denoted by $m(A_1, \ldots, A_d)$.

Ordered hyperedge We will use the subscript o to denote the set of ordered hyperedges. For example, $\mathcal{F}_o(\mathcal{H})$ denotes the set of ordered hyperedges, $m_o(\mathcal{H})$ denote $|\mathcal{F}_o(\mathcal{H})|$, $\mathcal{F}_o(u)$ denote the set of ordered hyperedges incident on u. The ordered hyperedge set puts an order on the vertices such that i-th vertex of a hyperedge comes from A_i. Formally, $\mathcal{F}_o(A_1, \ldots, A_d) = \{F_o \in \mathcal{F}_o(\mathcal{H}) : \text{the } i\text{-th vertex of } F_o \text{ is in } A_i, \forall i \in [d]\}$. The corresponding number for ordered hyperedges is $m_o(A_1, \ldots, A_d)$. We have the following relation between $m(A_1, \ldots, A_d)$ and $m_o(A_1, \ldots, A_d)$.

[3] Here the exponent of $\log n$ is $\mathcal{O}(d)$, but the exponent of ε is an absolute constant.

Fact 4.3. *For* $s \in [d]$, $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) = m(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \times \prod_{i=1}^{s} a_i!$, *where* $a_i \in [d]$ *such that* $\sum_{i=1}^{s} a_i = d$ *, and* $A_1, \ldots, A_s$ *are pairwise disjoint sets.*

For a set $\mathcal{P}$, "$\mathcal{P}$ is COLORED with [n]" means that elements of $\mathcal{P}$ is assigned a color out of $[n]$ colors independently and uniformly at random. We denote $[k] \times \cdots \times [k]$ (p times) using $[k]^p$, where $p \in \mathbb{N}$. For us, d is a constant. $\mathcal{O}_d(\cdot)$ denotes the standard $\mathcal{O}(\cdot)$ where the constant depends on d. By polylogarithmic, in this chapter, we mean $\mathcal{O}_d\left(\frac{(\log n)^{\mathcal{O}_d(1)}}{\varepsilon^{\mathcal{O}(1)}}\right)$. The notation $\widetilde{\mathcal{O}}_d(\cdot)$ hides a polylogarithmic term in $\mathcal{O}_d(\cdot)$.

Organization of the chapter

We discuss about ordered hyperedges, that will be useful in our algorithm and analysis, and define in Section 4.2 two other query oracles, GPIS_1 and GPIS_2 that can be simulated by using polylogarithmic GPIS queries. The role of these two oracles is mostly expository – they help us to describe our algorithms and the calculations in a neater way. Section 4.3 gives a broad overview of our query algorithm that involves exact estimation, sparsification, coarse estimation and sampling. Section 4.3.3 contextualizes our work vis-a-vis recent works [BHR+18, BBGM19a, BBGM19b, DLM19]. The novel contribution of this work is sparsification which is given in detail in Section 4.4. Sections 4.5 and 4.6 consider the proofs for exact and coarse estimation, respectively. The algorithm and its proof of correctness are discussed in Sections 4.7 and 4.8, respectively.

4.2 Preliminaries

4.2.1 GPIS oracle and its variants

Note that the GPIS query oracle takes as input d pairwise disjoint subsets of vertices. We now define two related query oracles GPIS_1 and GPIS_2 that remove the disjointness criteria on the input. We show that both these query oracles can be simulated by making polylogarithmic number of queries to the GPIS oracle with high probability.

GPIS_1 and GPIS_2 oracles will be used in the description of the algorithm for ease of exposition.

(GPIS_1) Given s pairwise disjoint subsets of vertices $A_1, \ldots, A_s \subseteq U(\mathcal{H})$ of a hypergraph $\mathcal{H}$ and $a_1, \ldots, a_s \in [d]$ such that $\sum_{i=1}^{s} a_i = d$, GPIS_1 query oracle on input $A_1^{[a_1]}, A_2^{[a_2]}, \cdots, A_s^{[a_s]}$ answers YES if and only if $m(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \neq 0$.

(GPIS_2) Given any d subsets of vertices $A_1, \ldots, A_d \subseteq U(\mathcal{H})$ of a hypergraph $\mathcal{H}$, GPIS_2 query oracle on input $A_1, \ldots, A_d$ answers YES if and only if $m(A_1, \ldots, A_d) \neq 0$.

Observe that the GPIS_2 query oracle is the same as the GPIS query oracle without the requirement that the input sets are disjoint. For the GPIS_1 query oracle, multiple repetitions of the same set is allowed in the input. It is obvious that a GPIS query can be simulated by a GPIS_1 or GPIS_2 query oracle. Using the following observations, we show how a GPIS_1 or a GPIS_2 query can be simulated by making polylogarithmically many GPIS queries.

Observation 4.4. (i) A GPIS_1 query can be simulated using polylogarithmic GPIS queries with high probability.

(ii) A GPIS_2 query can be simulated using $2^{\mathcal{O}(d^2)}$ GPIS_1 queries.

(iii) A GPIS_2 query can be simulated using polylogarithmic GPIS queries with high probability.

Proof. (i) Let the input of GPIS_1 query oracle be $A_1^{[a_1]}, \ldots, A_s^{[a_s]}$ such that $a_i \in [d]$ $\forall i \in [s]$ and $\sum_{i=1}^{s} a_i = d$. For each $i \in [s]$, we partition A_i (only one copy of A_i, and not a_i copies of A_i) randomly into a_i parts, let $\{B_i^j : j \in [a_i]\}$ be the resulting partition of A_i. Then we make a GPIS query with input $B_1^1, \ldots, B_1^{a_1}, \ldots, B_s^1, \ldots, B_s^{a_s}$. Note that

$$\mathcal{F}(B_1^1, \ldots, B_1^{a_1}, \ldots, B_s^1, \ldots, B_s^{a_s}) \subseteq \mathcal{F}(A_1^{[a_1]}, \ldots, A_s^{[a_s]}).$$

So, if GPIS_1 outputs 'NO' to query $A_1^{[a_1]}, \ldots, A_s^{[a_s]}$, then the above GPIS query will also report 'NO' as its answer. If GPIS_1 answers 'YES', then consider a particular hyperedge $F \in \mathcal{F}(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$. Observe that

$$\begin{aligned}
& \mathbb{P}(\text{GPIS oracle answers 'YES'}) \\
\geq\ & \mathbb{P}(F \text{ is present in } \mathcal{F}(B_1^1, \ldots, B_1^{a_1}, \ldots\ldots, B_s^1, \ldots, B_s^{a_s})) \\
\geq\ & \prod_{i=1}^{s} \frac{1}{a_i^{a_i}} \\
\geq\ & \prod_{i=1}^{s} \frac{1}{d^{a_i}} \qquad (\because a_i \leq d \text{ for all } i \in [d]) \\
=\ & \frac{1}{d^d} \qquad (\because \sum_{i=1}^{s} a_i = d)
\end{aligned}$$

We can boost up the success probability arbitrarily by repeating the above procedure polylogarithmically many times.

(ii) Let the input to GPIS_2 query oracle be $A_1, \ldots, A_d$. Let us partition each set A_i into at most $2^{d-1} - 1$ subsets depending on A_i's intersection with A_j's for $j \neq i$. Let $\mathcal{P}_i$ denote the corresponding partition of A_i, $i \in [d]$. In particular, $\mathcal{P}_i$ contains any maximal set $B \subset A_i$ such that B is also the subset of ℓ $A_j's$ for some ℓ with $1 \leq \ell \leq d-1$, where $j \neq i$.

Observe that for any $i \neq j$, if we take any $B_i \in \mathcal{P}_i$ and $B_j \in \mathcal{P}_j$, then either $B_i = B_j$ or $B_i \cap B_j = \emptyset$.

For each $(B_1, \ldots, B_d) \in \mathcal{P}_1 \times \ldots \times \mathcal{P}_d$, we make a GPIS_1 query with input $(B_1, \ldots, B_d)$. Total number of such GPIS_1 queries is at most $2^{\mathcal{O}(d^2)}$, and we report 'YES' to the GPIS_2 query if and only if at least one GPIS_1 query, out of the $2^{\mathcal{O}(d^2)}$ queries, reports 'YES'.

(iii) It follows from (i) and (ii).

□

To prove Theorem 4.2, we first consider the following lemma. This lemma is the central result of the chapter and from it, the main theorem (Theorem 4.2) follows.

Lemma 4.5. *Let $\mathcal{H} = (U(\mathcal{H}), \mathcal{F}(\mathcal{H}))$ be a hypergraph with n vertices, i.e., $|U(\mathcal{H})| = n$. For any $\varepsilon > \left(n^{-d}\log^{5d+5} n\right)^{1/4}$,* HYPEREDGE-ESTIMATION *can be solved with probability at least $1 - \frac{1}{n^{4d}}$ and using $\mathcal{O}\left(\varepsilon^{-4}\log^{5d+4} n\right)$ queries, where each query is either a GPIS_1 query or a GPIS_2 query.*

Assuming Lemma 4.5 to be true, we now prove Theorem 4.2.

Proof of Theorem 4.2 . If $\varepsilon \leq \left(n^{-d}\log^{5d+5} n\right)^{1/4}$, we make a GPIS query with $(\{a_1\}, \ldots, \{a_d\})$ for all distinct $a_1, \ldots, a_d \in U(\mathcal{H}) = U$ and enumerate by brute force the exact value of $m_o(\mathcal{H})$. So, we make at most $n^d = \mathcal{O}_d\left(\varepsilon^{-4}\log^{5d+5} n\right)$ GPIS queries as $\varepsilon \leq \left(n^{-d}\log^{5d+5} n\right)^{1/4}$. If $\varepsilon > \left(n^{-d}\log^{5d+5} n\right)^{1/4}$, we use the algorithm corresponding to Lemma 4.5, where each query is either a GPIS_1 query or a GPIS_2 query. However, by Observation 4.4, each GPIS_1 and GPIS_2 query can be simulated by $\mathcal{O}_d(\log n)$ GPIS queries with high probability. So, we can replace each step of the algorithm, where we make either GPIS_1 or GPIS_2 query, by $\mathcal{O}_d(\log n)$ GPIS queries. Hence, we are done with the proof of Theorem 4.2. □

In the rest of the chapter, we mainly focus on proving Lemma 4.5. Now we discuss some notations that will be needed to describe the algorithm and analysis.

4.3 Technical overview

We briefly describe the overview of our work and put our work in context with works in the literature.

4.3.1 The context of our work

Beame *et al.* [BHR+18] developed a framework to estimate the number of edges in a graph using BIS queries. This framework involves subroutines for sparsifying a graph

into a number of subgraphs each with reduced number of edges, and exactly or approximately counting the number of edges in these subgraphs. Sparsification constitutes the main building block of this framework. The sparsification routine of Beame *et al.* randomly colors the vertices of a graph with $2k$ colors where $k \in \mathbb{N}$ is a constant. Let A_i denote the set of vertices colored with color i. Beame *et al.* argued that the sum of the number of edges between A_i and A_{k+i} for $1 \leq i \leq k$ approximates the total number of edges in the graph within a constant factor with high probability. Therefore, the original problem reduces to the problem of counting the number of edges in bipartite subgraphs. Here, we have k bipartite subgraphs and it suffices to count the number of edges in these bipartite subgraphs. With this method, they obtained a query complexity of $O(\varepsilon^{-4} \log^{14} n)$ with high probability.

Bhattacharya *et al.* [BBGM19b] extended this framework to estimate the number of triangles in a graph using TIS queries, where given three disjoint subsets of vertices A, B, C, a TIS query with inputs A, B, C answers whether there exists a triangle with one endpoint in each of A, B, C. The sparsification routine now requires to color the vertices randomly using $3k$ colors and counts the number of properly colored triangles where a properly colored triangle has one endpoint each in A_i, A_{k+i}, and A_{2k+i} for any $1 \leq i \leq k$. Unlike the scenario in Beame *et al.* where two edges intersect in at most one vertex, here two triangles can share an edge. Therefore, the random variables used to estimate the number of *properly colored* triangles are not independent. Bhattacharya *et al.* estimated the number of triangles assuming that the number of triangles incident on any edge is bounded by a parameter Δ. In this way, they obtained a query complexity of $O(\varepsilon^{-12}\Delta^{12} \log^{25} n)$.

In this work, we fully generalize the frameworks of Stockmeyer [Sto83, Sto85], Ron and Tsur [RT16], Beame *et al.* [BHR+18] and Bhattacharya *et al.* [BBGM19b] to estimate the number of hyperedges in a d-uniform hypergraph using GPIS queries.

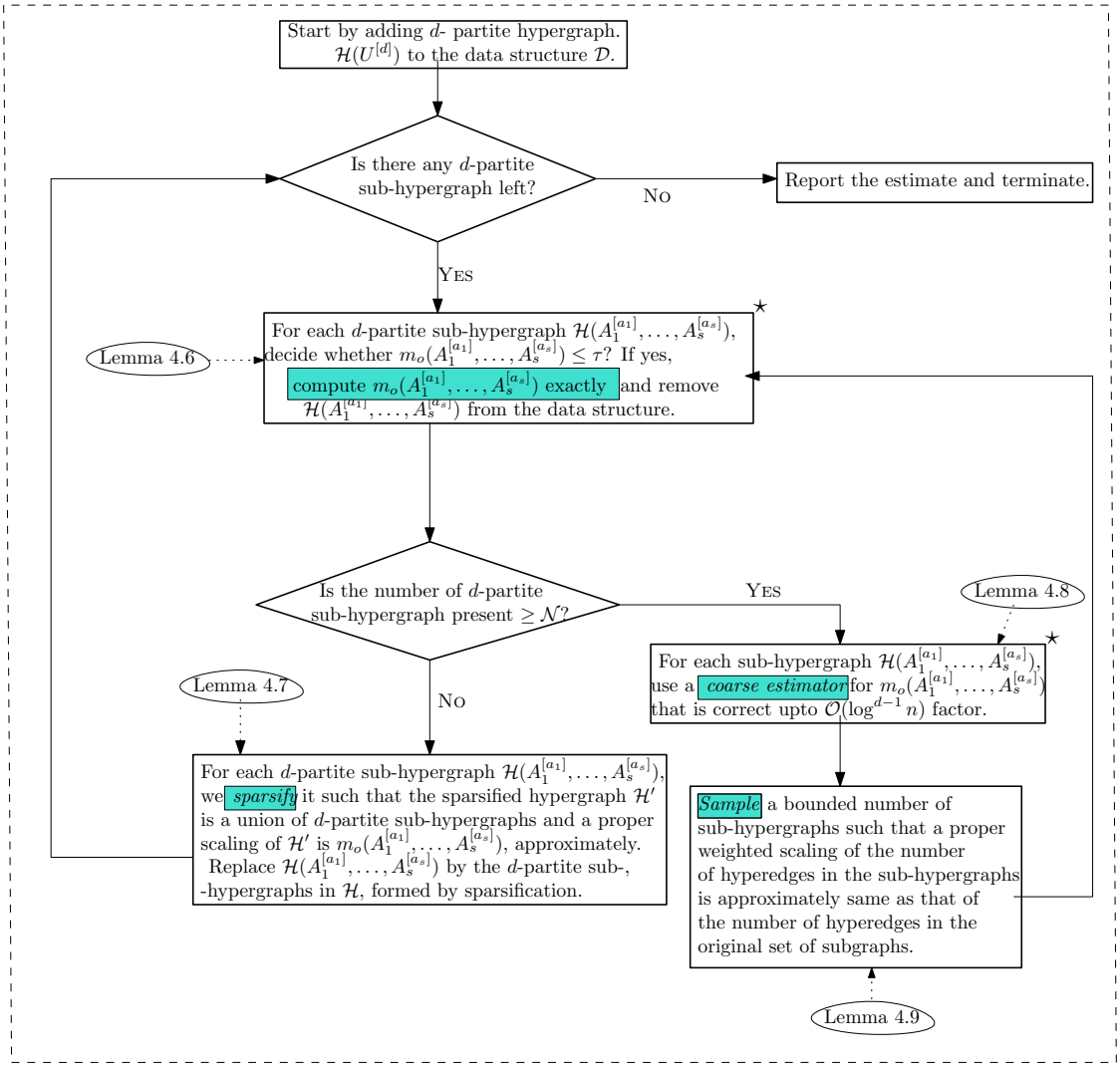

Figure 4.1: Flow chart of the algorithm. The highlighted texts indicate the basic building blocks of the algorithm. We also indicate the corresponding lemmas that support the building blocks. The building blocks marked with ⋆ correspond to the steps that make queries.

4.3.2 Our work in a nutshell

The algorithmic framework In Figure 4.1, we give a flowchart of the algorithm. In this chapter, sparsification (Lemma 4.7) and coarse estimation (Lemma 4.8) are our main non-trivial contributions. Our algorithm begins by adding the d-partite hypergraph

$\mathcal{H}(U^{[d]})$ in a suitable data structure, let us call it $\mathcal{D}$. For each hypergraph in $\mathcal{D}$, using the *exact estimation* process, we *exactly count* the number of hyperedges in a hypergraph if the number of hyperedges is less than a *threshold* τ and remove the corresponding hypergraph from $\mathcal{D}$. For hypergraphs in which the number of hyperedges is more than τ, we can not do *exact estimation*. So, we resort to *sparsification* if the number of hypergraphs left in $\mathcal{D}$ is not *large*, i.e., below a *threshold* $\mathcal{N}$. Hypergraph sparsification breaks a d-partite hypergraph into a disjoint union of d-partite sub-hypergraphs. As sparsification goes on in iterations, it may populate the data structure $\mathcal{D}$ with *significantly many* hypergraphs. In that case, we do a *coarse estimation* followed by *sampling* to have a reasonable number of sub-hypergraphs and then go back to the exact estimation step and continue in a loop. GPIS_1 and GPIS_2 queries will be used for exact and coarse estimations, respectively; their logarithmic equivalence with GPIS will prove the final result. For ease of analysis, we consider *ordered* hyperedges. Fact 4.3 relates $m(A_1, \ldots, A_d)$ and $m_o(A_1, \ldots, A_d)$, the number of unordered and ordered hyperedges, respectively.

Exact estimation. In this step, we look at a d-partite sub-hypergraph $\mathcal{H}(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$ and decide whether $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$, the number of ordered hyperedges, is larger or smaller than a threshold τ [4], and if $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \leq \tau$, compute the exact value of $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$ using Lemma 4.6, whose proof is in Section 4.5, and then delete it from $\mathcal{D}$. If the number of hypergraphs in $\mathcal{D}$ is below the threshold $\mathcal{N}$, with the number of ordered hyperedges in it more than τ, we move to the sparsification step, else to the coarse estimation step.

Lemma 4.6 (Exact Estimation)**.** *There exists a deterministic algorithm $\mathcal{A}_{exact}$ that takes as input – a d-uniform hypergraph $\mathcal{H}$, constants $a_1, \ldots, a_s \in [d]$ such that $\sum_{i=1}^{s} a_i = d$ where $s \in [d]$, pairwise disjoint subsets $A_1, \ldots, A_s$ of $U(\mathcal{H})$, and a threshold parameter $\tau \in \mathbb{N}$ – and decides whether the number of ordered hyperedges $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \leq \tau$ using $\mathcal{O}_d(\tau \log n)$ GPIS_1 queries. Moreover, $\mathcal{A}_{exact}$ finds the exact value of $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$ when $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \leq \tau$.*

[4] Threshold τ will be fixed later in Section 4.5.

Sparsification. We COLOR $U(\mathcal{H})$, the vertices of hypergraph, with $[k]$ colors to sparsify the d-partite hypergraph $\mathcal{H}(U^{[d]})$ so that

(i) the sparsified hypergraph consists of a set of d-partite sub-hypergraphs and

(ii) a proper scaling of the sum of the number of ordered hyperedges in the sub-hypergraphs is a *good* estimate of $m_o(U^{[d]}), U = U(\mathcal{H})$, with high probability.

The sparsification result is formally stated next. The proof uses the *method of averaged bounded differences* and *Chernoff-Hoeffding* inequality. The detailed proof is given in Section 4.4. The heart of our work is the general sparsification routine that we believe will find independent uses.

Lemma 4.7 (Sparsification)**.** *Let $\mathcal{H}$ be any d-uniform hypergraph and $k \geq 1$ be any positive integer. Let*

- *$h_d : [k]^d \to \{0,1\}$ be a hash function such that $\mathbb{P}[h_d(\mathbf{a}) = 1] = \frac{1}{k}$ and the set of random variables $\{h_d(\mathbf{a}) : \mathbf{a} \in [k]^d\}$ are mutually independent.*

- *$A_1, \ldots, A_s$ be any pairwise disjoint subsets of the vertex set $U(\mathcal{H})$, where $1 \leq s \leq d$. Let us choose any $a_i \in [d]$ vertices from A_i such that $\sum_{i=1}^{s} a_i = d$.*

- *Vertices in $A = \bigcup_{i=1}^{s} A_i$ are* COLORED *with $[k]$. $\chi(i,j)$'s denote the color classes for each A_i, that is, $\chi(i,j) = \{v \in A_i : v$ is* COLORED *with color $j\}$, where $i \in [s]$ and $j \in [k]$.*

- *An ordered hyperedge $(x_1, \ldots, x_d)$ is said to be* properly colored *if $h_d(c_1, \ldots, c_d) = 1$ where c_i is the color of x_i. Let $\mathcal{R}_d$ denote the number of* properly colored *hyperedges defined as follows. $\mathcal{R}_d =$*

$$\sum_{(c_1,\ldots,c_d)\in[k]^d} h_d(c_1,\ldots,c_d) \times m_o \left(\underbrace{\chi(1,c_1),\ldots,\chi(1,c_{a_1})}_{\text{color classes for } A_1^{[a_1]}}, \ldots, \underbrace{\chi(s,c_{d-a_s+1}),\ldots,\chi(s,c_d)}_{\text{color classes for } A_s^{[a_s]}} \right).$$

Then, for a suitable constant $\theta > d$ and $p_d = \frac{d!}{n^{4\theta-2d}}$,

$$\mathbb{P}\left(\left|\mathcal{R}_d - \frac{m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})}{k}\right| \geq 2^{2d}\theta^d \sqrt{d!\; m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \log^d n}\right) \leq p_d.$$

Sparsification ensures that $m_o(U^{[d]})$, the number of ordered hyperedges between $A_1^{[a_1]}, \ldots, A_s^{[a_s]}$, is approximately preserved when $m_o(U^{[d]})$ is above a threshold τ.

Assume that $m_o(U^{[d]})$ is *large* [5] and $\mathcal{H}(U^{[d]})$ has been sparsified. We add to the data structure $\mathcal{D}$ a set of d-partite sub-hypergraphs obtained from the sparsification step. Refer to the flowchart in Figure 4.1 again. With the new d-partite sub-hypergraphs, we loop back to the exact estimation step. After the exact estimation step, we are left with some d-partite hypergraphs such that the number of ordered hyperedges in each hypergraph is more than the threshold τ. If the number of such hypergraphs present in $\mathcal{D}$ is not large, i.e., below $\mathcal{N}$, then we sparsify each hypergraph $\mathcal{H}(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$ using the algorithm corresponding to Lemma 4.7.

Coarse estimation and sampling. If we have a large number of d-partite sub-hypergraphs of $\mathcal{H}(U^{[d]})$ and each sub-hypergraph contains a large number of ordered hyperedges, then we *coarsely* estimate the number of ordered hyperedges in each sub-hypergraph; see Figure 4.1. Our estimator is correct up to a $\mathcal{O}_d(\log^{d-1} n)$ factor using the algorithm corresponding to the following lemma, whose proof is given in Section 4.6.

Lemma 4.8 (Coarse Estimation)**.** *There exists an algorithm $\mathcal{A}_{coarse}$ that takes as input d subsets $A_1, \ldots, A_d$ of vertex set $U(\mathcal{H})$ of a d-uniform hypergraph $\mathcal{H}$ and returns $\widehat{E}$ as an estimate for $m_o(A_1, \ldots, A_d)$ such that*

$$\frac{m_o(A_1, \ldots, A_d)}{8d^{d-1}2^d \log^{d-1} n} \leq \widehat{E} \leq 20d^{d-1}2^d \cdot m_o(A_1, \ldots, A_d) \log^{d-1} n$$

with probability at least $1 - n^{-8d}$. Moreover, the number of GPIS_2 *queries made by the algorithm is $\mathcal{O}_d(\log^{d+1} n)$.*

[5] A fixed polylogarithmic quantity to be decided later in Section 4.7.

First, we coarsely estimate the number of ordered hyperedges in each sub-hypergraph. Then we sample a set of sub-hypergraphs such that a weighted sum of the number of hyperedges in the sample approximately preserves the sum of the number of ordered hyperedges in the sub-hypergraphs. This kind of sampling technique, also known as the importance sampling, is given for the edge estimation problem by Beame *et al.* [BHR$^+$18]. The lemma corresponding to this sampling technique is formally stated as Lemma 2.9 in Section 2.1. The importance sampling lemma that we require is stated as follows.

Lemma 4.9 (Importance Sampling)**.** *Let* $\{(A_{i1}, \ldots, A_{id}, w_i) : i \in [r]\}$ *be the set of tuples in the data structure* $\mathcal{D}$ *and* e_i *be the coarse estimate for* $m_o(A_{i1}, \ldots, A_{id}, w_i)$ *such that*

(i) $w_i, e_i \geq 1 \; \forall i \in [r]$,

(ii) $\frac{e_i}{\alpha} \leq m_o(A_{i1}, \ldots, A_{id}) \leq e_i \cdot \alpha$ *for some* $\alpha > 0$ *and* $\forall \; i \in [r]$,

(iii) $S = \sum_{i=1}^{r} w_i \cdot m_o(A_{i1}, \ldots, A_{id}) \leq M$.

Then, there exists an algorithm that finds a set $\{(A'_{i1}, \ldots, A'_{id}, w'_i) : i \in [r']\}$ *of tuples, with probability at least* $1 - \delta$*, such that the above three conditions hold and*

$$\left| \sum_{i=1}^{r'} w'_i \cdot m_o(A'_{i1}, \ldots, A'_{id}) - \sum_{i=1}^{r} w_i \cdot m_o(A_{i1}, \ldots, A_{id}) \right| \leq \lambda S \,,$$

where $r' = \mathcal{O}\left(\frac{\alpha^4 \log M}{\lambda^2} \left(\log\log M + \log \frac{1}{\delta}\right)\right)$.

Putting things together. The data structure $\mathcal{D}$ that our iterative algorithm uses (see the flowchart in Figure 4.1) has an accumulator Ψ for the number of hyperedges and a set of tuples representing the d-partite sub-hypergraphs along with their weights – these d-partite sub-hypergraphs are generated from sparsification. The algorithm uses the *exact estimation* step (viz. Algorithm 4.1) to figure out if the number of hyperedges $m_o(A_1, \ldots, A_d) \leq \tau$ using Lemma 4.6. If $m_o(A_1, \ldots, A_d) \leq \tau$, we add $w \cdot m_o(A_1, \ldots, A_d)$ to Ψ and remove the tuple $(A_1, \ldots, A_d, w)$ from $\mathcal{D}$. If the number of

tuples in $\mathcal{D}$ is at most $\mathcal{N} = \kappa_d \cdot \frac{\log^{4d} n}{\varepsilon^2}$, then we carry out a sparsification step, else we do an importance sampling; a coarse estimation step (viz. Algorithms 4.2 and 4.3) acts as a pre-processing step for importance sampling. We do either sparsification or coarse estimation followed by importance sampling in an iteration because their roles are complementary. While successive sparsifications ease estimating the number of hyperedges by breaking a d-partite hypergraph into a disjoint union of d-partite sub-hypergraphs, the number of such d-partite sub-hypergraphs may be as high as $4^d \cdot \mathcal{N} = 4^d \kappa_d \cdot \frac{\log^{4d} n}{\varepsilon^2}$ (see Observation 4.20). Coarse estimation followed by importance sampling (see Lemma 4.9) ensures that the effect of sparsification is neutralized so that the number of tuples in $\mathcal{D}$ is at most $\mathcal{N}$. This ensures that out of two successive iterations of the algorithm, there is at least one sparsification step. The coarse estimation applies when the number of tuples $r > \mathcal{N} = \frac{\kappa_d \log^{4d} n}{\varepsilon^2}$. For each tuple in $\mathcal{D}$, we find an estimate $\widehat{E_i}$ using $\mathcal{O}_d(\log^{d+1} n)$ GPIS_2 queries per tuple (see Lemma 4.8). This estimate serves as condition (ii) for the application of Lemma 4.9. Because of the importance sampling (see Lemma 4.9), one can replace the r tuples in $\mathcal{D}$ with a sample of r' tuples such that the required estimate is approximately maintained. Since $r' \leq \kappa_d \cdot \frac{\log^{4d} n}{\varepsilon^2} = \mathcal{N}$, the next step will be sparsification.

Observe that the query complexity of each iteration is polylogarithmic. Note that the number of ordered hyperedges reduces by a constant factor after each sparsification step. So, the number of iterations is bounded by $\mathcal{O}_d(\log n)$. Hence, the query complexity of our algorithm is polylogarithmic. This completes a high level description of our algorithm.

4.3.3 Our work vis-a-vis some recent works

Comparison with Beame et al. [BHR$^+$18] and Bhattacharya et al. [BBGM18,BBGM19b]: Beame et al. [BHR$^+$18] showed that EDGE ESTIMATION problem can be solved using $\mathcal{O}(\varepsilon^{-4} \log^{14} n)$ BIS queries. Note that a BIS query is a special case of GPIS query for $d = 2$. Bhattacharya et al. [BBGM18, BBGM19b] first considered the TRIANGLE ESTIMATION problem in graphs using a TIS query oracle, a generalization of BIS query

oracle. They showed that a $(1 \pm \varepsilon)$-approximation to the number of triangles can be found by using $\mathcal{O}(\Delta^{12}\varepsilon^{-12}\log^{25} n)$ TIS queries, where Δ denotes the maximum number of triangles sharing an edge. Note that the TRIANGLE ESTIMATION problem in graphs using TIS queries is analogous to the HYPEREDGE-ESTIMATION problem in 3-uniform hypergraphs using a special GPIS query when $d = 3$. In this work, we build on the works of Beame et al. [BHR$^+$18] and Bhattacharya et al. [BBGM18, BBGM19b] for HYPEREDGE-ESTIMATION using GPIS queries. Our main contribution in this work is twofold and that answers Questions 1 and 2 posed in Section 4.1. Firstly, our query complexity bounds are independent of Δ, this dependence on Δ was present in the work of Bhattacharya et al. [BBGM18, BBGM19b]. Secondly, our work can be seen as a complete generalization of the works of Beame et al. [BHR$^+$18] and Bhattacharya et al. [BBGM18, BBGM19b] to d-uniform hypergraphs. As mentioned earlier, the extension from the EDGE ESTIMATION problem using BIS queries to the HYPEREDGE ESTIMATION problem using GPIS queries is not obvious as edges in a graph can intersect in at most one vertex, but the intersection pattern of hyperedges in hypergraphs is complicated. The main building blocks of the algorithms of Beame *et al.* and Bhattacharya *et al.* are *sparsification*, *exact estimation* and *coarse estimation*. We generalize their exact estimation and coarse estimation algorithms by more detailed and careful analysis. The sparsification algorithm in the work of Beame et al. [BHR$^+$18] uses heavily the fact that two edges in a graph can intersect at most one vertex. This may be the reason why Bhattacharya et al. [BBGM18, BBGM19b] required a bound on the number of triangles sharing an edge. In this work, we show that an ingenious way of coloring the hyperedges in a d-uniform hypergraph and an involved use of induction on d generalize the results of both Beame et al. [BHR$^+$18] and Bhattacharya et al. [BBGM18, BBGM19b]. The main technical and non-trivial contribution in this work is the sparsification algorithm, which enables us to generalize the works of Beame et al. [BHR$^+$18] and Bhattacharya et al. [BBGM18, BBGM19b]. Their sparsification results do not generalize for arbitrary d without any assumptions on the intersection pattern of the hyperedges.

Comparison with Dell et al. [DLM19]: Concurrently and independently, Dell et al. [DLM19] obtained similar results for the hyperedge estimation problem. They showed that one can find a $(1 \pm \varepsilon)$-approximation to the number of hyperedges in a d-uniform hypergraph by using $\mathcal{O}_d\left(\frac{\log^{4d+7} n}{\varepsilon^2} \log \frac{1}{\delta}\right)$ *colorful independent set* queries with probability at least $1-\delta$. A colorful independent set query is same as that of a GPIS query oracle considered in this work; GPIS query was introduced by Bishnu et al. [BGK$^+$18a]. Note that the algorithm of Dell et al. [DLM19] for HYPEREDGE-ESTIMATION uses comparable but fewer number of GPIS queries. However, our algorithm is different and conceptually much simpler than that of Dell et al. [DLM19]. Most importantly, the main technical tool, our sparsification step is completely different from theirs.

4.4 Sparsification: Proof of Lemma 4.7

We start by reproducing the statement of the Lemma 4.7 below. Sparsification ensures that the number of ordered hyperedges $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$ is approximately preserved across any pairwise disjoint subsets of vertices $A_1, \ldots, A_s$ and integers $a_1, \ldots, a_s$ such that $\sum_{i=1}^{s} a_i = d$.

Lemma 4.10 (Sparsification)**.** *Let $\mathcal{H}$ be any d-uniform hypergraph and $k \geq 1$ be any positive integer. Let*

- *$h_d : [k]^d \to \{0,1\}$ be a hash function such that $\mathbb{P}[h_d(\mathbf{a}) = 1] = \frac{1}{k}$ and the set of random variables $\{h_d(\mathbf{a}) : \mathbf{a} \in [k]^d\}$ are mutually independent.*
- *$A_1, \ldots, A_s$ be any pairwise disjoint subsets of $U(\mathcal{H})$, where $1 \leq s \leq d$. Let us choose any $a_i \in [d]$ vertices from A_i such that $\sum_{i=1}^{s} a_i = d$.*
- *Vertices in $A = \bigcup_{i=1}^{s} A_i$ are* COLORED *with $[k]$. $\chi(i,j)$'s denote the color classes for each A_i, that is, $\chi(i,j) = \{v \in A_i : v$ is* COLORED *with color $j\}$, where $i \in [s]$ and $j \in [k]$.*
- *An ordered hyperedge $(x_1, \ldots, x_d)$ is said to be* properly colored *if $h_d(c_1, \ldots, c_d) =$*

1 *where c_i is the color of x_i. Let $\mathcal{R}_d$ denote the number of* properly colored *hyperedges defined as follows and* $\mathcal{R}_d =$

$$\sum_{(c_1,\ldots,c_d)\in[k]^d} m_o\left(\underbrace{\chi(1,c_1),\ldots,\chi(1,c_{a_1})}_{\textit{color classes for } A_1^{[a_1]}},\ldots,\underbrace{\chi(s,c_{d-a_s+1}),\ldots,\chi(s,c_d)}_{\textit{color classes for } A_s^{[a_s]}}\right)\times h_d(c_1,\ldots,c_d).$$

Then, for a suitable constant $\theta > d$ and $p_d = \frac{d!}{n^{4\theta-2d}}$,

$$\mathbb{P}\left(\left|\mathcal{R}_d - \frac{m_o(A_1^{[a_1]},\ldots,A_s^{[a_s]})}{k}\right| \geq 2^{2d}\theta^d\sqrt{d!m_o(A_1^{[a_1]},\ldots,A_s^{[a_s]})\log^d n}\right) \leq p_d.$$

The above lemma says that the properly colored hyperedges approximate the number of ordered hyperedges upto a scaling factor when the number of (ordered) hyperedges is above a thershold. Before getting into the formal proof of the lemma, let us explore the role of the hash function in sparsification.

4.4.1 The role of the hash function in sparsification

We feel our work could generalize the framework of Beame et al. [BHR$^+$18] to hyperedge estimation mainly because of the new sparsification result aided by an involved use of induction. The sparsification result depends on the particular choice of hash function. As we move from edge to triangle and then to hyperedge in a d-uniform hypergraph, implicit structures blow up, and managing their interrelations also become difficult. Consider the following. Two edges can intersect in at most one vertex. Two triangles can intersect in at most two vertices and two hyperedges in d-uniform hypergraph intersect in at most $d-1$ vertices.

To count the number of edges in G using sparsification, Beame et al. [BHR$^+$18] COLORED the vertices with k colors and looked at the edges between certain pairs of color classes. So, it boils down to counting the number of properly colored edges. This random coloring process can be encoded by the hash function $h : [k] \to [k]$. In

an effort to generalize Beame et al.'s [BHR$^+$18] work to count the number of triangles in a graph in Chapter 3, we colored the vertices of the graph for sparsification in a *specific* way. Though we have not described the sparsification in Chapter 3 in terms of hash function, the coloring function can be thought of as a hash function of the form — $h : [k] \times [k] \to [k]$ in [BBGM18, BBGM19b]. Look at any edge (a, b) in this graph under the above hash function. The edge is properly colored if the colors on the vertices of the edge come from specified color classes. We count all triangles on these properly colored edges, and scale it appropriately. Once an edge is not properly colored, we may miss it and hence, all triangles incident on that edge. This is why we need a bound on the number of triangles incident on an edge in [BBGM18, BBGM19b], otherwise the variance blows up and the concentration inequality gives poor result.

In this work, we consider a different hashing scheme with the hash function $h : [k]^d \to \{0, 1\}$, in the d-uniform hypergraph setting, with a suitable probability of hash values becoming 1. For triangles (3-uniform hypergraph), the particular hash function is thus $h : [k]^3 \to \{0, 1\}$. So, there is no concept of proper and improper edges. All edges are taken into consideration and then all triangles, and we can work with any number of triangles incident on an edge. Moreover, there is an inductive nature to this particular choice of hash function. Consider fixing the color of a vertex of a triangle. Then the hash function $h : [k] \times [k] \to \{0, 1\}$ behaves as a random coloring on the other two vertices of the triangle.

To elaborate further, our sparsification algorithm independently colors the vertices of the hypergraph by selecting a color uniformly at random from $\{1, \ldots, k\}$, and colors each d-tuple in $\{1, \ldots, k\}^d$ (i.e., $\{1, \ldots, k\} \times \cdots \times \{1, \ldots, k\}$) randomly with a color out of two colors, say 0 and 1, such that the probability of each d-tuple being colored with 1 is $\frac{1}{k}$. We say that an ordered hyperedge F_o is *properly* colored if the d-tuple of colors, assigned to the vertices of F_o, is colored with 1. The main insight is that, when the number of hyperedges is above a threshold, a suitable scaling of properly colored hyperedges, will approximate the number of hyperedges. The case when the number of hyperedges is below a threshold can be taken care of by the exact estimation step. Indeed, we overcome the bottleneck of arbitrary intersection patterns in the case of d-

uniform hyperedges by the two levels of randomness.

4.4.2 Proof of the lemma

Proof. We prove this lemma using induction on d.

The base case: For $d = 1$, the hash function is $h_1 : [k] \to \{0, 1\}$. The vertices in A_1 are COLORED with $[k]$. So, $\mathcal{R}_1 = \sum_{c_1 \in [k]} m_o(\chi(1, c_1)) \times h_1(c_1)$. We have $\mathbb{E}[\mathcal{R}_1] = \frac{m_o(A_1)}{k}$. Note that the set of 1-uniform hyperedges $\mathcal{F}_o(A_1)$ is a subset of A_1. Consider $F \in \mathcal{F}_o(A_1)$. Let $x \in [k]$ be the color assigned to the single element present in F. Let X_F be the indicator random variable such that $X_F = 1$ if and only if $h_1(x) = 1$. Therefore, $\mathbb{P}(X_F = 1) = \frac{1}{k}$. Observe that the random variables X_F are mutually independent as F depends only on A_1. The number of properly colored hyperedges $\mathcal{R}_1$ can be expressed as the sum of the mutually independent random variables X_F, i.e., $\mathcal{R}_1 = \sum_{F \in \mathcal{F}_o(A_1)} X_F$. Now, applying Hoeffding's bound (see Lemma 2.3 in Section 2.1), we get

$$\mathbb{P}\left(\left|\mathcal{R}_1 - \frac{m_o(A_1)}{k}\right| \geq 4\theta \cdot \sqrt{\log n \cdot m_o(A_1)}\right) \leq \frac{2}{n^{16\theta^2}} \leq p_1.$$

The inductive case: Observe that the probability that a hyperedge is *properly colored* with a color tuple $(c_1, \ldots, c_d)$ is same as the probability of the hash function evaluating to 1 for that color tuple and is equal to $\frac{1}{k}$. $\mathcal{R}_d$, as defined in the statement of Lemma 4.10, represents the number of properly colored hyperedges. So, $\mathbb{E}[\mathcal{R}_d] = m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})/k$.

Note that $A = \cup_{i=1}^{s} A_i$. Without loss of generality, assume that the vertices in A are $1, \ldots, n'$, where $n' \leq n$. Viewing the coloring process as a random process, let us focus on the instance when, in some arbitrary order, vertices $1, \ldots, t-1$ have been colored and we are exposing the vertex t for coloring, where $2 \leq t \leq n' - 1$. Recall that $\mathcal{F}_o(t)$ denotes the set of ordered hyperedges incident on t. We classify the ordered hyperedges $\mathcal{F}_o(t)$ based on the position of t, i.e., let $\mathcal{F}_o(t, \mu) \subseteq \mathcal{F}_o(t)$ be the set of ordered hyperedges where t is fixed as the μ-th vertex, $\mu \in [d]$. As $|\mathcal{F}_o(t)| = d!\mathcal{F}(t)$ and

$|\mathcal{F}_o(t,\mu)| = (d-1)!\mathcal{F}(t)$, we have the following observation that will be used later in the proof.

Observation 4.11. $|\mathcal{F}_o(t,\mu)| = \frac{(d-1)!\ |\mathcal{F}_o(t)|}{d!} = \frac{|\mathcal{F}_o(t)|}{d}$ where $\mu \in [d]$.

Next, we bring in a definition that would allow us to capture the randomness in the coloring process when the vertex t is colored.

Definition 4.12. An ordered hyperedge $F \in \mathcal{F}_o(t,\mu)$ is said to be of *type-λ* if F has exactly λ vertices from the set $[t]$, where $\lambda \in [d]$.

To get a sense of the above definition, consider the cases $\lambda < d$ and $\lambda = d$; the second case fixes all vertices in F to be from $[t]$, while in the first case, there is room for vertices beyond $[t]$. For $\mu, \lambda \in [d]$, let $\mathcal{F}_o^{\lambda}(t)$ and $\mathcal{F}_o^{\lambda}(t,\mu)$ be the set of ordered hyperedges of type-λ in $\mathcal{F}_o(t)$ and $\mathcal{F}_o(t,\mu)$, respectively. Given that the vertex t is colored with color $c \in [k]$, let $N_c^{\lambda}(t)$ and $N_c^{\lambda}(t,\mu)$ be the random variables that denote the number of ordered hyperedges in $\mathcal{F}_o^{\lambda}(t)$ and $\mathcal{F}_o^{\lambda}(t,\mu)$ that are properly colored, respectively. Let $Z_i \in [k]$ be the random variable that denotes the color assigned to the vertex $i \in A$ (recall $A = \bigcup_{i=1}^{s} A_i) = [n']$, where $n' \leq n$. Note that $\mathcal{R}_d$ is a function of $Z_1, \ldots, Z_{n'}$, that is, $\mathcal{R}_d = f(Z_1, \ldots, Z_{n'})$.

Let $\mathbb{E}_{\mathcal{R}_d}^t$ denote the difference in the conditional expectation of the number of properly colored ordered hyperedges given that the t-th vertex is differently colored by considering the hyperedges in each $\mathcal{F}_o^{\lambda}(t,\mu)$, where $\lambda, \mu \in [d]$.

$$\begin{aligned}
\mathbb{E}_{\mathcal{R}_d}^t &= \left|\mathbb{E}\left[\mathcal{R}_d \mid Z_1, \ldots, Z_{t-1}, Z_t = \rho\right] - \mathbb{E}\left[\mathcal{R}_d \mid Z_1, \ldots, Z_{t-1}, Z_t = \nu\right]\right| && (4.1)\\
&= \left|\sum_{\lambda=1}^{d} \mathbb{E}\left[N_{\rho}^{\lambda}(t) - N_{\nu}^{\lambda}(t)\right]\right| && (4.2)\\
&= \left|\left(N_{\rho}^{d}(t) - N_{\nu}^{d}(t)\right) + \sum_{\lambda=1}^{d-1} \mathbb{E}\left[N_{\rho}^{\lambda}(t) - N_{\nu}^{\lambda}(t)\right]\right| && (4.3)\\
& && (4.4)
\end{aligned}$$

$$= \left| \sum_{\mu=1}^{d} \left(N_\rho^d(t,\mu) - N_\nu^d(t,\mu) \right) + \sum_{\lambda=1}^{d-1} \mathbb{E}\left[N_\rho^\lambda(t) - N_\nu^\lambda(t) \right] \right| \tag{4.5}$$

$$\leq \sum_{\mu=1}^{d} \left| N_\rho^d(t,\mu) - N_\nu^d(t,\mu) \right| + \left| \sum_{\lambda=1}^{d-1} \mathbb{E}\left[N_\rho^\lambda(t) - N_\nu^\lambda(t) \right] \right| \tag{4.6}$$

Since the hyperedges without having t as one of its vertex have the same contribution to both $\mathbb{E}\left[\mathcal{R}_d \mid Z_1, \ldots, Z_{t-1}, Z_t = \rho\right]$ and $\mathbb{E}\left[\mathcal{R}_d \mid Z_1, \ldots, Z_{t-1}, Z_t = \nu\right]$, Equation 4.2 follows from Equation 4.1.

Now, consider the following claim.

Claim 4.13. *(a)* $\mathbb{P}\left(\left| N_\rho^d(t,\mu) - N_\nu^d(t,\mu) \right| \leq 2^{2d-1}\theta^{d-1}\sqrt{(d-1)!\mathcal{F}_o(t,\mu)\log^{d-1} n} \right) \geq 1 - 2p_{d-1}$, *where* $\mu \in [d]$ *and* $\theta > d$ *is the constant mentioned in the statement of Lemma 4.10.*

(b) $\mathbb{E}\left[N_\rho^\lambda(t) - N_\nu^\lambda(t) \right] = 0,\ \forall \lambda \in [d-1]$.

Proof of Claim 4.13. (a) For simplicity, we argue for $\mu = 1$, i.e., t appears at the first position. However, the argument will be similar for any $\mu \in [d]$. Consider a $(d-1)$-uniform hypergraph $\mathcal{H}'$ such that $U(\mathcal{H}') = \{1, \ldots, t-1\}$and $\mathcal{F}_o(\mathcal{H}') = \{(x_1, \ldots, x_{d-1}) : (t, x_1, \ldots, x_{d-1}) \in \mathcal{F}_o^d(t,1)\}$. Let $h_{d-1} : [k]^{d-1} \to \{0,1\}$ be a hash function such that $h_{d-1}(x_1, \ldots, x_{d-1}) = h_d(t, x_1, \ldots, x_d)$. Observe that $\mathbb{P}(h_{d-1}(\mathbf{a}) = 1) = \frac{1}{k}$ for each tuple $\mathbf{a} \in [k]^{d-1}$. Consider the $(d-1)$-partite hypergraph $\mathcal{H}'\left(B_1^{[a_1-1]}B_2^{[a_2]}\cdots B_s^{[a_s]}\right)$, where $B_i = A_i \cap [t-1]$ [6]. Observe that $m_o\left(B_1^{[a_1-1]}B_2^{[a_2]}\cdots B_s^{[a_s]}\right) = \left|\mathcal{F}_o^d(t,1)\right|$. Recall that the vertices in $\cup_{i=1}^s A_i$ are COLORED with $[k]$. Let $\chi'(i,j) = \{v \in B_i : v \text{ is COLORED with color } j\}$. Let $\mathcal{R}_{d-1} =$

$$\sum_{(c_1,\ldots,c_{d-1})\in[k]^{d-1}} m_o\left(\chi'(1,c_1), \ldots, \chi'(1,c_{a_1-1}), \ldots, \chi'(s,c_{d-a_s})\ldots\chi'(s,c_{d-1})\right) \times h_{d-1}(c_1, \ldots, c_{d-1}).$$

[6] Recall that $B^{[b]}$ denotes $B \ldots B$(b times).

Observe that the random variables $N_\rho^d(t,1)$ and $N_\nu^d(t,1)$ follow the distribution of the random variable $\mathcal{R}_{d-1}$. By the induction hypothesis,

$$\mathbb{P}\left(\left|\mathcal{R}_{d-1} - \frac{m_o\left(B_1^{[a_1-1]}B_2^{[a_2]}\cdots B_s^{[a_s]}\right)}{k}\right| \geq \aleph\right) \leq p_{d-1},$$

where $\aleph = 2^{2d-2}\theta^{d-1}\sqrt{(d-1)!m_o\left(B_1^{[a_1-1]}B_2^{[a_2]}\cdots B_s^{[a_s]}\right)\log^{d-1} n}$.

Using $m_o\left(B_1^{[a_1-1]}B_2^{[a_2]}\cdots B_s^{[a_s]}\right) = \left|\mathcal{F}_o^d(t,1)\right|$, we have

$$\mathbb{P}\left(\left|\mathcal{R}_{d-1} - \frac{\left|\mathcal{F}_o^d(t,1)\right|}{k}\right| \geq 2^{2d-2}\theta^{d-1}\sqrt{(d-1)!\mathcal{F}_o^d(t,1)\log^{d-1} n}\right) \leq p_{d-1} \tag{4.7}$$

Using the above equation, the claim follows using the following.

$$\mathbb{P}\left(\left|N_\rho^d(t,1) - N_\nu^d(t,1)\right| \geq 2^{2d-1}\theta^{d-1}\sqrt{(d-1)!\mathcal{F}_o(t,1)\log^{d-1} n}\right) \leq 2p_{d-1}.$$

Let $L = 2^{2d-1}\theta^{d-1}\sqrt{(d-1)!\mathcal{F}_o(t,1)\log^{d-1} n}$. Now,

$$\begin{aligned}
& \mathbb{P}\left(\left|N_\rho^d(t,1) - N_\nu^d(t,1)\right| \geq 2^{2d-1}\theta^{d-1}\sqrt{(d-1)!\mathcal{F}_o(t,1)\log^{d-1} n}\right) \\
\leq\ & \mathbb{P}\left(\left|N_\rho^d(t,1) - \frac{\left|\mathcal{F}_o^d(t,1)\right|}{k}\right| \geq \frac{L}{2}\right) + \mathbb{P}\left(\left|N_\nu^d(t,1) - \frac{\left|\mathcal{F}_o^d(t,1)\right|}{k}\right| \geq \frac{L}{2}\right) \\
=\ & 2\cdot\mathbb{P}\left(\left|N_\rho^d(t,1) - \frac{\left|\mathcal{F}_o^d(t,1)\right|}{k}\right| \geq 2^{2d-2}\theta^{d-1}\sqrt{(d-1)!\mathcal{F}_o(t,1)\log^{d-1} n}\right) \\
\leq\ & 2\cdot\mathbb{P}\left(\left|N_\rho^d(t,1) - \frac{\left|\mathcal{F}_o^d(t,1)\right|}{k}\right| \geq 2^{2d-2}\theta^{d-1}\sqrt{(d-1)!\mathcal{F}_o^d(t,1)\log^{d-1} n}\right) \\
& \qquad\qquad (\because \mathcal{F}_o^d(t,1) \leq \mathcal{F}_o(t,1)) \\
\leq\ & 2p_{d-1} \quad \text{(By Equation 4.7)}
\end{aligned}$$

(b) First, consider the case when t is COLORED with color ρ. For $F \in \mathcal{F}_o^\lambda(t), \lambda \in [d-1]$, let X_F be the indicator random variable such that $X_F = 1$ if and only if F is properly colored. As F is of type λ, there exists at least one vertex in F that is not colored yet, that is, $\mathbb{P}(X_F = 1) = \frac{1}{k}$. Observe that $N_\rho^\lambda(t) = \sum_{F \in \mathcal{F}_o^\lambda(t)} X_F$. Hence, $\mathbb{E}\left[N_\rho^\lambda(t)\right] = \left|\mathcal{F}_o^\lambda(t)\right|/k$. Similarly, one can show that $\mathbb{E}\left[N_\nu^\lambda(t)\right] = \left|\mathcal{F}_o^\lambda(t)\right|/k$. Hence, $\mathbb{E}\left[N_\rho^\lambda(t) - N_\nu^\lambda(t)\right] = 0$.

□

Now, let us come back to the proof of Lemma 4.10. By Claim 4.13 and Observation 4.11, we have the following with probability at least $1 - 2d \cdot p_{d-1}$.

$$\mathbb{E}^t_{\mathcal{R}_d} \leq 2^{2d-1}\theta^{d-1} \cdot d \cdot \sqrt{(d-1)!\frac{\mathcal{F}_o(t)}{d}\log^{d-1} n} = 2^{2d-1}\theta^{d-1}\sqrt{d!\mathcal{F}_o(t)\log^{d-1} n} = c_t,$$

where $c_t = 2^{2d-1}\theta^{d-1}\sqrt{d!\mathcal{F}_o(t)\log^{d-1} n}$.

Let $\mathcal{B}$ be the event that there exists $t \in [n]$ such that $\mathbb{E}^t_{\mathcal{R}^d} > c_t$. By the union bound over all $t \in [n]$, $\mathbb{P}(\mathcal{B}) \leq 2dnp_{d-1} = 2dn\frac{(d-1)!}{n^{4\theta-2(d-1)}} \leq \frac{2d!}{n^{4\theta-2d+1}}$. Using the method of *averaged bounded difference* [DP09] (See Lemma 2.2 in Section 2.1), we have

$$\mathbb{P}\left(|\mathcal{R}_d - \mathbb{E}[\mathcal{R}_d]| > \delta + m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})\mathbb{P}(\mathcal{B})\right) \leq e^{-\delta^2/\sum_{t=1}^n c_t^2} + \mathbb{P}(\mathcal{B})$$

We set $\delta = 2\sqrt{\theta \log n \cdot \sum_{t\in[n]} c_t^2} = 2^{2d}\theta^{d-1/2}\sqrt{d!m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})\log^d n}$. Using $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \leq n^d$ and $\mathbb{P}(\mathcal{B}) \leq \frac{2d!}{n^{4\theta-4d+1}}$, and taking

$$\eth = 2^{2d}\theta^{d-1/2}\sqrt{d!m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})\log^d n} + \frac{2d!n^d}{n^{4\theta-2d+1}},$$

we have

$$\mathbb{P}\left(\left|\mathcal{R}_d - \frac{m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})}{k}\right| > \eth\right) \leq \frac{1}{n^{4\theta}} + \frac{2d!}{n^{4\theta-2d+1}}.$$

Assuming $n \gg d$,

$$\begin{aligned} & \mathbb{P}\left(\left|\mathcal{R}_d - \frac{m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})}{k}\right| \geq 2^{2d}\theta^d \sqrt{d! m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \log^d n}\right) \\ \leq & \frac{d!}{n^{4\theta-2d}} = p_d. \end{aligned}$$

□

4.5 Proof of lemma for exact estimation

In this Section, we prove Lemma 4.6. We restate the lemma for easy reference.

Lemma 4.14 (Exact Estimation: Lemma 4.6 restated)**.** *There exists a deterministic algorithm $\mathcal{A}_{exact}$ that takes as input — a d-uniform hypergraph $\mathcal{H}$, constants $a_1, \ldots, a_s \in [d]$ such that $\sum_{i=1}^{s} a_i = d$ where $s \in [d]$, pairwise disjoint subsets $A_1, \ldots, A_s$ of $U(\mathcal{H})$, and threshold parameter $\tau \in \mathbb{N}$ — and decides whether the number of ordered hyperedges $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \leq \tau$ using $\mathcal{O}_d(\tau \log n)$ GPIS_1 queries. Moreover, the algorithm finds the exact value of $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$ when $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \leq \tau$.*

Proof. First, we discuss an algorithm that determines the exact value of $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$ using $\mathcal{O}_d(m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \log n)$ GPIS_1 queries. Then, we argue how to modify it to obtain the desired bound. We initialize a multi-ary tree $\mathcal{T}$ with $(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$ as the root and then build it level by level. At the next level, each A_i is split into two halves – $A_{i,1}$ and $A_{i,2}$ and each node in the next level is labeled with $A_{i,j}$, $i \in \{1, \ldots, s\}$, $j \in \{1, 2\}$; so in the next level there are $2^s \leq 2^d$ nodes. Notice that as we go down the levels in $\mathcal{T}$, each node corresponds to GPIS_1 queries with smaller and smaller sets. The nodes of the tree are labeled with either 0 or 1. If $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) = 0$, we label the root with 0 and terminate. Otherwise, we label the root (node) with 1 and keep branching, and as long as there is a leaf node $(B_1^{[b_1]}, \ldots, B_t^{[b_t]})$ in this branch labeled with 1, we do the following. Note that $0 \leq b_i \leq d$ and $\sum_{i=1}^{t} b_i = d$. Also, for $i \neq j$, either $B_i \cap B_j = \emptyset$ or $B_i = B_j$ because it is a tree-based disjoint partitioning.

Algorithm 4.1: $\mathcal{A}_{\text{exact}}(A_1, \ldots, A_d, \tau)$

Input: Subsets $A_1, \ldots, A_d$ of vertex set $U(\mathcal{H})$ of a d-uniform hypergraph $\mathcal{H}$ and a threshold τ.

Output: Check if $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \leq \tau$. If Yes, return the exact value of $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$.

Initialize a tree $\mathcal{T}$ with $(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$ as the root and label the root with 1.

if *the number of nodes in* $\mathcal{T}$ *is more than* $2^{d+2}\tau \log n$ **then**

  Report that $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) > \tau$ as the output.

end

while *there is a leaf node* $m_o(B_1^{[b_1]}, \ldots, B_t^{[b_t]})$ *in the tree with label* 1 **do**

  Note that $0 \leq b_i \leq d$ and $\sum_{i=1}^{t} b_i = d$. Also, for $i \neq j$, either $B_i \cap B_j = \emptyset$ or $B_i = B_j$.

  if $m_o(B_1^{[b_1]}, \ldots, B_t^{[b_t]}) = 0$ *or there exists an* $i \in [t]$ *such that* $|B_i| < b_i$ **then**

    Label $(B_1^{[b_1]}, \ldots, B_t^{[b_t]})$ with 0.

  end

  else

    Label $(B_1^{[b_1]}, \ldots, B_t^{[b_t]})$ with 1.

    Partition each B_i into two parts B_{i1} and B_{i2} such that $|B_{i1}| = \lceil \frac{B_i}{2} \rceil$ and $|B_{i2}| = \lfloor \frac{B_i}{2} \rfloor$.

    Add nodes of the form $(C_{11j}, \ldots, C_{1b_1j}, \ldots, C_{t1j}, \ldots, C_{tb_tj})$ as children of $(B_1^{[b_1]}, \ldots, B_t^{[b_t]})$ and label them 1, for $j \in \{1, 2\}$, where $C_{ib_i1} = \lceil \frac{C_{ib_i}}{2} \rceil$ and $C_{ib_i2} = \lfloor \frac{C_{ib_i}}{2} \rfloor$.

  end

end

Generate $\mathcal{T}'$ as the tree after deleting all the leaf nodes in $\mathcal{T}$.

Report the number of leafs in $\mathcal{T}'$ as $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$ as the output.

(i) If $m_o(B_1^{[b_1]}, \ldots, B_t^{[b_t]}) = 0$ or there exists an $i \in [t]$ such that $|B_i| < b_i$, then we label $(B_1^{[b_1]}, \ldots, B_t^{[b_t]})$ with 0. Otherwise, we label $(B_1^{[b_1]}, \ldots, B_t^{[b_t]})$ with 1.

(ii) We partition each B_i into two parts B_{i1} and B_{i2} such that $|B_{i1}| = \left\lceil \frac{B_i}{2} \right\rceil$ and $|B_{i2}| = \left\lfloor \frac{B_i}{2} \right\rfloor$. There may exist some B_i for which $B_{i2} = \emptyset$. We add nodes of the form $(\underbrace{C_{11j}, \ldots, C_{1b_1j}}, \ldots, \underbrace{C_{t1j}, \ldots, C_{tb_tj}})$ as the children of $(B_1^{[b_1]}, \ldots, B_t^{[b_t]})$ and label them 1, for $j \in \{1, 2\}$ where $C_{ib_i1} = \left\lceil \frac{C_{ib_i}}{2} \right\rceil$ and $C_{ib_i2} = \left\lfloor \frac{C_{ib_i}}{2} \right\rfloor$. Note that for

each node $(B_1^{[b_1]}, \ldots, B_t^{[b_t]})$ with label 1, we add $\prod_{i=1}^{t} 2^{b_i} = 2^d$ nodes as children of it.

Note that the algorithm stops when all the leaf nodes of tree $\mathcal{T}$ are labeled with 0. Let $\mathcal{T}'$ be the tree after deleting all the leaf nodes in $\mathcal{T}$. Observe that $m_o(B_1^{[b_1]}, \ldots, B_t^{[b_t]})$ is the number of leaf nodes in $\mathcal{T}'$ and

- the height of $\mathcal{T}$ is bounded by $\log\left(\max_{i\in[t]} |A_i|\right) + 1 \leq \log n + 1$.
- the query complexity of the above procedure is bounded by the number of nodes in $\mathcal{T}$ as we make at most one query per node of $\mathcal{T}$.

The number of nodes in $\mathcal{T}'$ which equals the number of internal nodes of $\mathcal{T}$ is bounded by $(\log n + 1) \cdot m_o(B_1^{[b_1]}, \ldots, B_t^{[b_t]})$. Hence, the number of leaf nodes in $\mathcal{T}$ is at most $2^d(\log n + 1) \cdot m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$. The total number of nodes in $\mathcal{T}$ is at most $(2^d + 1)(\log n + 1) \cdot m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \leq 2^{d+2} \cdot m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \log n$. Putting everything together, the number of GPIS_1 queries made by our algorithm is at most $2^{d+2} \cdot m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \log n$.

The algorithm, as claimed in the statement of Lemma 4.14, proceeds similar to the one presented above by initializing a tree $\mathcal{T}$ with $(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$ as the root. If $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \leq \tau$, then we can find the exact value of $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$ using at most $2^{d+2}\tau \log n$ GPIS_1 queries and the number of nodes in $\mathcal{T}$ is bounded by $2^{d+2}\tau \log n$. So, if the number of nodes in $\mathcal{T}$ is more than $2^{d+2} \cdot m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \log n$ at any instance during the execution of the algorithm, we report $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) > \tau$ and terminate. Hence, our algorithm makes $\mathcal{O}_d(\tau \log n)$ GPIS_1 queries, decides whether $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \leq \tau$, and determines the exact value of $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$ in the case $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) \leq \tau$. □

4.6 Proof of lemma for coarse estimation

We now prove Lemma 4.8. The algorithm corresponding to Lemma 4.8 is Algorithm 4.3 (named COARSE-ESTIMATE). Algorithm 4.2 (named VERIFY-ESTIMATE) is a subrou-

tine of Algorithm 4.3. Algorithm 4.2 determines whether a given estimate $\widehat{R}$ is correct up to $\mathcal{O}_d(\log^{2d-3} n)$ factor. Lemma 4.15 and 4.16 are intermediate results needed to prove Lemma 4.8; they bound the probability from above and below, respectively of VERIFY-ESTIMATE accepting the estimate $\widehat{\mathcal{R}}$.

Lemma 4.15. *If* $\widehat{\mathcal{R}} \geq 20d^{2d-3}4^d\, m_o(A_1, \ldots, A_d) \log^{2d-3} n$, *then*

$$\mathbb{P}(\text{VERIFY-ESTIMATE } (A_1, \ldots, A_d, \widehat{\mathcal{R}}) \textit{ accepts the estimate } \widehat{R}) \leq \frac{1}{20 \cdot 2^d}$$

Proof. Consider the set of ordered hyperedges $\mathcal{F}_o(A_1, \ldots, A_d)$ in $\mathcal{H}(A_1, \ldots, A_d)$. Algorithm VERIFY-ESTIMATE taking parameters $A_1, \ldots, A_d$, and $\widehat{\mathcal{R}}$ and described in Algorithm 4.2, loops over all possible $\mathbf{j} = (j_1, \ldots, j_{d-1}) \in [(d \log n)^*]^{d-1}$ [8]. For each $\mathbf{j} = (j_1, \ldots, j_{d-1}) \in [(d \log n)^*]^{d-1}$, VERIFY-ESTIMATE $(A_1, \ldots, A_d, \widehat{\mathcal{R}})$ samples vertices in each A_i with *suitable* probability values $p(i, \mathbf{j})$, depending on $\mathbf{j}$, $\widehat{R}$, d and $\log n$, to generate the sets $B_{i,\mathbf{j}}$ for $1 \leq i \leq d$. See Algorithm 4.2 for the exact values of $p(i, \mathbf{j})$'s. VERIFY-ESTIMATE $(A_1, \ldots, A_d, \widehat{\mathcal{R}})$ reports ACCEPT if there exists one $\mathbf{j} \in [(d \log n)^*]^{d-1}$ such that $m_o\left(B_{1,\mathbf{j}}, \ldots, B_{d,\mathbf{j}}\right) \neq 0$. Otherwise, REJECT is reported by VERIFY-ESTIMATE $(A_1, \ldots, A_d, \widehat{\mathcal{R}})$.

For an ordered hyperedge $F_o \in \mathcal{F}_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]})$ and $\mathbf{j} \in [(d \log n)^*]^{d-1}$, let $X_{F_o}^{\mathbf{j}}$ denote the indicator random variable such that $X_{F_o}^{\mathbf{j}} = 1$ if and only if $F_o \in \mathcal{F}_o(B_{1,\mathbf{j}}, \ldots, B_{d,\mathbf{j}})$. Let $X_{\mathbf{j}} = \sum_{F_o \in \mathcal{F}_o(A_1, \ldots, A_d)} X_{F_o}^{\mathbf{j}}$. Note that $m_o(B_{1,\mathbf{j}}, \ldots, B_{d,\mathbf{j}}) = X_{\mathbf{j}}$. We have,

$$\mathbb{P}\left(X_{F_o}^{\mathbf{j}} = 1\right) = \prod_{i=1}^{d} (p(i, \mathbf{j}))\ [9] \leq \frac{2^{j_1}}{\widehat{\mathcal{R}}} \cdot \frac{2^{j_2}}{2^{j_1}} d \log n \cdot \cdots \cdot \frac{2^{j_{d-1}}}{2^{j_{d-2}}} d \log n \cdot \frac{1}{2^{j_{d-1}}} = \frac{d^{d-2} \log^{d-2} n}{\widehat{\mathcal{R}}}$$

Then, $\mathbb{E}\left[X_{\mathbf{j}}\right] \leq \frac{m_o(A_1, \ldots, A_d)}{\widehat{\mathcal{R}}} d^{d-2} \log^{d-2} n$, and since $X_{\mathbf{j}} \geq 0$,

$$\mathbb{P}\left(X_{\mathbf{j}} \neq 0\right) = \mathbb{P}(X_{\mathbf{j}} \geq 1) \leq \mathbb{E}\left[X_{\mathbf{j}}\right] \leq \frac{m_o(A_1, \ldots, A_d)}{\widehat{\mathcal{R}}} d^{d-2} \log^{d-2} n$$

[8] Recall that $[n]^*$ denotes the set $\{0, \ldots, n\}$

[9] See Algorithm 4.2 for the values of $p(i, \mathbf{j})$'s

Algorithm 4.2: VERIFY-ESTIMATE $(A_1, \ldots, A_d, \widehat{\mathcal{R}})$

Input: d subsets $A_1, \ldots, A_d$ of the vertex set $U(\mathcal{H})$ of a d-uniform hypergraph $\mathcal{H}$ and an estimate $\widehat{\mathcal{R}}$

Output: If $\widehat{\mathcal{R}}$ is a *good*[7] estimate, then ACCEPT. Otherwise, REJECT

for *($j_1 = d \log n$ to 0)* **do**
  find $B_1 \subseteq A_1$ by sampling every element of A_1 with probability $p_1 = \min\{\frac{2^{j_1}}{\widehat{\mathcal{R}}}, 1\}$ independently of other elements.
  for *($j_2 = d \log n$ to 0)* **do**
    find $B_2 \subseteq A_2$ by sampling every element of A_2 with probability $p_2 = \min\{2^{j_2 - j_1} \cdot d \log n, 1\}$ independently of other elements.
    $\vdots$
    for *($j_{d-1} = d \log n$ to 0)* **do**
      find $B_{d-1} \subseteq A_{d-1}$ by sampling every element of A_{d-1} with probability $p_{d-1} = \min\{2^{j_{d-1} - j_{d-2}} \cdot d \log n, 1\}$ independently of other elements.
      Let $\mathbf{j} = (j_1, \ldots, j_{d-1}) \in [(d \log n)^*]^{d-1}$
      Let $p(i, \mathbf{j}) = p_i$, where $1 \leq i \leq d-1$
      Let $B(i, \mathbf{j}) = B_i$, where $1 \leq i \leq d-1$
      find $B(d, \mathbf{j}) = B_d \subseteq A_d$ by sampling every element of A_d with probability $p_d = \min\{2^{-j_{d-1}}, 1\}$ independently of other elements.
      if $(m(B_{1,\mathbf{j}}, \ldots, B_{d,\mathbf{j}}) \neq 0)$ **then**
        ACCEPT /*[Note that GPIS$_2$ query is called in the above line.]*/
      end
    end
  end
end
REJECT

Now, using the fact that $\widehat{\mathcal{R}} \geq 20d^{2d-3} \cdot 4^d \cdot m_o(A_1, \ldots, A_d) \log^{2d-3} n$, we have

$$\mathbb{P}\left(X_{\mathbf{j}} \neq 0\right) \leq \frac{1}{20d^{d-1} \cdot 4^d \cdot \log^{d-1} n}.$$

Recall that VERIFY-ESTIMATE accepts if and only if there exists $\mathbf{j}$ such that $X_{\mathbf{j}} \neq$

0 [10]. Using the union bound, we get

$$\begin{aligned}
&\mathbb{P}(\text{VERIFY-ESTIMATE } (A_1, \ldots, A_d, \widehat{\mathcal{R}}) \text{ accepts the estimate } \widehat{R}) \\
\leq &\sum_{\mathbf{j} \in [(d \log n)^*]^{d-1}} \mathbb{P}(X_{\mathbf{j}} \neq 0) \\
\leq &\frac{(d \log n + 1)^{d-1}}{20 \cdot 4^d \cdot (d \log n)^{d-1}} \\
\leq &\frac{1}{20 \cdot 2^d}.
\end{aligned}$$

□

Lemma 4.16. *If* $\widehat{\mathcal{R}} \leq \frac{m_o(A_1, \ldots, A_d)}{4d \log n}$,

$$\mathbb{P}(\text{VERIFY-ESTIMATE } (A_1, \ldots, A_d, \widehat{\mathcal{R}}) \textit{ accepts the estimate } \widehat{R}) \geq \frac{1}{2^d}.$$

Proof. To build up towards the proof of Lemma 4.16, we first define some quantities and prove Claim 4.17. For that, let us think of partitioning the vertex set in A_1 into buckets such that the vertices in each bucket are present in approximately the same number of hyperedges in $\mathcal{H}(A_1, \ldots, A_d)$ as the first vertex. Similarly, we extend the bucketing idea to tuples as follows. Consider a vertex u_1 in a particular bucket of A_1 and consider all the hyperedges $\mathcal{F}(u_1)$ containing u_1 as the first vertex. We can bucket the vertices in A_2 such that the vertices in each bucket of A_2 are present in approximately the same number of hyperedges in $\mathcal{F}(u_1)$ as the second vertex. We generalize the above bucketing strategy with the vertices in A_is, which is formally described below. Notice that this way of bucketing will allow us to use conditionals.

For $q_1 \in [(d \log n)^*]$ [11], let $A_1(q_1) \subseteq A_1$ be the set of vertices in A_1 such that for each $u_1 \in A_1(q_1)$, the number of hyperedges in $\mathcal{F}_o(A_1, \ldots, A_d)$, containing u_1 as the first vertex, lies between 2^{q_1} and $2^{q_1+1} - 1$. For $2 \leq i \leq d-1$, and $q_j \in [(d \log n)^*]$ $\forall j \in [i-1]$, consider $u_1 \in A_1(q_1), u_2 \in A_2((q_1, u_1), q_2), \ldots, u_{i-1} \in A_{i-1}((q_1, u_1), \ldots, (q_{i-2}, u_{i-2}), q_{i-1})$. Let $A_i((q_1, u_1), \ldots, (q_{i-1}, u_{i-1}), q_i)$ be the set of

[10] Note that $\mathbf{j}$ is a vector but $X_{\mathbf{j}}$ is a scalar.

[11] Recall that $[n]^* = \{0, 1, \ldots, n\}$

vertices in A_i such that for each $u_i \in A_i((q_1,u_1),\ldots,(q_{i-1},u_{i-1}),q_i)$, the number of ordered hyperedges in $\mathcal{F}_o(A_1,\ldots,A_d)$, containing u_j as the j-th vertex for all $j \in [i]$, lies between 2^{q_i} and $2^{q_i+1}-1$. We need the following result to proceed further. For ease of presentation, we use (Q_i,U_i) to denote $(q_1,u_1),\ldots,(q_{i-1},u_{i-1})$ for $2 \leq i \leq d-1$. Informally, Claim 4.17 says that for each $i \in [d-1]$, there exists a bucket in A_i having a *large* number of vertices contributing approximately the same number of hyperedges.

Claim 4.17. *(i) There exists $q_1 \in [(d\log n)^*]$ such that $|A_1(q_1)| > \frac{m_o(A_1,\ldots,A_d)}{2^{q_1+1}(d\log n+1)}$.*

(ii) Let $2 \leq i \leq d-1$ and $q_j \in [(d\log n)^*]$ $\forall j \in [i-1]$. Let $u_1 \in A_1(q_1)$, $u_j \in A_j((Q_{j-1},U_{j-1}),q_j)$ $\forall j \neq 1$ and $j < i$. There exists $q_i \in [(d\log n)^*]$ such that $|A_i((Q_i,U_i),q_i)| > \frac{2^{q_{i-1}}}{2^{q_i+1}(d\log n+1)}$.

Proof. (i) Observe that $m_o(A_1,\ldots,A_d) = \sum_{q_1=0}^{d\log n} m_o(A_1(q_1),A_2,\ldots,A_d)$. So, there exists $q_1 \in [(d\log n)^*]$ such that $m_o(A_1(q_1),A_2,\ldots,A_d) \geq \frac{m_o(A_1,\ldots,A_d)}{d\log n+1}$. From the definition of $A_1(q_1)$, $m_o(A_1(q_1),A_2,\ldots,A_d) < |A_1(q_1)|\cdot 2^{q_1+1}$. Hence, there exists $q_1 \in [(d\log n)^*]$ such that

$$|A_1(q_1)| > \frac{m_o(A_1(q_1),A_2,\ldots,A_d)}{2^{q_1+1}} \geq \frac{m_o(A_1,\ldots,A_d)}{2^{q_1+1}(d\log n+1)}.$$

(ii)

$$\begin{aligned} \text{Note that} \quad & m_o(\{u_1\},\ldots,\{u_{i-1}\},A_i,\ldots,A_d) \\ = & \sum_{q_i=0}^{d\log n} m_o(\{u_1\},\ldots,\{u_{i-1}\},A_i((Q_{i-1},U_{i-1}),q_i),\ldots,A_d). \end{aligned}$$

So, there exists $q_i \in [(d\log n)^*]$ such that

$$\begin{aligned} m_o(\{u_1\},\ldots,\{u_{i-1}\},A_i((Q_{i-1},U_{i-1}),q_i),\ldots,A_d) \\ \geq \frac{m_o(\{u_1\},\ldots,\{u_{i-1}\},A_i,\ldots,A_d)}{d\log n+1}. \end{aligned}$$

From the definition of $A_i((Q_{i-1}, U_{i-1}), q_i)$, we have

$$m_o(\{u_1\}, \ldots, \{u_{i-1}\}, A_i((Q_{i-1}, U_{i-1}), q_i), \ldots, A_d) < |A_i((Q_{i-1}, U_i), q_i)| \cdot 2^{q_i+1}$$

Hence, there exists $q_i \in [(d \log n)^*]$ such that

$$\begin{aligned} |A_i((Q_{i-1}, U_i), q_i)| &> \frac{m_o(\{u_1\}, \ldots, \{u_{i-1}\}, A_i((Q_{i-1}, U_{i-1}), q_i), \ldots, A_d\})}{2^{q_i+1}} \\ &\geq \frac{m_o(\{u_1\}, \ldots, \{u_{i-1}\}, A_i, \ldots, A_d\})}{2^{q_i+1}(d \log n + 1)} \\ &\geq \frac{2^{q_{i-1}}}{2^{q_i+1}(d \log n + 1)} \end{aligned}$$

□

Coming back to the proof of Lemma 4.16, we will be done by showing the following. VERIFY-ESTIMATE accepts with probability at least $1/5$ when the loop variables $j_1, \ldots, j_{d-1}$ respectively attain values $q_1, \ldots, q_{d-1}$ such that $|A_1(q_1)| > \frac{m_o(A_1, \ldots, A_d)}{2^{q_1+1}(d \log n+1)}$ and $|A_i((Q_i, U_i), q_i)| > \frac{2^{q_{i-1}}}{2^{q_i+1}(d \log n+1)}$ $\forall i \in [d-1] \setminus \{1\}$. The existence of such j_is is evident from Claim 4.17. Let $\mathbf{q} = (q_1, \ldots, q_{d-1})$. Recall that $B_{i,\mathbf{q}} \subseteq A_i$ is the sample obtained when the loop variables $j_1, \ldots, j_{d-1}$ attain values $q_1, \ldots, q_{d-1}$, respectively. Let $\mathcal{E}_i, i \in [d-1]$, be the events defined as follows.

- $\mathcal{E}_1$: $A_1(q_1) \cap B_{1,\mathbf{q}} \neq \emptyset$.
- $\mathcal{E}_i$: $A_j((Q_{j-1}, U_{j-1}), q_j) \cap B_{j,\mathbf{q}} \neq \emptyset$, where $2 \leq i \leq d-1$.

As noted earlier, Claim 4.17 says that for each $i \in [d-1]$, there exists a bucket in A_i having a *large* number of vertices contributing approximately the same number of hyperedges. The above events correspond to the nonempty intersection of vertices in heavy buckets corresponding to A_i and the sampled vertices $B_{i,\mathbf{j}}$, where $i \in [d-1]$. Observe that

$$\begin{aligned} \mathbb{P}(\overline{\mathcal{E}_1}) \leq \left(1 - \frac{2^{q_1}}{\widehat{\mathcal{R}}}\right)^{|A_1(q_1)|} \leq \exp\left(-\frac{2^{q_1}}{\widehat{\mathcal{R}}} |A_1(q_1)|\right) &\leq \exp\left(-\frac{2^{q_1}}{\widehat{\mathcal{R}}} \cdot \frac{m_o(A_1, \ldots, A_d)}{2^{q_1+1}(d \log n + 1)}\right) \\ &\leq \exp(-1). \end{aligned}$$

The last inequality uses the fact that $\widehat{\mathcal{R}} \leq \frac{m_o(A_1,\ldots,A_d)}{4d\log n}$, from the condition of the lemma. Assume that $\mathcal{E}_1$ occurs and $u_1 \in A_1(q_1) \cap B_{1,\mathbf{q}}$. We will bound the probability that $A_2(Q_1, U_1), q_2) \cap A_{2,\mathbf{q}} = \emptyset$, that is $\overline{\mathcal{E}_2}$. Note that, by Claim 4.17 (ii), $|A_2(Q_1, U_1), q_2)| \geq \frac{2^{q_1}}{2^{q_2+1}(d\log n+1)}$. So,

$$\mathbb{P}\left(\overline{\mathcal{E}_2} \mid \mathcal{E}_1\right) \leq \left(1 - \frac{2^{q_2}}{2^{q_1}}\log n\right)^{|A_2(Q_1,U_1),q_2)|} \leq \exp\left(-1\right)$$

Assume that $\mathcal{E}_1, \ldots, \mathcal{E}_{i-1}$ hold, where $3 \leq i \in [d-1]$. Let $u_1 \in A_1(q_1)$ and $u_{i-1} \in A_{i-1}((Q_{i-2}, U_{i-2}), q_{i-1})$. We will bound the probability that $A_i((Q_{i-1}, U_{i-1}), q_i) \cap B_{i,\mathbf{q}} = \emptyset$, that is $\overline{\mathcal{E}_i}$. Note that $|A_i((Q_{i-1}, U_{i-1}), q_i)| \geq \frac{2^{q_{i-1}}}{2^{q_i+1}(d\log n+1)}$. So, for $3 \leq i \in [d-1]$,

$$\mathbb{P}\left(\overline{\mathcal{E}_i} \mid \mathcal{E}_1, \ldots, \mathcal{E}_{i-1}\right) \leq \left(1 - \frac{2^{q_i}}{2^{q_{i-1}}}\log n\right)^{|A_i(Q_{i-1},U_{i-1}),q_i)|} \leq \exp\left(-1\right)$$

Assume that $\mathcal{E}_1, \ldots, \mathcal{E}_{d-1}$ hold. Let $u_1 \in A_1(q_1)$ and $u_{i-1} \in A_{i-1}((Q_{i-2}, U_{i-2}), q_{i-1})$ for all $i \in [d] \setminus \{1\}$. Let $S \subseteq A_d$ be the set of d-th vertex of the ordered hyperedges in $\mathcal{F}_o(A_1, \ldots, A_d)$ having u_j as the j-th vertex for all $j \in [d-1]$. Note that $|S| \geq 2^{q_{d-1}}$. Let $\mathcal{E}_d$ be the event that represents the fact $S \cap B_{d,\mathbf{q}} \neq \emptyset$. So,

$$\mathbb{P}(\overline{\mathcal{E}_d} \mid \mathcal{E}_1, \ldots, \mathcal{E}_{d-1}) \leq \left(1 - \frac{1}{2^{q_{d-1}}}\right)^{q_{d-1}} \leq \exp\left(-1\right)$$

Observe that VERIFY-ESTIMATE accepts if $m(A_{B,\mathbf{q}}, \ldots, B_{d,\mathbf{q}}) \neq 0$. Also,

$$m(B_{1,\mathbf{q}}, \ldots, B_{d,\mathbf{q}}) \neq 0 \text{ if } \bigcap_{i=1}^{d} \mathcal{E}_i \text{ occurs.}$$

Hence,

$$\begin{aligned}\mathbb{P}(\text{VERIFY-ESTIMATE } (A_1,\ldots,A_d,\widehat{\mathcal{R}}) \text{ accepts}) &\geq \mathbb{P}\left(\bigcap_{i=1}^{d}\mathcal{E}_i\right)\\ &= \mathbb{P}(\mathcal{E}_1)\prod_{i=2}^{d}\mathbb{P}(\mathcal{E}_i \mid \bigcap_{j=1}^{i-1}\mathcal{E}_j)\\ &> (1-\exp(-1))^d \geq \frac{1}{2^d}\end{aligned}$$

□

Now, we will prove Lemma 4.8 that will be based on Algorithm 4.3. We restate the lemma.

Lemma 4.18 (Coarse estimation : Lemma 4.8 restated)**.** *There exists an algorithm $\mathcal{A}$ that takes as input d subsets $A_1,\ldots,A_d$ of the vertex set $U(\mathcal{H})$ of a d-uniform hypergraph $\mathcal{H}$. The algorithm $\mathcal{A}$ returns $\widehat{E}$ as an estimate for $m_o(A_1,\ldots,A_d)$ such that*

$$\frac{m_o(A_1,\ldots,A_d)}{8d^{d-1}2^d\log^{d-1}n} \leq \widehat{E} \leq 20d^{d-1}2^d\cdot m_o(A_1,\ldots,A_d)\log^{d-1}n$$

with probability $1-n^{-8d}$. Moreover, the number of GPIS_2 *queries made by the algorithm is $\mathcal{O}_d(\log^{d+1}n)$.*

Algorithm 4.3: COARSE-ESTIMATE $(A_1,\ldots,A_d)$

Input: d subsets $A_1,\ldots,A_d \subseteq U(\mathcal{H})$.

Output: An estimate $\widehat{E}$ for $m_o(A_1,\ldots,A_d)$.

1 **for** ($\widehat{\mathcal{R}} = n^d, n^d/2,\ldots,1$) **do**

2 Repeat VERIFY-ESTIMATE $(A_1,\ldots,A_d,\widehat{\mathcal{R}})$ for $\Gamma = d\cdot 4^d\cdot 2000\log n$ times. If more than $\frac{\Gamma}{10\cdot 2^d}$ runs of VERIFY-ESTIMATE accepts, then output $\widehat{E} = \frac{\widehat{\mathcal{R}}}{d^{d-2}\cdot 2^d}$.

Proof. Note that an execution of COARSE-ESTIMATE for a particular $\widehat{\mathcal{R}}$ repeats VERIFY-ESTIMATE for $\Gamma = d\cdot 4^d\cdot 2000\log n$ times and gives output $\widehat{\mathcal{R}}$ if more than $\frac{\Gamma}{10\cdot 2^d}$ runs of

VERIFY-ESTIMATE accepts. For a particular $\widehat{\mathcal{R}}$, let X_i be the indicator random variable such that $X_i = 1$ if and only if the i-th execution of VERIFY-ESTIMATE accepts. Also take $X = \sum_{i=1}^{\Gamma} X_i$. COARSE-ESTIMATE gives output $\widehat{\mathcal{R}}$ if $X > \frac{\Gamma}{10 \cdot 2^d}$.

Consider the execution of COARSE-ESTIMATE for a particular $\widehat{\mathcal{R}}$. If $\widehat{\mathcal{R}} \geq 20d^{2d-3}4^d \cdot m_o(A_1, \ldots, A_d) \cdot \log^{2d-3} n$, we first show that COARSE-ESTIMATE does not accept with high probability. Recall Lemma 4.15. If $\widehat{\mathcal{R}} \geq 20d^{2d-3}4^d \cdot m_o(A_1, \ldots, A_d) \log^{2d-3} n$, $\mathbb{P}(X_i = 1) \leq \frac{1}{20 \cdot 2^d}$ and hence $\mathbb{E}[X] \leq \frac{\Gamma}{20 \cdot 2^d}$. By using Chernoff-Hoeffding's inequality (See Lemma 2.4 (i) in Section 2.1),

$$\mathbb{P}\left(X > \frac{\Gamma}{10 \cdot 2^d}\right) = \mathbb{P}\left(X > \frac{\Gamma}{20 \cdot 2^d} + \frac{\Gamma}{20 \cdot 2^d}\right) \leq \frac{1}{n^{10d}}$$

Using the union bound for all $\widehat{\mathcal{R}}$, the probability that COARSE-ESTIMATE outputs some $\widehat{E} = \frac{\widehat{\mathcal{R}}}{d^{d-2} \cdot 2^d}$ such that $\widehat{\mathcal{R}} \geq 20d^{2d-3}4^d \cdot m_o(A_1, \ldots, A_d) \log^{2d-3} n$, is at most $\frac{d \log n}{n^{10}}$. Now consider the instance when the for loop in COARSE-ESTIMATE executes for a $\widehat{\mathcal{R}}$ such that $\widehat{\mathcal{R}} \leq \frac{m_o(A_1,\ldots,A_d)}{4d \log n}$. In this situation, $\mathbb{P}(X_i = 1) \geq \frac{1}{2^d}$. So, $\mathbb{E}[X] \geq \frac{\Gamma}{2^d}$. By using Chernoff-Hoeffding's inequality (See Lemma 2.4 (ii) in Section 2.1),

$$\mathbb{P}\left(X \leq \frac{\Gamma}{10 \cdot 2^d}\right) \leq \mathbb{P}\left(X < \frac{\Gamma}{2^d} - \frac{4}{5} \cdot \frac{\Gamma}{2^d}\right) \leq \frac{1}{n^{100d}}$$

By using the union bound for all $\widehat{\mathcal{R}}$, the probability that COARSE-ESTIMATE outputs some $\widehat{E} = \frac{\widehat{\mathcal{R}}}{d^{d-2} \cdot 2^d}$ such that $\widehat{\mathcal{R}} \leq \frac{m_o(A_1,\ldots,A_d)}{4d \log n}$, is at most $\frac{d \log n}{n^{100d}}$. Observe that, the probability that COARSE-ESTIMATE outputs some $\widehat{E} = \frac{\widehat{\mathcal{R}}}{d^{d-2} \cdot 2^d}$ such that $\widehat{\mathcal{R}} \geq d^{2d-3}4^d m_o(A_1, \ldots, A_d) \log^{2d-3} n$ or $\widehat{\mathcal{R}} \leq \frac{m_o(A_1,\ldots,A_d)}{4d \log n}$, is at most $\frac{d \log n}{n^{10d}} + \frac{d \log n}{n^{100d}} \leq \frac{1}{n^{8d}}$. Putting everything together, COARSE-ESTIMATE gives some $\widehat{E} = \frac{\widehat{\mathcal{R}}}{d^{d-2} \cdot 2^d}$ as the output with probability at least $1 - \frac{1}{n^{8d}}$ satisfying

$$\frac{m_o(A_1, \ldots, A_d)}{8d^{d-1}2^d \log^{d-1} n} \leq \widehat{E} = \frac{\widehat{\mathcal{R}}}{d^{d-2} \cdot 2^d} \leq 20d^{d-1}2^d \cdot m_o(A_1, \ldots, A_d) \log^{d-1} n$$

From the pseudocode of VERIFY-ESTIMATE (Algorithm 4.2), we call for GPIS_2 queries only at line number 12. In the worst case, VERIFY-ESTIMATE executes line

number 12 for each $\mathbf{j} \in [(d \log n)^*]$. That is, the query complexity of VERIFY-ESTIMATE is $\mathcal{O}(\log^{d-1} n)$. From the description of COARSE-ESTIMATE, COARSE-ESTIMATE calls VERIFY-ESTIMATE $\mathcal{O}_d(\log n)$ times for each choice of $\widehat{R}$. Hence, COARSE-ESTIMATE makes $\mathcal{O}_d(\log^{d+1} n)$ GPIS$_2$ queries. □

4.7 Algorithm

In this section, we describe our algorithm for a $(1 \pm \varepsilon)$ multiplicative approximation of the number of hyperedges $m_o(\mathcal{H})$ in a hypergraph $\mathcal{H}$. As mentioned earlier, when $\varepsilon \leq \left(n^{-d} \log^{5d+5} n\right)^{1/4}$, we compute $m_o(\mathcal{H})$ exactly by querying $m_o(\{a_1\}, \ldots, \{a_d\})$ for all distinct $a_1, \ldots, a_d \in U(\mathcal{H})$, and this requires only polylogarithmic number of queries. We do the following when $\varepsilon > \left(n^{-d} \log^{5d+5} n\right)^{1/4}$. We build a data structure $\mathcal{D}$ that maintains the following two quantities at any point of time.

(i) An accumulator Ψ for the number of hyperedges. We initialize $\Psi = 0$.

(ii) A set of tuples $(A_{11}, \ldots, A_{1d}, w_1), \ldots, (A_{\zeta 1}, \ldots, A_{\zeta d}, w_\zeta)$ for some $\zeta > 0$, where tuple $(A_{i1}, \ldots, A_{id})$ corresponds to the d-partite subgraph $\mathcal{H}(A_{i1}, \ldots, A_{id})$ and w_i is the weight associated to $\mathcal{H}(A_{i1}, \ldots, A_{id})$.

The data structure is initialized with $\Psi = 0$, and only one tuple $(U^{[d]}, 1)$. The Algorithm performs the following steps.

(1) When there are no tuples left in the data structure $\mathcal{D}$, the Algorithm outputs Ψ as the estimate.

(2) **(Exact Counting)** Fix a threshold τ as $\frac{k^2 \cdot 4^{2d} \theta^{2d} \cdot 16 d^2 \cdot d! \log^{d+2} n}{\varepsilon^2}$, where $k = 4$ [12]. For each tuple $(A_1, \ldots, A_d, w)$ in $\mathcal{D}$, decide whether $m_o(A_1, \ldots, A_d) \leq \tau$ using Lemma 4.6. If $m_o(A_1, \ldots, A_d) \leq \tau$, we add the weighted number of ordered hyperedges $w \cdot m_o(A_1, \ldots, A_d)$ to Ψ and remove $(A_1, \ldots, A_d, w)$ from $\mathcal{D}$. If there are no tuples left in $\mathcal{D}$, then go to Step 1. Otherwise, if the number of tuples is

[12] We take $k = 4$ for ease of calculations. The argument works for any k.

at most $\mathcal{N} = \kappa_d \cdot \frac{\log^{4d} n}{\varepsilon^2}$, then go to Step 3, else go to Step 4. Note that κ_d is a constant to be fixed later. By Lemma 4.6, the query complexity of Step 2 is $\mathcal{O}_d(\tau \log n) = \mathcal{O}_d\left(\frac{\log^{d+3} n}{\varepsilon^2}\right)$ per tuple.

(3) **(Sparsification)** For any tuple $(A_1^{[a_1]}, \ldots, A_s^{[a_s]}, w)$ here we have $m_o(A_1^{[a_1]}, \ldots, A_s^{[a_s]}) > \tau$. Note that A_i and A_j are pairwise disjoint subsets for each $1 \leq i < j \leq s$. We take $h_d : [k]^d \to \{0, 1\}$ to be a hash function such that $\mathbb{P}\left(h_d(\mathbf{a} = 1)\right) = 1/k$ independently for each tuple $\mathbf{a} \in [k]^d$. The vertices in $A = \bigcup_{i=1}^{s} A_i$ are COLORED with $[k] = [4]$, and let $\chi(i, j) = \{v \in A_i : v \text{ is COLORED with color } j\}$, where $i \in [s]$ and $j \in [k]$.

We add each tuple $(\chi(1, c_1), \ldots, \chi(1, c_{a_1}), \ldots, \chi(s, c_{d-a_s+1}), \ldots, \chi(s, c_d), 4w)$ to $\mathcal{D}$ for which $h_d(c_1, \ldots, c_d) = 1$. We remove the tuple $(A_1^{[a_1]}, \ldots, A_s^{[a_s]}, w)$ from $\mathcal{D}$. After processing all the tuples, we go to Step 2. Note that no query is required in Step 3. The constant 4 is obtained by putting $k = 4$ in Lemma 4.10.

(4) (**Coarse Estimation**) Let $r > \mathcal{N} = \frac{\kappa_d \log^{4d} n}{\varepsilon^2}$. Let $\{(A_{i1}, \ldots, A_{id}, w_i) : i \in [r]\}$ be the set of tuples stored in $\mathcal{D}$. For each tuple $(A_{i1}, \ldots, A_{id}, w_i)$ in $\mathcal{D}$, we find an estimate $\widehat{E_i}$ such that $\frac{m_o(A_1, \ldots, A_d)}{8 \cdot 2^d d^{d-1} \log^{d-1} n} \leq \widehat{E_i} \leq 20 \cdot 2^d d^{d-1} \log^{d-1} n \cdot m_o(A_1, \ldots, A_d)$. Lemma 4.8 proves that this can be done with $\mathcal{O}_d(\log^{d+1} n)$ GPIS_2 queries per tuple. We know that the number of tuples in $\mathcal{D}$ is more than $\mathcal{N} = \kappa_d \frac{\log^{4d} n}{\varepsilon^2}$. We take a sample from the set of tuples such that the sample maintains the required estimate *approximately* using Lemma 4.19. This lemma follows from a lemma by Beame *et al.* [BHR+18]. The original statement of the lemma by Beame et al. is given in Lemma 2.9 in Section 2.1.

Lemma 4.19. *Let $\{(A_{i1}, \ldots, A_{id}, w_i) : i \in [r]\}$ be the set of tuples in the data structure $\mathcal{D}$ and e_i be the coarse estimation for $m_o(A_{i1}, \ldots, A_{id}, w_i)$ such that*

(i) $w_i, e_i \geq 1 \; \forall i \in [r]$

(ii) $\frac{e_i}{\alpha} \leq m_o(A_{i1}, \ldots, A_{id}) \leq e_i \cdot \alpha$ *for some* $\alpha > 0$ *and* $\forall \; i \in [r]$

(iii) $\sum_{i=1}^{r} w_i \cdot m_o(A_{i1}, \ldots, A_{id}) \leq M$

There exists an algorithm that finds a set $\{(A'_{i1}, \ldots, A'_{id}, w'_i) : i \in [r']\}$ *of tuples such that the above three conditions hold and*

$$\left| \sum_{i=1}^{r'} w'_i \cdot m_o(A'_{i1}, \ldots, A'_{id}) - \sum_{i=1}^{r} w_i \cdot m_o(A_{i1}, \ldots, A_{id}) \right| \leq \lambda S$$

with probability $1 - \delta$ *where* $S = \sum_{i=1}^{r} w_i \cdot m_o(A_{i1}, \ldots, A_{id})$ *and* $\lambda, \delta > 0$*. Also,* $r' = \mathcal{O}\left(\lambda^{-2}\alpha^4 \log M \left(\log\log M + \log \frac{1}{\delta}\right)\right)$.

We use the algorithm for Lemma 4.19 with $\lambda = \frac{\varepsilon}{4d \log n}$, $\alpha = 20 \cdot 2^d d^{d-1} \log^{d-1} n$ and $\delta = \frac{1}{n^{6d}}$ to find a new set $\{(A'_{i1}, \ldots, A'_{id}, w'_i) : i \in [r']\}$ of tuples satisfying the following: $\left| S - \sum_{i=1}^{r'} w'_i m_o(A'_{i1}, \ldots, A'_{id}) \right| \leq \lambda S$ with probability $1 - \frac{1}{n^{6d}}$, where $S = \sum_{i=1}^{r} w_i \cdot m_o(A_{i1}, \ldots, A_{id})$. Here, $r' \leq \kappa_d \cdot \frac{\log^{4d} n}{\varepsilon^2}$. This κ_d is same as the one mentioned in Step 2. We remove the set of r tuples, $r > \mathcal{N}$, from $\mathcal{D}$ and add the set of r' tuples, where $r' \leq \kappa_d \cdot \frac{\log^{4d} n}{\varepsilon^2} = \mathcal{N}$. As no query is required to execute the algorithm of Lemma 4.19, the number of GPIS_2 queries in this step in each iteration, is $\mathcal{O}_d(\log^{d+1} n)$ per tuple.

4.8 Proof of correctness

We start with the following observation for the proof of correctness.

Observation 4.20. There are at most $4^d \cdot \mathcal{N} = 4^d \kappa_d \cdot \frac{\log^{4d} n}{\varepsilon^2}$ tuples in the data structure $\mathcal{D}$ during the execution of the algorithm.

Proof. The number of tuples in $\mathcal{D}$ can increase by a factor of 4^d in the sparsification step. Note that we apply the sparsification step only when there are at most $\mathcal{N} = \kappa_2 \cdot \frac{\log^{4d} n}{\varepsilon^2}$ tuples in $\mathcal{D}$. Hence, the number of tuples in $\mathcal{D}$ is at most $4^d \cdot \mathcal{N}$. □

Now we prove Lemma 4.5. We restate the lemma for easy reference.

Lemma 4.21 (Lemma 4.5 restated)**.** *If* $\varepsilon \geq \left(n^{-d} \log^{5d+5} n\right)^{1/4}$*, our algorithm produces a* $(1 \pm \varepsilon)$*-approximation to* $m_o(\mathcal{H})$ *with probability at least* $1 - \frac{1}{n^{4d}}$ *and makes* $\mathcal{O}\left(\frac{\log^{5d+4} n}{\varepsilon^4}\right)$ *queries, where each query is either a* GPIS_1 *query or a* GPIS_2 *query.*

To prove the above lemma, we need the following definition 4.22 along with Observations 4.23 and 4.24.

Definition 4.22. Let TUPLE_i be the set of tuples in $\mathcal{D}$ at the end of the i-th iteration. We partition them into two parts — a set of tuples $\text{TUPLE}_i^{\leq\tau}$ and $\text{TUPLE}_i^{>\tau}$ contributing less than and more than τ ordered hyperedges, respectively. Formally, $\text{TUPLE}_i^{\leq\tau} = \{(A_1, \ldots, A_d, w) : m_o(A_1, \ldots, A_d) \leq \tau\}$ and $\text{TUPLE}_i^{>\tau} = \text{TUPLE}_i \setminus \text{TUPLE}_i^{\leq\tau}$. Let Ψ_i denote the value of Ψ after the i-th iteration where $i \in \mathbb{N}$. The estimate for $m_o(\mathcal{H}) = m_o(U^{[d]})$ after the i-th iteration is given as

$$\text{EST}_i = \Psi_i + \sum_{(A_1,\ldots,A_d,w)\in\text{TUPLE}_i} w \cdot m_o(A_1, \ldots, A_d).$$

The number of active hyperedges after the i-th iteration is given as

$$\text{ACT}_i = \sum_{(A_1,\ldots,A_d,w)\in\text{TUPLE}_i} m_o(A_1, \ldots, A_d).$$

Note that if there are some tuples left in $\mathcal{D}$ at the end of the i-th iteration, we do not know the value of EST_i and ACT_i. However, we know Ψ_i. Observe that $\Psi_0 = 0$ and $\text{EST}_0 = \text{ACT}_0 = m_o(\mathcal{H})$.

Observation 4.23. Let there be only one tuple in $\mathcal{D}$ after the i-th iteration for any non-negative integer i. Then, EST_{i+1} is a $(1+\lambda)$-approximation to EST_i with probability at least $1 - \frac{1}{n^{5d}}$, where $\lambda = \frac{\varepsilon}{4d \log n}$.

Proof. From Definition 4.22,

$$\begin{aligned}
\text{EST}_i &= \Psi_i + \sum_{(A_1,\ldots,A_d,w)\in\text{TUPLE}_i} w \cdot m_o(A_1, \ldots, A_d) \\
&= \Psi_i + \sum_{(A_1,\ldots,A_d,w)\in\text{TUPLE}_i^{\leq\tau}} w \cdot m_o(A_1, \ldots, A_d) + \\
&\qquad \sum_{(A_1,\ldots,A_d,w)\in\text{TUPLE}_i^{>\tau}} w \cdot m_o(A_1, \ldots, A_d)
\end{aligned}$$

In Step 2 (that is the sparsification step) of the algorithm, for each tuple $(A_1, \ldots, A_d, w) \in \text{TUPLE}_i^{\leq \tau}$, we determine the exact value of $m_o(A_1, \ldots, A_d)$, add $w \cdot m_o(A_1, \ldots, A_d)$ to the current value of Ψ and remove the tuple from $\mathcal{D}$. Observe that

$$\Psi_{i+1} - \Psi_i = \sum_{(A_1,\ldots,A_d,w)\in \text{TUPLE}_i^{\leq \tau}} w \cdot m_o(A_1, \ldots, A_d) \tag{4.8}$$

If $\text{TUPLE}_i^{>\tau}$ is empty, we go to Step 1 to report the output. Observe that in that case $\text{EST}_{i+1} = \text{EST}_i$, and we are done. If $\text{TUPLE}_i^{>\tau}$ is non-empty, then we go to either Step 3 (as mentioned in Case 1 below) or Step 4 (as mentioned in Case 2 below) depending on whether the number of tuples in $\mathcal{D}$ is at most $\mathcal{N}$ or more than $\mathcal{N}$, respectively, where $\mathcal{N} = \kappa_d \frac{\log^{4d} n}{\varepsilon^2}$.

Case 1 (Go to Step 3): Note that for each tuple $(A_1, \ldots, A_d, w)$ in $\mathcal{D}$, we have $\text{TUPLE}_i^{\geq \tau}$. We apply the sparsification step (Step 3) for each tuple. For each tuple $(A_1, \ldots, A_d, w)$, we add a set of tuples $\mathcal{Z}$ (each tuple in $\mathcal{Z}$ is of the form $(B_1, \ldots, B_d, 4w)$) by removing $(A_1, \ldots, A_d, w)$ from $\mathcal{D}$. By Lemma 4.7, we have the following with probability $1 - \frac{1}{n^{4\theta-2d}}$.

$$\left| k \sum_{(B_1,\ldots,B_d,4w)\in\mathcal{Z}} m_o(B_1, \ldots, B_d) - m_o(A_1, \ldots, A_d) \right| \leq k 2^{2d} \theta^d \sqrt{d! m_o(A_1, \ldots, A_d) \log^d n}.$$

Note that this is the same k as that mentioned in Step 2 of our algorithm (see Section 4.7). Moreover, $k = 4$.

Now using $m_o(A_1, \ldots, A_d) \geq \tau = \frac{k^2 \cdot 4^{2d} \theta^{2d} \cdot 16 d^2 \cdot d! \log^{d+2} n}{\varepsilon^2}$ and $k = 4$ and taking $\theta = 2d$, we have Equation 4.9 holding with probability $1 - \frac{1}{n^{6d}}$.

$$\left| \sum_{(B_1,\ldots,B_d,4w)\in\mathcal{Z}} 4w \cdot m_o(B_1, \ldots, B_d) - w \cdot m_o(A_1, \ldots, A_d) \right| \leq \frac{\varepsilon}{4d \log n} \cdot w \cdot m_o(A_1, \ldots, A_d) \tag{4.9}$$

Since we are in Step 3, there are at most $\mathcal{N} = \kappa_d \frac{\log^{4d} n}{\varepsilon^2}$ tuples in $\text{TUPLE}_i^{>\tau}$. As

$\varepsilon > \left(\frac{\log^{5d+5}}{n^d}\right)^{1/4}$, the probability that Equation 4.9 holds for each tuple in $\text{TUPLE}_i^{>\tau}$ is at least $1 - \frac{1}{n^{5d}}$.

By Definition 4.22,

$$\text{EST}_{i+1} = \Psi_{i+1} + \sum_{(B_1,\ldots,B_d,w') \in \text{TUPLE}_{i+1}} w' \cdot m_o(B_1, \ldots, B_d)$$

Using Equations 4.8 and 4.9, we can show that EST_{i+1} is a $(1+\lambda)$-approximation to EST_i, where $\lambda = \frac{\varepsilon}{4d \log n}$, and the probability of success is $1 - \frac{1}{n^{5d}}$.

Case 2 **(Go to Step** 4**):** Here, we apply coarse estimation algorithm for each tuple $(A_1, \ldots, A_d, w)$ present in $\mathcal{D}$ to find $\widehat{E}$ such that $\frac{m_o(A_1,\ldots,A_d)}{\alpha} \leq \widehat{E} \leq \alpha \cdot m_o(A_1, \ldots, A_d)$ as described in Step-4. By Lemma 4.8, the probability of success of finding the required coarse estimation for a particular tuple, is at least $1 - \frac{1}{n^{8d}}$. By Observation 4.20, we have at most $4^d\mathcal{N} = \frac{\kappa_d 4^d \log^{4d} n}{\varepsilon^2}$ tuples at any instance of the algorithm. Hence, as $\varepsilon > \left(\frac{\log^{5d+5} n}{n^d}\right)^{1/4}$, the probability that we have the desired coarse estimation for all tuples present in $\mathcal{D}$, is at least $1 - \frac{1}{n^{6d}}$. We have $r > \mathcal{N} = \kappa_d \frac{\log^{4d} n}{\varepsilon^2}$ tuples in $\mathcal{D}$. Under the conditional space that we have the desired coarse estimation for all tuples present in $\mathcal{D}$, we apply the algorithm ALG corresponding to Lemma 4.19. In doing so, we get $r' \leq \mathcal{N}$ tuples, as described in the Step-4, with probability $1 - \frac{1}{n^{6d}}$. Observe that TUPLE_{i+1} is the set of r' tuples returned by ALG satisfying

$$\left| \sum_{(B_1,\ldots,B_d,w')} w' \cdot m_o(B_1, \ldots, B_d) - S \right| \leq \lambda S, \tag{4.10}$$

where $\lambda = \frac{\varepsilon}{4d \log n}$ and $\sum_{(A_1,\ldots,A_d,w) \in \text{TUPLE}_i^{>\tau}} w \cdot m_o(A_1, \ldots, A_d)$. Now, by Definition 4.22,

$$\text{EST}_{i+1} = \Psi_{i+1} + \sum_{(B_1,\ldots,B_d,w') \in \text{TUPLE}_{i+1}} w' \cdot m_o(B_1, \ldots, B_d)$$

Using Equations 4.8 and 4.10, we can show that EST_{i+1} is a $(1+\lambda)$-approximation to EST_i and the probability of success is $1 - \left(\frac{1}{n^{6d}} + \frac{1}{n^{6d}}\right) \geq 1 - \frac{1}{n^{6d-1}}$. □

Observation 4.24. Let there be at least one tuple $(A_1, \ldots, A_d, w)$ in the data structure $\mathcal{D}$ after the i-th iteration such that $m_o(A_1, \ldots, A_d) > \tau$ for any integer $i > 0$. Then, $\text{ACT}_{i+2} \leq \frac{\text{ACT}_i}{2}$ with probability at least $1 - \frac{2}{n^{5d}}$.

Proof. As there exists one tuple in $\text{TUPLE}_i^{>\tau}$, our algorithm will not terminate in Step-2. It will determine the exact values of $m_o(A_1, \ldots, A_d)$ for each $(A_1, \ldots, A_d, w) \in \text{TUPLE}_i^{\leq\tau}$, and then will go to either Step-2 or Step-3 depending on the cardinality of $\text{TUPLE}_i^{\leq\tau}$. By adapting the same approach as that in the proof of Observation 4.23, we can show that

(i) in the $(i+1)$-th iteration, if our algorithm goes to Step-3, then $\text{ACT}_{i+1} \leq \frac{\text{ACT}_i}{2}$ with probability $1 - \frac{1}{n^{5d}}$; and

(ii) in the $(i+1)$-th iteration, if our algorithm goes to Step-4, then $\text{ACT}_{i+1} \leq \text{ACT}_i$ with probability $1 - \frac{1}{n^{6d-1}}$.

From the description of the algorithm, it is clear that we apply sparsification either in iteration $(i+1)$ or $(i+2)$. That is, either we do sparsification in both the iterations, or we do sparsification in one iteration and coarse estimation in the other iteration, or we do sparsification in $(i+1)$-th iteration and termination of the algorithm after executing Step-2 in the $(i+2)$-th iteration. Observe that in the last case, that is, if we terminate in the $(i+2)$-th iteration, then $\text{ACT}_{i+2} = 0 \leq \frac{\text{ACT}_i}{2}$. In the other two cases, by (i) and (ii), we have $\text{ACT}_{i+2} \leq \frac{\text{ACT}_i}{2}$ with probability at least $1 - \frac{2}{n^{5d}}$. □

Now, we are ready to prove Lemma 4.21.

Proof of Lemma 4.21. Let i^* be the largest integer such that there exists at least one tuple $(A_1, \ldots, A_d, w)$ in the data structure $\mathcal{D}$ in the i^*-th iteration such that $m_o(A_1, \ldots, A_d) > \tau$. That is $\text{ACT}_{i^*} > \tau$. For ease of analysis, let us define the two following events.

- $\mathcal{E}_1 : i^* \leq 2d \log n$.
- $\mathcal{E}_2 : \text{EST}_{i^*}$ is a $(1 \pm \varepsilon)$-approximation to $m_o(\mathcal{H})$.

Using the fact $\text{ACT}_0 = m_o(\mathcal{H}) \leq n^d$ along with the Observation 4.24, we have $i^* \leq 2d \log n$ with probability at least $1 - 2d \log n \frac{2}{n^{5d}}$. That is $\mathbb{P}(\mathcal{E}_1) \geq 1 - \frac{4d \log n}{n^{5d}}$.

Now let us condition on the event $\mathcal{E}_1$. By the definition of i^*, we do the following in the $(i^* + 1)$-th iteration. In Step 2, for each tuple $(A_1, \ldots, A_d, w)$ present in $\mathcal{D}$, we determine $m_0(A_1, \ldots, A_d)$ exactly, add it to Ψ and remove $(A_1, \ldots, A_d, w)$ from $\mathcal{D}$. Observe that $\text{ACT}_{i^*+1} = 0$, that is, $\text{EST}_{i^*+1} = \Psi_{i^*+1} = \text{EST}_{i^*}$. Since there are no tuples left in $\mathcal{D}$, we go to Step 1. At the start of the (i^*+2)-th iteration, we report $\Psi_{i^*+1} = \text{EST}_{i^*}$ as the output. Using Observation 4.23, EST_{i^*} is a $(1 \pm \lambda)^{i^*}$-approximation to EST_0 with probability at least $1 - \frac{2d \log n}{n^{5d}}$. As $\text{EST}_0 = m_o(\mathcal{H})$, $\lambda = \frac{\varepsilon}{4d \log n}$, and $\mathcal{E}_1$ has occurred, we have EST_{i^*} is a $(1 \pm \varepsilon)$-approximation to $m_o(\mathcal{H})$ with probability at least $1 - \frac{2d \log n}{n^{3d+1}}$. That is $\mathbb{P}(\mathcal{E}_2 \mid \mathcal{E}_1) \geq 1 - \frac{2d \log n}{n^{5d}}$.

Now, we analyze the query complexity of the algorithm on the conditional space that the events $\mathcal{E}_1$ and $\mathcal{E}_2$ have occurred. By the description of the algorithm, we make $\mathcal{O}_d\left(\frac{\log^{d+3} n}{\varepsilon^2}\right)$ GPIS_1 queries per tuple in Step 2, and $\mathcal{O}_d(\log^{d+1} n)$ GPIS_2 queries per tuple in Step 4. Using Observation 4.20, there can be $\mathcal{O}_d\left(\frac{\log^{4d} n}{\varepsilon^2}\right)$ tuples present in any iteration. Recall that the number of iterations is $(i^* + 2)$, that is, $\mathcal{O}_d(\log n)$. Since $i^* \leq 2d \log n$, the query complexity of our algorithm is

$$\mathcal{O}_d\left(\log n \cdot \frac{\log^{4d} n}{\varepsilon^2} \cdot \left(\frac{\log^{d+3} n}{\varepsilon^2} + \log^{d+1} n\right)\right) = \mathcal{O}_d\left(\frac{\log^{5d+4} n}{\varepsilon^4}\right),$$

where each query is either a GPIS_1 or a GPIS_2 query.

Now we compute the probability of success of our algorithm. Observe that

$$\mathbb{P}(\text{SUCCESS}) \geq \mathbb{P}(\mathcal{E}_1 \cap \mathcal{E}_2) = \mathbb{P}(\mathcal{E}_1) \cdot \mathbb{P}(\mathcal{E}_2 \mid \mathcal{E}_1) \geq \left(1 - \tfrac{4d \log n}{n^{5d}}\right) \cdot \left(1 - \tfrac{2d \log n}{n^{5d}}\right) \geq 1 - \tfrac{1}{n^{4d}}$$

□

4.9 Conclusion

This chapter generalizes the results of [BHR+18,BHR+20] and the results of [BBGM21] presented in Chapter 3. The bottleneck to this generalization was discussed at the end of

Chapter 3. We believe we could overcome this bottleneck because of our sparsification technique which we believe might find other uses. We gave some justification in the chapter as to why this sparsification technique could overcome the bottleneck. Apart from the sparsification technique, our other non-trivial contribution was in coarse estimation. We would like to again note that the query complexity of Dell et al. [DLM20a], though polylogarithmic, is better than us in the exponent of the logarithmic term. Improving the query complexity of Dell et al. [DLM20a] remains an interesting future direction.

Chapter 5

Hitting Set Estimation using GPIS Queries

Contents

5.1 Introduction

There is a vast literature available on the query complexity of graph problems with classical polynomial time algorithms (refer to book [Gol17]). There have been works that look at algorithmically hard problems through the lens of query complexity [IMR$^+$18, IY18, ORRR12]. In this work, we use ideas of *parameterized complexity* in order to study the query complexity of an NP-hard problem. The HITTING SET (and VERTEX COVER) problem is a test problem for all new techniques of parameterized complexity and also in every subarea that parameterized complexity has explored. We continue the tradition and study the query complexity of such a problem. Our query model is Generalized Partite Independent Set Query (GPIS) oracle discussed in Chapter 4.

Independent set based oracles mostly report on the intersection of the edge set with set(s) of vertices – the oracles give a YES/NO answer to the existence of an intersection, in a few cases they even count the number of such intersections. Lately, there has been a wide range of interest in them. By now, they have been used for solving a lot of problems – edge and hyperedge estimation in graphs and hypergraphs [BHR$^+$20, CLW20, DLM20b, BBGM19a], sampling edges and hyperedges [CLW20, DLM20b], fine-grained complexity of approximate counting problems [DL21], computing minimum cut [RSW18] and submodular function minimization [GPRW20] using CUT queries; in the CUT query for a graph G, the oracle took as input a vertex subset $S \subseteq V(G)$ and outputs the number of cut edges between S and $V(G) \setminus S$.

Pushing the frontiers of the use of independent set based oracles into the context of NP-Hard problems, it is reasonable to study query complexity of their parameterized versions. Before we move forward, let us introduce a few basic definitions in parameterized complexity. The goal of parameterized complexity is to find ways of solving NP-hard problems more efficiently than brute force: the aim is to restrict the combinatorial explosion to a parameter that is hopefully much smaller than the input size. Formally, a *parameterization* of a problem is assigning an integer k to each input instance. A parameterized problem is said to be *fixed-parameter tractable (FPT)* if there is an algorithm that solves the problem in time $f(k) \cdot |I|^{O(1)}$, where $|I|$ is the size of the input and f is

an arbitrary computable function depending only on the parameter k. There is a long list of NP-hard graph problems that are FPT under various parameterizations: finding a vertex cover of size k, finding a cycle of length k, finding a maximum independent set in a graph of treewidth at most k, etc. For more details, the reader is referred to the monograph [CFK$^+$15].

As HITTING SET has been a kind of a test problem for any new area that parameterized complexity has explored, our focus in this thesis is on

- GPIS oracle to study parameterized decision (optimization) version of HITTING SET;

In this context, it may be mentioned that Iwama et al. [IY18] initiated the study of parameterized version of some NP-Hard problems in the graph property testing framework with access to standard oracles, like degree query and neighbor query along with some *added power* to the oracle. We will give the details of their work in Section 5.2. We believe that apart from the oracles used in [IY18], these independent set based query models will be useful to study the (parameterized) query complexity of other NP-Hard problems.

5.1.1 Problem definition and our results

Definition 5.1 (d-hitting set of a d-uniform hypergraph)**.** Let us consider a d-uniform hypergraph $\mathcal{H}(U(\mathcal{H}), \mathcal{F}(\mathcal{H}))$. The *$d$-hitting set* of $\mathcal{H}$, denoted by $HS(\mathcal{H})$, is defined as a subset of $U(\mathcal{H})$, of minimum cardinality, that intersects with every hyperedge in $\mathcal{F}(\mathcal{H})$.

The d-HITTING-SET problem (we consider in this chapter) is defined as follows.

d-HITTING-SET
Input: The set of n vertices $U(\mathcal{H})$ of a d-uniform hypergraph $\mathcal{H}$, access to a GPIS oracle, and a positive integer k.
Output: A d-hitting set $HS(\mathcal{H})$ such that $\lvert HS(\mathcal{H})\rvert \leq k$ if such a set exists. Otherwise, we report such a set does not exist.

Throughout this chapter, n denotes the number of vertices in the graph or hypergraph that can be understood from the context. Note that d-HITTING-SET is the generalization of *vertex cover* in graphs. d-DECISION-HITTING-SET is the usual decision version of d-HITTING-SET, respectively. The main results of our work are as follows; they include both upper and matching lower bounds for the HITTING-SET problem.

Theorem 5.2 (Upper bounds)**. (i)** d-HITTING-SET *can be solved with* $\widetilde{\mathcal{O}}_d(k^d \log n)$ GPIS *queries.*

(ii) d-DECISION-HITTING-SET *can be solved with* $\widetilde{\mathcal{O}}_d\left(\min\left\{k^d \log n, k^{2d^2}\right\}\right)$ GPIS *queries.*

Theorem 5.3 (Lower bound)**.** *Any algorithm that solves* d-DECISION-HITTING-SET *requires* $\Omega\left(\binom{k+d}{d}\right)$ GPIS *queries.*

Organization of the chapter

We start this chapter by reviewing relevant literature in Section 5.2. We discuss preliminaries in Section 5.3.. In Section 5.4, we deal with the promised version (in the promised version of d-HITTING-SET, there is a promise that the size of the minimum hitting set is at most k) of the problem leading to the hitting set problem. The decision version of the hitting set problem is discussed in Section 5.5. Section 5.6 has details on the lower bound proof. Section 5.7 concludes this chapter with a discussion.

5.2 Related works

To the best of our knowledge, the only work prior to ours related to parameterization in the query complexity model was by Iwama and Yoshida [IY18]. They studied property testing for several parameterized NP optimization problems in the query complexity model. For the query, they could ask for the degree of a vertex, neighbors of a vertex – both *local queries* and had an added power of sampling an edge uniformly at random. As

the probability space is over the entire edge set, asking for a random edge does not qualify as a local query. To justify the added power of the oracle to sample edges uniformly at random, they have shown that $\Omega(\sqrt{n})$ degree and neighbor queries are required to solve VERTEX-COVER. Apart from that, an important assumption in their work is that the algorithms knew the number of edges, which is not what is usually done in query complexity models. Also, the algorithms that are designed gives correct answer only for stable instances [1]. Under these assumptions, they study the parameterized query complexity of vertex cover, feedback vertex set, multicut, dominating set and non-existence of paths of specific length and give constant query testable algorithms if the parameter k is treated to be a constant.

Note that our query oracles can access some global information. However, our oracles do not use any randomness, does not know the number of edges, consider all instances, and have a simple unifying structure in terms of asking for the existence of an edge between disjoint sets of vertices. We feel that our work marked by its use of independent set based oracle queries is not comparable to the work by Iwama and Yoshida [IY18]. We mention in passing that their vertex cover algorithm admits a query complexity of $\widetilde{\mathcal{O}}(\frac{2^k}{\varepsilon^2})$ and either finds a vertex cover of size at most k or decides that there is no vertex cover of size bounded by k even if we delete εm edges, where the number of edges m is known in advance. In contrast to the work of Iwama and Yoshida, our algorithm uses BIS query for the vertex cover problem; it neither knows the number of edges, nor estimates it. Our algorithm admits a query complexity of $\widetilde{\mathcal{O}}(k^4)$ and we either find a vertex cover of size at most k if it exists or decide that there is no vertex cover of size bounded by k. We also provide lower bound arguments.

[1] If the input is stable, then there are *large* number of interesting objects of interest. So, random sampling of edges will work well if the input is stable. Stable instance assumption, which is standard in *property testing*, is necessary in the work by Iwama and Yoshida [IY18] as they consider only local queries.

5.3 Preliminaries

Recall once again that a *hypergraph* is a set system $(U(\mathcal{H}), \mathcal{F}(\mathcal{H}))$, where $U(\mathcal{H})$ is the set of vertices and $\mathcal{F}(\mathcal{H})$ is the set of hyperedges. Given hypergraphs $\mathcal{H}_1, \mathcal{H}_2$ defined on the same set of n vertices, the hypergraph $\mathcal{H}_1 \cup \mathcal{H}_2$ is such that $U(\mathcal{H}_1 \cup \mathcal{H}_2) = U(\mathcal{H}_1) = U(\mathcal{H}_2)$ and $\mathcal{F}(\mathcal{H}_1 \cup \mathcal{H}_2) = \mathcal{F}(\mathcal{H}_1) \cup \mathcal{F}(\mathcal{H}_2)$. A hypergraph $\mathcal{H}'$ is a *sub-hypergraph* of $\mathcal{H}$ if $U(\mathcal{H}') \subseteq U(\mathcal{H})$ and $\mathcal{F}(\mathcal{H}') \subseteq \mathcal{F}(\mathcal{H})$. For a hyperedge $F \in \mathcal{F}(\mathcal{H})$, $U(F)$ or simply F denotes the subset of vertices that form the hyperedge. All hyperedges of a *d-uniform hypergraph* have exactly d vertices. $HS(\mathcal{H})$ denotes a minimum d-HITTING SET of the d-uniform hypergraph $\mathcal{H}$.

For us "choose a random hash function $h : V \to [N]$", means that each vertex in V is colored with one of the N colors uniformly and independently at random. In this chapter, for a problem instance (I, k) of a parameterized problem Π, a *high probability* event means that it occurs with probability at least $1 - \frac{1}{k^c}$, where k is the given parameter and c is a positive constant. The following known observation is important for the analysis of algorithms described in this work.

Observation 5.4. (i) Let Π be a parameterized maximization (minimization) problem and let (I, k) be an instance of Π. Let $\mathcal{A}$ be a randomized algorithm for Π, with success probability at least p, where $0 < p < 1$ is a constant. Then, if we repeat $\mathcal{A}$ for $C \log k$ times for a suitably large constant C and report the maximum (minimum) sized output over $C \log k$ outcomes, then the event that $\mathcal{A}$ succeeds occurs with high probability. If the query complexity of algorithm $\mathcal{A}$ is q, then the query complexity of the $C \log k$ repetitions of $\mathcal{A}$ is $\widetilde{\mathcal{O}}(q)$.

(ii) Let Π be a parameterized decision problem and let (I, k) be an instance of Π. Let $\mathcal{A}$ be a randomized algorithm for Π, with success probability at least p, where $\frac{1}{2} < p < 1$ is a constant. Then, if we repeat $\mathcal{A}$ for $C \log k$ times for a suitably large constant C and report the *majority* of the $C \log k$ outcomes, then the event that $\mathcal{A}$ succeeds occurs with high probability. If the query complexity of algorithm $\mathcal{A}$ is q, then the query complexity of the $C \log k$ repetitions of $\mathcal{A}$ is $\widetilde{\mathcal{O}}(q)$.

Representative Sets[2]: In the following, we formally define representative set for hypergraphs.

For the hypergraph $\mathcal{H}$, $\mathcal{F}' \subseteq \mathcal{F}(\mathcal{H})$ is said to be a *k-representative set* corresponding to $\mathcal{H}$ if the following is satisfied for any $X \subset U(\mathcal{H})$ of size k. If there is an $F \in \mathcal{F}(\mathcal{H})$ satisfying $X \cap F = \emptyset$, then there exists $F' \in \mathcal{F}'$ such that $X \cap F' = \emptyset$.

The following proposition gives a bound on the size of a k-representative set corresponding to a d-uniform hypergraph.

Proposition 5.5 ([BT81]). *If $\mathcal{H}$ is a d-uniform hypergraph, then there exists a $\binom{k+d}{d}$ sized k-representative set corresponding to $\mathcal{H}$.*

Corollary 5.6 ([CFK+15]). *For a set system $\mathcal{H}$ as above, consider the family $\mathcal{Z} = \{U(F) \mid F \in \mathcal{F}(\mathcal{H})\}$ and let $\widehat{\mathcal{Z}}$ be a k-representative set of $\mathcal{Z}$ as obtained in Proposition 5.5. Let $\mathcal{H}'$ be the set system where $U(\mathcal{H}') = \bigcup_{Z \in \widehat{\mathcal{Z}}} S$ and $\mathcal{F}(\mathcal{H}') = \{F \in \mathcal{F} \mid U(F) \in \widehat{\mathcal{S}}\}$. $(\mathcal{H}, k)$ is a YES instance of d-DECISION-HITTING-SET if and only if $(\mathcal{H}', k)$ is a YES instance of d-DECISION-HITTING-SET.*

We will use this fact of representative set crucially to solve d-DECISION-HITTING-SET using GPIS oracle.

5.3.1 Technical preliminary

For ease of exposition, we now define a query oracle GPISE which is equivalent to GPIS upto $\mathcal{O}(\log n)$ factor. GPISE returns a witness hyperedge for a YES answer of GPIS and returns NULL, otherwise. The formal definition follows.

Generalized d-partite independent set edge oracle **(**GPISE**):** For a d-uniform hypergraph $\mathcal{H}$, given d pairwise non-empty disjoint subsets $A_1, A_2, \ldots, A_d \subseteq U(\mathcal{H})$ as input, a GPISE query oracle outputs a hyperedge $(u_1, \ldots, u_d) \in \mathcal{F}(\mathcal{H})$ such that $u_i \in A_i$, for each $i \in [d]$; otherwise, the GPISE oracle reports NULL.

[2]Informally speaking, representative set in parameterized complexity is analogous to *coreset* in computational geometry.

Observation 5.7. Let $A_1, \ldots, A_d$ be d pairwise disjoint subsets of $U(\mathcal{H})$. A GPISE query with input $A_1, \ldots, A_d$ can be simulated by using $\mathcal{O}_d(\log n)$ GPIS queries.

Proof. We describe the simulation process in a recursive fashion. We first make a GPIS query with input $A_1, \ldots, A_d$. If GPIS reports there is no hyperedge spanning the sets $A_1, \ldots, A_d$, then we report NULL as the answer to the GPISE query. Otherwise, for each $i \in [d]$, we partition each A_i into two parts, that is, A_{i1} and A_{i2} such that $|A_{i1}| = \left\lceil \frac{|A_i|}{2} \right\rceil$ and $|A_{i2}| = \left\lfloor \frac{|A_i|}{2} \right\rfloor$. For each $A_{1j}, \ldots, A_{dj}$ with $j \in \{1, 2\}$, we make a GPIS query with input $A_{1j}, \ldots, A_{dj}$. Note that we make 2^d GPIS queries. Observe that there exists at least one combination of $A_{1j}, \ldots, A_{dj}$ such that GPIS reports that $m(A_{1j}, \ldots, A_{dj}) \neq 0$ [3]. Now we call for GPISE query with one such $A_{1j}, \ldots, A_{dj}$ (such that $m(A_{1j}, \ldots, A_{dj}) \neq 0$) as input, and reports the answer of the GPISE query with input $A_{1j}, \ldots, A_{dj}$ as the answer to the GPISE query with input $A_1, \ldots, A_d$. The correctness of the answer to the GPISE query follows from the description of the simulation process. Let $Q_E(A_1, \ldots, A_d)$ denotes the number of GPIS query, that our simulation process makes, to answer GPISE query with input $A_1, \ldots, A_d$. Hence,

$$Q_E(A_1, \ldots, A_d) \leq 1 + 2^d + \max_{A_{1j}, \ldots, A_{dj}} Q_E\left(A_{1j}, \ldots, A_{dj}\right)$$

Observe that $Q_E(A_1, \ldots, A_d) = \mathcal{O}_d(\log n)$. □

Observation 5.8. Let $\mathcal{G}$ be **a** subgraph, of a d uniform hypergraph $\mathcal{H}$, induced by $V \subseteq U(\mathcal{H})$. There exists an algorithm $\mathcal{A}$ that makes $\mathcal{O}_d\left(\log \frac{1}{\delta}\right)$ GPISE queries and performs as follows: if there exists at least one hyperedge in $\mathcal{G}$, then $\mathcal{A}$ returns a (arbitrary) hyperedge in $\mathcal{G}$ with probability $1 - \delta$; otherwise, $\mathcal{A}$ reports that there is no hyperedge in $\mathcal{G}$.

Remark 5.1. By Observation 5.7, the above algorithm $\mathcal{A}$ implies an algorithm that uses $\mathcal{O}_d\left(\log n \log \frac{1}{\delta}\right)$ GPIS queries and gives an output that is same as that of $\mathcal{A}$.

[3] $m(A_{1j}, \ldots, A_{dj})$ is the number of hyperedges having a vertex in A_{ij}'s for each $i \in [d]$

Proof of Observation 5.8. Let us consider partitioning the vertex set V into d parts $B_1, \ldots, B_d$ such that each vertex in V is present in one of the B_is uniformly at random. Then let us make a GPISE query with input $B_1, \ldots, B_d$. If there exists a hyperedge in $\mathcal{G}$, then the probability that all of the d vertices of a particular hyperedge are in different B_is is $\frac{(d-1)!}{d^d}$ and the GPISE query reports such an edge with probability at least $\frac{(d-1)!}{d^d}$. If there is no hyperedge in $\mathcal{G}$, then the GPISE query reports NULL.

The algorithm $\mathcal{A}$ repeats the above procedure $R = \mathcal{O}\left(\frac{d^d}{(d-1)!} \log \frac{1}{\delta}\right)$ times, that is, partitions the vertices in V (into d parts) R times. $\mathcal{A}$ reports a hyperedge if at least one of the R GPISE queries reports a hyperedge. Otherwise, $\mathcal{A}'$ reports that there is no hyperedge in $\mathcal{G}$. The correctness of the algorithm $\mathcal{A}$ follows from its description. Moreover, note that $\mathcal{A}$ makes $R = \mathcal{O}_d\left(\log \frac{1}{\delta}\right)$ GPISE queries. □

5.4 Algorithm for d-HITTING-SET

In this Section, we will prove the following result. The oracle access will be to GPISE (instead of GPIS) and because of the already proved equivalence of them in Section 5.3.1, the final results will follow.

Theorem 5.9 (Restatement of Theorem 5.2 (i) in terms of GPISE queries)**.** d-HITTING-SET *can be solved with* $\widetilde{\mathcal{O}}_d\left(k^d\right)$ GPISE *queries.*

Observe that Theorem 5.2 (i) directly follows from the above theorem and Observation 5.7.

The algorithm for d-HITTING-SET, having a query complexity of $\widetilde{\mathcal{O}}(k^{2d})$ GPISE queries, will use an algorithm admitting a query complexity of $\widetilde{\mathcal{O}}(k^d)$ for a version of this problem where the input instance is promised to have a hitting set of size at most k. The main idea to solve the version is to sample a *suitable* sub-hypergraph having **a** bounded number of hyperedges, using GPISE queries on the input hypergraph, such that the hitting set of the sampled hypergraph is a hitting set of the original hypergraph and vice versa. Two main ingredients in the proof of Theorem 5.9 are the following:

1. Structure of a *sunflower* in a hypergraph [ER60].

2. An algorithm for GAP-d-HITTING-SET problem using $\widetilde{\mathcal{O}}(k)$ GPISE queries.

The d-HITTING SET problem can be solved by using the algorithm for the promised version of the d-HITTING SET problem along with the algorithm for GAP-d-HITTING-SET problem.

5.4.1 GAP-d-HITTING-SET problem

In GAP-d-HITTING-SET on a d-uniform hypergraph $\mathcal{H}$, the objective is to report ACCEPT if $\mathcal{H}$ has a hitting set of size at most k, to report REJECT if the size of any minimum hitting set of $\mathcal{H}$ is more than dk, and to report ACCEPT or REJECT arbitrarily if the hitting set lies between k and dk. We will show (in Observation 5.10) that GAP-d-HITTING-SET can be solved by using $\widetilde{\mathcal{O}}(k)$ GPISE queries. For the d-HITTING SET problem, we first solve GAP-d-HITTING-SET. If the algorithm for GAP-d-HITTING-SET reports REJECT, then we conclude that the size of the minimum hitting set of $\mathcal{H}$ is at least k. If algorithm for GAP-d-HITTING-SET reports ACCEPT, then $\mathcal{H}$ has a hitting set of size at most dk. Now we can use our algorithm for the promised version of d-HITTING SET to give the final answer to the non-promised d-HITTING SET.

Observation 5.10. GAP-d-HITTING-SET can be solved by using $\widetilde{\mathcal{O}}(k)$ GPISE queries.

Proof. We find a *packing* [4] of size at most $k+1$ in a greedy fashion, by using $\widetilde{\mathcal{O}}(k)$ GPISE queries as follows.

(i) Set $V = U(\mathcal{H})$, $\mathcal{G} = \mathcal{H}$.

(ii) Run algorithm $\mathcal{A}$ (the algorithm corresponding to Observation 5.8) on $\mathcal{G}$ with parameter $\delta = \frac{1}{k^c}$, where c is a suitably large constant more than 1.

(iii) If $\mathcal{A}$ reports that there is no edge in $\mathcal{G}$, then report ACCEPT and QUIT.

[4] Packing refers to a set of hyperedges that are vertex disjoint.

(iv) Let F be the hyperedge in $\mathcal{G}$ that is returned by $\mathcal{A}$. If we have seen $k+1$ hyperedges (including F), then we report REJECT and QUIT.

(v) Otherwise, we delete all the vertices in F from $\mathcal{G}$, that is, we set $\mathcal{G} = \mathcal{G} \setminus F$. Go to Step (ii).

The above algorithm calls algorithm $\mathcal{A}$ with parameter $\delta = \frac{1}{k^c}$ at most $k+1$ times. From Observation 5.8, each call of algorithm $\mathcal{A}$ requires $\mathcal{O}_d(\log k)$ GPISE queries and succeeds with probability at least $1 - \delta = 1 - \frac{1}{k^c}$. So, the above algorithm for GAP-d-HITTING-SET makes $\widetilde{\mathcal{O}}(k)$ GPISE queries and succeeds with probability at least $1 - \frac{1}{k^{c-1}}$. Now we discuss the correctness proof of our algorithm for GAP-d-HITTING-SET assuming all calls to algorithm $\mathcal{A}$ suceed. Observe that our algorithm (for GAP-d-HITTING-SET) finds a packing of size at most $k+1$. Moreover, if the algorithm stops after finding a packing of size at most k, then those set of at most k hyperedges correspond to a maximal packing. If hypergraph $\mathcal{H}$ has a hitting set of size at most k, then the size of any (maximal) packing is at most k. In this case, our algorithm quits after finding at most k hyperedges that correspond to a maximal packing, and we report ACCEPT. Now, if the size of the minimum hitting set of $\mathcal{H}$ is more than dk, then the size of any maximal packing is at least $k+1$. In this case, our algorithm will be able to find a packing of size at least $k+1$, and we report REJECT. □

5.4.2 Algorithm for d-HITTING-SET via d-PROMISED-HITTING-SET

In this Section, we begin by studying the following promised problem.

d-PROMISED-HITTING-SET **Input:** The set of vertices $U(\mathcal{H})$ of a d-uniform hypergraph $\mathcal{H}$ such that $\lvert HS(\mathcal{H})\rvert \leq k$ and the access to a GPISE oracle. **Output:** A hitting set of $\mathcal{H}$ that is of size at most k.

We will show at the end of this section that the algorithm for d-HITTING-SET follows from the algorithms for GAP-d-HITTING-SET and d-PROMISED-HITTING-SET problems.

For d-PROMISED-HITTING-SET, we design an algorithm with query complexity $\widetilde{\mathcal{O}}(k^d)$. See Algorithm 5.1 for the pseudocode of d-PROMISED-HITTING-SET.

Algorithm 5.1: Algorithm for d-PROMISED-HITTING-SET

Input: The set of vertices $U(\mathcal{H})$ of a d-uniform hypergraph $\mathcal{H}$ such that $|HS(\mathcal{H})| \leq k$ and the access to a GPISE oracle.
Output: A hitting set of $\mathcal{H}$ that is of size at most k.

begin
  Take $\alpha \log k$ random hash functions of the form $h : U(\mathcal{H}) \to [\beta k]$, where $\alpha = 100d^2$ and $\beta = 100d^3 2^{d+5}$.
  for *(each hash function h)* **do**
    Find $U_i = \{u \in U(\mathcal{H}) : h(u) = i\}$, where $i \in [\beta k]$.
    Make a GPISE query with input $(U_{i_1}, \ldots, U_{i_d})$ for each $1 \leq i_1 < \ldots < i_d \leq \beta k$ such that $U_{i_j} \neq \emptyset \; \forall j \in [d]$.
    Let $\mathcal{F}'$ be the set of hyperedges that are output by the $\mathcal{O}(k^d)$ GPISE queries.
    Generate a subhypergraph $\mathcal{H}^h$ of $\mathcal{H}$ such that $U(\mathcal{H}^h) = U(\mathcal{H})$ and $\mathcal{F}(\mathcal{H}^h) = \mathcal{F}'$.
  Let $\mathcal{H}_1, \ldots, \mathcal{H}_{\alpha \log k}$ be the subhypergraphs generated by $\alpha \log k$ hash functions.
  Find $\widehat{\mathcal{H}} = \mathcal{H}_1 \cup \cdots \cup \mathcal{H}_{\alpha \log k}$.
  Report $HS(\widehat{\mathcal{H}})$ as the output.

Theorem 5.11. *d-PROMISED-HITTING-SET can be solved with $\widetilde{\mathcal{O}}(k^d)$ GPISE queries.*

Here, we give an outline of the algorithm. The first step of the algorithm involves, for a positive integer b, a sampling primitive $\mathcal{L}_b$ for the problem. This sampling primitive was used in a streaming setting in [CCE+16]. We extend it to our setting of query complexity. Let $\mathcal{H}$ be the d-uniform hypergraph whose vertex set $U(\mathcal{H})$ is known and hyperedge set $\mathcal{F}(\mathcal{H})$ is unknown to us. Let $h : U(\mathcal{H}) \to [b]$ be a random hash function. Let $U_i = \{u \in U(\mathcal{H}) : h(u) = i\}$, where $i \in [b]$. Note that $U_1, \ldots, U_b$ form a partition of $U(\mathcal{H})$, some of the U_is can be empty. We make a GPISE query with input $(U_{i_1}, \ldots, U_{i_d})$ for each $1 \leq i_1 < \ldots < i_d \leq b$ such that $U_{i_j} \neq \emptyset \; \forall j \in [d]$. Observe that we make $\mathcal{O}(b^d)$ queries to the oracle. Let $\mathcal{F}'$ be the set of hyperedges that are output by the $\mathcal{O}(b^d)$ GPISE

queries. Now, we can generate a subhypergraph $\mathcal{H}^h$ of $\mathcal{H}$ such that $U(\mathcal{H}^h) = U(\mathcal{H})$ and $\mathcal{F}(\mathcal{H}^h) = \mathcal{F}'$.

In the rest of this section, we abuse the standard graph theoretic terminology by sometimes calling a d-uniform hypergraph as a graph and a hyperedge as an edge, respectively.

We find $\alpha \log k$ samples by calling the sampling primitive $\mathcal{S}_{\beta k}$ for $\alpha \log k$ times, where $\alpha = 100d^2$ and $\beta = 100d^3 2^{d+5}$. Let the subgraphs resulting from the sampling be $\mathcal{H}_1, \ldots, \mathcal{H}_{\alpha \log k}$. Let $\widehat{\mathcal{H}} = \mathcal{H}_1 \cup \cdots \cup \mathcal{H}_{\alpha \log k}$. Note that we can construct $\widehat{\mathcal{H}}$ by making $\widetilde{\mathcal{O}}(k^d)$ GPISE queries. For completeness, the detailed proof of Theorem 5.11 is given at the end of Section 5.4.2. Observe that if we prove the following lemma, then we are done with the proof of Theorem 5.11. Recall that $HS(\mathcal{H})$ denotes a minimum hitting set of $\mathcal{H}$.

Lemma 5.12 (Proof in Section 5.4.3)**.** *If $|HS(\mathcal{H})| \leq k$, then $HS(\mathcal{H}) = HS(\widehat{\mathcal{H}})$ with high probability.*

Remark 5.2. The statement of our Lemma 5.12 is same as that of Theorem 3.2 of [CCE$^+$16], but the proof is not. We feel the proof of Theorem 3.2 in [CCE$^+$16] is incomplete because of the following reason. The authors argue that $HS(G) = HS(U \cup F)$ where G denotes the hypergraph, U is the set of large cores and F is the set of hyperedges that do not include any significant core. Next, the authors argue that $HS(U \cup F) = HS(U' \cup F)$ where U' is the set of large cores that do not contain significant cores. We feel that their statement is correct but the part of the proof meant for this, is sketchy. This is mainly because $HS(U' \cup F)$ may not hit some hyperedges in G that contain a large core C such that C contains a core significant but not large core C'. As Lemma 5.12 is crucial for us, we leave nothing for chance and give an alternate, detailed proof in Section 5.12.

Now, we will show that Theorem 5.11 follows from Lemma 5.12.

Proof of Theorem 5.11. Our query procedure will be as follows. We find $\alpha \log k$ samples using the primitive $\mathcal{S}^d_{\beta k}$, where $\alpha = 100d^2$ and $\beta = 100d^3 2^{d+5}$. Let those subgraphs be $\mathcal{H}_1, \ldots, \mathcal{H}_{\alpha \log k}$. Let $\widehat{\mathcal{H}} = \mathcal{H}_1 \cup \cdots \cup \mathcal{H}_{\alpha \log k}$. We find a minimum hitting set of $\widehat{\mathcal{H}}$. We

report $HS(\widehat{\mathcal{H}})$ as $HS(\mathcal{H})$. The correctness of the algorithm follows from Lemma 5.12. The query complexity of the algorithm is $\widetilde{\mathcal{O}}(k^d)$, which is evident from the sampling primitive described at the beginning of this Section. □

We finally come to the proof of Theorem 5.2 (restated as Theorem 5.9). Recall that for a hypergraph $\mathcal{H}$, $HS(\mathcal{H})$ denotes a minimum hitting set of $\mathcal{H}$.

Proof of Theorem 5.9. We first run the algorithm of GAP-d-HITTING-SET that succeeds with high probability (see Observation 5.10). Under the assumption that the algorithm of GAP-d-HITTING-SET succeeds, it reports ACCEPT if $\mathcal{H}$ has a hitting set of size at most k, reports REJECT if the size of any minimum hitting set of $\mathcal{H}$ is more than dk, and it reports ACCEPT or REJECT arbitrarily if the hitting set is more than k and at most dk.

If the algorithm of GAP-d-HITTING-SET reports REJECT, we conclude that $|HS(\mathcal{H})| \geq k+1$. So, in this case we report that there does not exist any hitting set of size at most k. Otherwise, if the algorithm of GAP-d-HITTING-SET reports ACCEPT, then $|HS(\mathcal{H})| \leq dk$. As $|HS(\mathcal{H})| \leq dk$, $HS(\mathcal{H})$ can be found using our algorithm for d-PROMISED-HITTING-SET by making $\widetilde{\mathcal{O}}_d(k^d)$ GPISE queries. If $|HS(\mathcal{H})| \leq k$, we output $HS(\mathcal{H})$ and if $|HS(\mathcal{H})| > k$, we report that there does not exist a hitting set of size at most k. The total number of GPISE queries made by our algorithm for d-HITTING-SET is $\widetilde{\mathcal{O}}_d(k^d)$. □

Only thing that is left to show is the proof of Lemma 5.12.

5.4.3 Proof of Lemma 5.12

To prove Lemma 5.12, we need some intermediate definitions and results. As mentioned earlier, we use the structure of the sunflower in a hypergraph [ER60].

Some definitions

The core of a sunflower is the pairwise intersection of the hyperedges present in the sunflower, which is formally defined as follows.

Definition 5.13. Let $\mathcal{H}$ be a d-uniform hypergraph; $\mathcal{S} = \{F_1, \ldots, F_r\} \subseteq \mathcal{F}(\mathcal{H})$ is a *r-sunflower* in $\mathcal{H}$ if there exists $C \subseteq U(\mathcal{H})$ such that $F_i \cap F_j = C$ for all $1 \leq i < j \leq r$. C is defined to be the *core* of the sunflower in $\mathcal{H}$ and $\mathcal{P} = \{F_i \setminus C : i \in [r]\}$ is defined as the set of *petals* of the sunflower $\mathcal{S}$ in $\mathcal{H}$.

Based on the number of hyperedges forming the sunflower, the core of a sunflower can be *large*, *significant*, or *small*. We fix them in such a way that each large core is significant and each significant core (and hence, large core also) must intersect with any hitting set. The formal definition follows.

Definition 5.14. Let $S_{\mathcal{H}}(C)$ denote the maximum integer r such that C is the core of a r-sunflower in $\mathcal{H}$. If $S_{\mathcal{H}}(C) > 10dk$, C is *large*. If $S_{\mathcal{H}}(C) > k$, C is *significant*.

The promise that the hitting set is bounded by k, will help us

(i) to bound the number of hyperedges that do not contain any large core as a subset,

(ii) to guarantee that all the large cores in the original hypergraph, that do not contain any significant cores as a subset, are significant in the sampled hypergraph with high probability. This will ensure that the large cores in the original hypergraph will intersect any hitting set of the sampled hypergraph, and

(iii) to guarantee that all the hyperedges that do not contain any large core as a subset, are present in the sampled hypergraph with high probability.

Using the above observations, we can prove that the hitting set of the sampled hypergraph is the hitting set of the original graph with high probability. To formalize the above discussion, we state the following proposition and then define some sets, which will be needed for our analysis.

Proposition 5.15 ([ER60])**.** *Let $\mathcal{H}$ be a d-uniform hypergraph. If $|\mathcal{F}(\mathcal{H})| > d!k^d$, then there exists a $(k+1)$-sunflower in $\mathcal{H}$.*

Definition 5.16. In the hypergraph $\mathcal{H}$, $\mathcal{C}$ is the set of *large* cores; $\mathcal{F}_s$ is the family of edges that do not contain any *large* core; $\mathcal{C}'$ is the family of *large* cores none of which contain a *significant* core as a subset.

Bounding $|\mathcal{F}_S|$ and $\mathcal{C}'$ when $HS(\mathcal{H}) \leq k$

The following two results (Lemma 5.17 and 5.18) give useful bounds on $|\mathcal{F}_S|$ and $\mathcal{C}'$ with respect to the input instances of d-PROMISED-HITTING-SET.

Lemma 5.17. *If $|HS(\mathcal{H})| \leq k$, then $|\mathcal{F}_s| \leq d!(10dk)^d$. That is, if the hitting set of the hypergraph $\mathcal{H}$ is bounded by k, then the number of hyperedges that do not contain any large core is at most $d!(10dk)^d$.*

Proof. If $|\mathcal{F}_s| > d!(10dk)^d$, then there exists a $(10dk+1)$-sunflower $\mathcal{S}$ in $\mathcal{H}$ by Proposition 5.15 such that each edge in $\mathcal{S}$ belongs to $\mathcal{F}_s$. First, since $|HS(\mathcal{H})| \leq k$, the core $C_{\mathcal{H}}(\mathcal{S})$ of $\mathcal{S}$ must be non-empty. Note that $C_{\mathcal{H}}(\mathcal{S})$ is a large core and $C_{\mathcal{H}}(\mathcal{S})$ is contained in every edge in $\mathcal{S}$. Observe that we arrived at a contradiction, because any edge in $\mathcal{S}$ is also an edge in $\mathcal{F}_s$ and any edge in $\mathcal{F}_s$ does not contain a large core by definition. Hence, $|\mathcal{F}_s| \leq d!(10dk)^d$. □

Lemma 5.18. *If $|HS(\mathcal{H})| \leq k$, then $|\mathcal{C}'| \leq (d-1)!k^{d-1}$. That is, if the hitting set of the hypergraph $\mathcal{H}$ is bounded by k, then the number of large cores without containing any significant core as a subset is at most $(d-1)!k^{d-1}$.*

Proof. Let us consider the set system of all cores in $\mathcal{C}'$. Note that the number of elements present in each core in $\mathcal{C}'$ is at most $d-1$. If $|\mathcal{C}'| > (d-1)! \cdot k^{d-1}$, then there exists a $(k+1)$-sunflower $\mathcal{S}'$, by Proposition 5.15. Let $C_1, \ldots, C_{k+1}$ be the sets present in the sunflower $\mathcal{S}'$ and let $C_{\mathcal{S}'}$ be the core of $\mathcal{S}'$. Observe that if $C_{\mathcal{S}'} = C_1 \cap \cdots \cap C_{k+1} = \emptyset$, then $|HS(\mathcal{H})| > k$.

To complete the proof of this lemma, we consider the following observation when $C_{\mathcal{S}'}$ is non-empty.

Observation 5.19. If $C_{\mathcal{S}'}$ is non-empty, then $C_{\mathcal{S}'}$ is the pair-wise intersection of a family of $k+1$ edges in $\mathcal{H}$.

Proof. Let A_i be the set of at least $10dk$ edges that form a sunflower with core C_i, where $i \in [k+1]$. Observe that this is possible as each C_i is a large core. Before proceeding

further, note that $C_i \cap C_j = C_{\mathcal{S}'}$ and $(C_i \setminus C_{\mathcal{S}'}) \cap (C_j \setminus C_{\mathcal{S}'}) = \emptyset$ for all $i, j \in [k+1]$ and $i \neq j$.

Consider $B_i \subseteq A_i$ such that for each $F \in B_i$, $F \cap C_j = C_{\mathcal{S}'}$ $\forall j \neq i$ and $|B_i| \geq 9dk$. First, we argue that B_i exists for each $i \in [k+1]$. Recall that for each $j \in [k+1]$, $|C_j| \leq d-1$. Also, for any pair of edges $F_1, F_2 \in A_i$, $(F_1 \setminus C_i) \cap (F_2 \setminus C_i) = \emptyset$. Thus, using the fact that $C_i \cap C_j = C_{\mathcal{S}'}$ for $i \neq j$, a vertex in $C_j \setminus C_{\mathcal{S}'}$ can belong to at most one edge in A_i. This implies that there are at most $(d-1)k < dk$ sets F in A_i such that $F \cap C_j \neq C_{\mathcal{S}'}$ for some $j \neq i \in [k+1]$. We can safely assume that $k+1 \geq d$ and therefore, the number of edges $F \in A_i$ such that $F \cap C_j = C_{\mathcal{S}'}$ $\forall j \neq i \in [k+1]$ is at least $10dk - dk = 9dk$. Next, we argue that there exists $k+1$ edges $F_1, \ldots, F_{k+1}$ such that $F_i \in B_i$ $\forall i \in [k+1]$ and $F_i \cap F_j = C_{\mathcal{S}'}$ for all $i, j \in [k+1]$ and $i \neq j$. We show the existence of the F_i's inductively. For the base case, take any arbitrary edge in B_1 as F_1. Assume that we have chosen $F_1, \ldots, F_p$, where $1 \leq p \leq k$, such that the required conditions hold. We will show that there exists $F_{p+1} \in B_{p+1}$ such that $F_i \cap F_{p+1} = C_{\mathcal{S}'}$ for each $i \in [p]$. By construction of B_i's, no edge in B_{p+1} intersects with $C_i \setminus C_{\mathcal{S}'}, i \leq p$; but every edge in B_{p+1} contains $C_{\mathcal{S}'}$. Also, none of the chosen edges out of $F_1, \ldots, F_p$, intersects $C_{p+1} \setminus C_{\mathcal{S}'}$. So, if we can select an edge $F \in B_{p+1}$ such that $F \setminus C_{p+1}$ is disjoint from $F_i \setminus C_i$, $\forall i \in [p]$, then we are done. Note that for two edges $F', F'' \in B_{p+1}$, $F' \setminus C_{p+1}$ and $F'' \setminus C_{p+1}$ are disjoint. Consider the set $B'_{p+1} \subseteq B_{p+1}$ such that each edge $F \in B'_{p+1}$ intersects with at least one out of $\{F_1 \setminus C_1, \ldots, F_p \setminus C_p\}$. $\left|B'_{p+1}\right| \leq dp \leq dk$, because $(F_i \setminus C_i) \cap (F_j \setminus C_j) = \emptyset$, $\forall i \neq j \in [p]$ and $|F_i| \leq d, i \in [p]$. As $|B_{p+1}| \geq 9dk$, we select any edge in $B_{p+1} \setminus B'_{p+1}$ as F_{p+1}. □

The above observation implies the following. If $C_{\mathcal{S}'}$ is non-empty, then there exists a $(k+1)$-sunflower in $\mathcal{H}$. So, $S_{\mathcal{H}}(C_{\mathcal{S}'}) > k$ or equivalently $C_{\mathcal{S}'}$ is a significant core. Note that each C_i contains $C_{\mathcal{S}'}$, which is a significant core; which contradicts the definition of $\mathcal{C}'$. Hence, $|\mathcal{C}'| \leq (d-1)!k^{d-1}$. □

Remark 5.3. In [CCE+16], the statement of Lemma 3.5 is same as the combination of our Lemma 5.17 and 5.18. The proof of our Lemma 5.17 is same as that of the corresponding part of the proof Lemma 3.5 in [CCE+16]. However, the corresponding part

of our Lemma 5.18 in the proof of Lemma 3.5 of [CCE$^+$16] is sketchy and incomplete. That is why, we give the complete proof of Lemma 5.17 and 5.18 in this thesis. In particular, inside the proof of Lemma 3.5 in [CCE$^+$16], they have an equivalent statement of Observation 5.19, whose proof is sketchy.

The structure of $\widehat{H}$

The following lemma provides insight into the structure of $\widehat{\mathcal{H}}$ and thereby is the most important part of proving Lemma 5.12.

Lemma 5.20. *Let $\widehat{\mathcal{H}} = \mathcal{H}_1 \cup \cdots \cup \mathcal{H}_{\alpha \log k}$. If $|HS(\mathcal{H})| \leq k$, then the followings hold with high probability.*

(a) $\mathcal{F}_s \subseteq \mathcal{F}(\widehat{\mathcal{H}})$, that is, any hyperedge of the hypergraph $\mathcal{H}$ that does not contain any large core is a hyperedge in the sampled hypergraph $\widehat{\mathcal{H}}$;

(b) $S_{\widehat{\mathcal{H}}}(C) > k,\ \forall C \in \mathcal{C}'$, that is, every large core in the hypergraph $\mathcal{H}$ that does not contain any significant core as a subset is a significant core in the sampled hypergraph $\widehat{\mathcal{H}}$.

Proof. First, consider the two claims stated below.

Claim 5.21. $\forall i \in [\alpha \log k]$, $\mathbb{P}(F \in \mathcal{F}(\mathcal{H}_i) \mid F \in \mathcal{F}_s) \geq \frac{1}{2}$.

Claim 5.22. $\forall i \in [\alpha \log k]$, $\mathbb{P}(S_{\mathcal{H}_i}(C) > k \mid C \in \mathcal{C}') \geq \frac{1}{2}$.

Claim 5.21 says that any herperedge in $\mathcal{F}_s$ is also a hyperedge in $\mathcal{H}_i$ with probability at least $1/2$. Claim 5.22 says that any large core in $\mathcal{C}'$ is a significant core in $\mathcal{H}_i$ with probability at least $1/2$. The proofs of Claims 5.21 and 5.22 are involved which we hold back for now and see its implications.

Recall that $\widehat{\mathcal{H}} = \mathcal{H}_1 \cup \cdots \cup \mathcal{H}_{\alpha \log k}$. Using Claims 5.21 and 5.22, we get

$$\mathbb{P}(F \notin \mathcal{F}(\widehat{\mathcal{H}}) \mid F \in \mathcal{F}_s) \leq \left(1 - \frac{1}{2}\right)^{\alpha \log k} \leq \frac{1}{k^{\alpha}}$$

and

$$\mathbb{P}(S_{\widehat{\mathcal{H}}}(C) \leq k \mid C \in \mathcal{C}') \leq \left(1 - \frac{1}{2}\right)^{\alpha \log k} \leq \frac{1}{k^{\alpha}}$$

Using the union bound together with Lemma 5.17, we can deduce the following

$$\begin{aligned} \mathbb{P}(\mathcal{F}_s \nsubseteq \mathcal{F}(\widehat{\mathcal{H}})) &\leq \sum_{F \in \mathcal{F}_s} \mathbb{P}(F \notin \mathcal{F}(\widehat{\mathcal{H}}) \mid F \in \mathcal{F}_s) \\ &\leq \frac{d!(10k)^d}{k^{\alpha}} \leq \frac{1}{k^{98}} \end{aligned}$$

and

$$\begin{aligned} \mathbb{P}(\exists\, C \in \mathcal{C}' \text{ such that } S_{\widehat{\mathcal{H}}}(C) \leq k) &\leq \sum_{C \in \mathcal{C}'} \mathbb{P}(S_{\widehat{\mathcal{H}}}(C) \leq k \mid C \in \mathcal{C}') \\ &\leq \frac{(d-1)!k^{d-1}}{k^{\alpha}} \leq \frac{1}{k^{99}}. \end{aligned}$$

Hence,

$$\mathbb{P}(\mathcal{F}_s \nsubseteq \mathcal{F}(\widehat{\mathcal{H}}) \text{ or } \exists\, C \in \mathcal{C}' \text{ such that } S_{\widehat{\mathcal{H}}}(C) \leq k) \leq \frac{2}{k^{98}}.$$

This implies that with high probability, $\mathcal{F}_s \subseteq \mathcal{F}(\widehat{\mathcal{H}})$ and $S_{\widehat{\mathcal{H}}}(C) > k,\ \forall C \in \mathcal{C}'$ $\square$

Proofs of Claims 5.21 and 5.22

We now come back to the proofs of Claims 5.21 and 5.22.

Proof of Claim 5.21. Without loss of generality, we will prove the statement for the graph $\mathcal{H}_1$. Let $h : U(\mathcal{H}) \to [\beta k]$ be the random hash function used in the sampling of $\mathcal{H}_1$. Observe that by the construction of $\mathcal{H}_1$, $F \in \mathcal{F}(\mathcal{H}_1)$ if the following two conditions hold.

- $h(u) = h(v)$ if and only if $u = v$, where $u, v \in F$.

- For any $F' \neq F$ and $F' \in \mathcal{F}(\mathcal{H})$, F' and F differ in the color of at least one vertex.

Hence,

$$\mathbb{P}(F \notin \mathcal{F}(\mathcal{H}_1) \mid F \in \mathcal{F}_s) \leq \sum_{u,v \in F: u \neq v} \mathbb{P}(h(u) = h(v)) + \mathbb{P}(\mathcal{E}_1),$$

where $\mathcal{E}_1$ is the event defined as follows

$\mathcal{E}_1$: $\exists$ an edge $F' \in \mathcal{F}(\mathcal{H})$ such that $F' \neq F$ and $\{h(z) : z \in F\} = \{h(z) : z \in F'\}$.

Before we bound the probability of the occurrence of $\mathcal{E}_1$, we show the existence of a set $D \subseteq U(\mathcal{H}) \setminus F$ of bounded cardinality such that each edge in $\mathcal{F}(\mathcal{H}) \setminus \{F\}$ intersects with D.

Observation 5.23. Let $F \in \mathcal{F}_s$. There exists a set $D \subseteq U(\mathcal{H}) \setminus F$ such that each edge in $\mathcal{F}(\mathcal{H}) \setminus \{F\}$ intersects with D and $|D| \leq 2^{d+5}d^2k$.

Proof. For each $C \subset F$, consider the hypergraph $\mathcal{H}_C$ such that $U(\mathcal{H}_C) = U(\mathcal{H}) \setminus C$ and $\mathcal{F}(\mathcal{H}_c) = \{F' \setminus C : F' \in \mathcal{F}(\mathcal{H})$ and $F' \cap F = C\}$. First, we prove that the size of $HS(\mathcal{H}_C)$ is at most $dS_{\mathcal{H}}(C)$. For the sake of contradiction, assume that $|HS(\mathcal{H}_C)| > dS_{\mathcal{H}}(C)$. Then we argue that there exists $\mathcal{F}' \subseteq \mathcal{F}(\mathcal{H}_C)$ such that each pair of hyperedges in $\mathcal{F}'$ are vertex disjoint and $|\mathcal{F}'| > S_{\mathcal{H}}(C)$. If $|\mathcal{F}'| \leq S_{\mathcal{H}}(C)$, then the vertex set $\{w : w \in F', F' \in \mathcal{F}'\}$ is a hitting set of $\mathcal{H}_c$ and it has size at most $dS_{\mathcal{H}}(C)$, which is a contradiction. Therefore, there is a $\mathcal{F}' \subseteq \mathcal{F}(\mathcal{H}_C)$ such that each pair of hyperedges in $\mathcal{F}'$ is vertex disjoint and $|\mathcal{F}'| > S_{\mathcal{H}}(C)$. Observe that the set of edges $\{F'' \cup C : F'' \in \mathcal{F}'\}$ forms a t-sunflower in $\mathcal{H}$, where $t > S_{\mathcal{H}}(C)$; which contradicts the definition of $S_{\mathcal{H}}(C)$.

The required set D is defined as $D = (HS(\mathcal{H}) \setminus F) \cup \bigcup_{C \subset F} HS(\mathcal{H}_C)$.

If a hyperedge F^* in $\mathcal{F}(\mathcal{H}) \setminus \{F\}$ intersects with F, then it must intersect with $HS(\mathcal{H}_C)$ for some $C \subset F$; otherwise F^* intersects with $HS(\mathcal{H}) \setminus F$. So, each hyperedge in $\mathcal{F}(\mathcal{H}) \setminus \{F\}$, intersects with D. Now, we bound the size of D. Since $|HS(\mathcal{H})| \leq k$

and $|HS(\mathcal{H}_C)| \leq dS_{\mathcal{H}}(C))$, we have

$$\begin{aligned} |D| &\leq |HS(\mathcal{H})| + \left|\bigcup_{C \subset F} HS(\mathcal{H}_C)\right| \\ &\leq k + \sum_{C \subset F} dS_{\mathcal{H}}(C) \\ &\leq k + 2^d \cdot d \cdot 10dk \leq 2^{d+5}d^2k. \end{aligned}$$

The last inequality follows from the fact that F does not contain any large core. □

With respect to the set D, we define another event $\mathcal{E}_2 \supseteq \mathcal{E}_1$ and we bound $\mathbb{P}(\mathcal{E}_2)$. Let

$$\mathcal{E}_2\text{: } \exists\, z \in D \text{ such that } h(z) = h(y) \text{ for some } y \in F.$$

So,

$$\mathbb{P}(\mathcal{E}_2) \leq d\frac{|D|}{\beta k} = \frac{d \cdot 2^{d+5}d^2k}{\beta k} = \frac{d^3 2^{d+5}}{\beta} < \frac{1}{10}.$$

The last inequality holds as $\beta = 100d^3 2^{d+5}$. Putting everything together,

$$\begin{aligned} \mathbb{P}(F \notin \mathcal{F}(\mathcal{H}_1) | F \in \mathcal{F}_s) &\leq \sum_{u,v \in F: u \neq v} \mathbb{P}(h(u) = h(v)) + \mathbb{P}(\mathcal{E}_1) \\ &\leq \frac{d^2}{\beta k} + \mathbb{P}(\mathcal{E}_2) \\ &\leq \frac{d^2}{\beta k} + \frac{1}{10} \\ &< \frac{1}{2}. \end{aligned}$$

□

Proof of Claim 5.22. Without loss of generality, we will prove the statement for the graph $\mathcal{H}_1$. Let $h : U(\mathcal{H}) \to [\beta k]$ be the random hash function used in the sampling of $\mathcal{H}_1$.

Let $\mathcal{S}$ be the sunflower with core C and $\mathcal{F}'$ be the arbitrary set of $10dk$ hyperedges corresponding to sunflower $\mathcal{S}$. Let us consider a partition of a subset of $\{F \backslash C : F \in \mathcal{F}'\}$ into equivalence classes $T_1, \ldots, T_t$ such that

- $\bigcup_{x \in F_1 \backslash C} h(x) = \bigcup_{x \in F_2 \backslash C} h(x)$ if F_1 and F_2 belong to the same equivalence class, and
- $\left(\bigcup_{x \in F_1 \backslash C} h(x)\right) \cap \left(\bigcup_{x \in F_2 \backslash C} h(x)\right) = \emptyset$ if F_1 and F_2 belong to different equivalence classes.

Before proceeding further, we have the following observation.

Observation 5.24. $\mathbb{P}(t > 2k) \geq \frac{3}{4}$.

Proof. For $F \in \mathcal{F}'$, let X_F be the indicator random variable that takes value 1 if and only if $\left(\bigcup_{x \in F_1 \backslash C} h(x)\right) \cap \left(\bigcup_{x \in F_2 \backslash C} h(x)\right) \neq \emptyset$ for some $F_1 \in \mathcal{F}' \setminus F$. Observe that $t \geq 10dk - X$, where

$$X = \sum_{F \in \mathcal{F}'} X_F.$$

Observe

$$\begin{aligned} \mathbb{P}(X_F = 1) &\leq \sum_{F_1 \in \mathcal{F}'} \mathbb{P}\left(\left(\bigcup_{x \in F_1 \backslash C} h(x)\right) \cap \left(\bigcup_{x \in F_2 \backslash C} h(x)\right) \neq \emptyset\right) \\ &\leq 10dk \cdot d^2 \cdot \frac{1}{\beta k} \leq \frac{1}{20}. \end{aligned} \tag{5.1}$$

So, $\mathbb{E}[X] \leq \frac{1}{20} \cdot 10dk \leq \frac{dk}{2}$. Now,

$$\begin{aligned} \mathbb{P}\left(t \leq 2k\right) &= \mathbb{P}\left(X \geq 10dk - 2k\right) && (\because t = 10dk - X) \\ &\leq \frac{\mathbb{E}[X]}{10dk - 2k} && \text{(Markov Ineqality)} \\ &< \frac{1}{4} && (\mathbb{E}[X] \leq \frac{dk}{2}) \end{aligned}$$

□

Let $h(T) = \bigcup_{x \in F} h(x)$, where $F \in T$. Let $\mathcal{T}$ be the equivalence classes such that the following holds for each $T \in \mathcal{T}$, $h(T) \cap \left(\bigcup_{x \in HS(\mathcal{H}) \setminus C} h(x) \right) = \emptyset$. As $|HS(\mathcal{H})| \leq k$, $|\mathcal{T}| > k$ holds with probability at least $3/4$. Now, consider the following observation.

Observation 5.25. For each $T \in \mathcal{T}$, there exists a hyperedge F in $\mathcal{H}_1$ such that $\bigcup_{x \in F \setminus C} h(x) = h(T)$ with probability at least $1 - \frac{1}{100k}$.

Proof. Consider the set of hyperedges $\mathcal{F}'' = \{F : \bigcup_{x \in F \setminus C} h(x) = h(T)\}$. Any edge outside $\mathcal{F}''$ has one vertex z such that $z \in HS(\mathcal{H}) \setminus C$ or $h(z) \in h(T')$ for some $T' \in \mathcal{T} \setminus \{T\}$. By the construction of $\mathcal{T}$ and by the description of the algorithm, there exists a hyperedge F in $\mathcal{H}_1$ such that $\bigcup_{x \in F \setminus C} h(x) = h(T)$ and the following event $\mathcal{E}$ holds. $\mathcal{E} : h(u) = h(v)$ if and only if $u = v$ for all $u, v \in F$.

$$\mathbb{P}(\mathcal{E}^c) \leq \sum_{u,v \in F} \mathbb{P}(h(u) = h(v)) \leq \frac{d^2}{\beta k} \leq \frac{1}{100k}.$$

So, $\mathbb{P}(\mathcal{E}) \geq 1 - \frac{1}{100k}$. □

From the above Observation, there exist at least $|\mathcal{T}|$ hyperedges in $\mathcal{H}_1$ that form a sunflower with C with probability at least $1 - \frac{1}{100k}(k+1) \geq \frac{49}{50}$. As $\mathbb{P}(|\mathcal{T}| > k) \geq \frac{3}{4}$, $S_{\mathcal{H}_1}(C) > k$ holds with probability $\frac{49}{50} \cdot \frac{3}{4} > \frac{1}{2}$.

□

Now, we have all the ingredients to prove Lemma 5.12.

The proof of Lemma 5.12

Proof of Lemma 5.12. First, since $\widehat{\mathcal{H}}$ is a subgraph of $\mathcal{H}$, a minimum hitting set of $\mathcal{H}$ is also a hitting set of $\widehat{\mathcal{H}}$. To prove this Lemma, it remains to show that when $|HS(\mathcal{H})| \leq k$, then a minimum hitting set of $\widehat{\mathcal{H}}$ is also a hitting set of $\mathcal{H}$. By Lemma 5.20, it is true that with high probability $\mathcal{F}_s \subseteq \mathcal{F}(\widehat{\mathcal{H}})$ and $S_{\widehat{\mathcal{H}}}(C) > k$ if $C \in \mathcal{C}'$. It is enough to show that

when $\mathcal{F}_s \subseteq \mathcal{F}(\widehat{\mathcal{H}})$ and $S_{\widehat{\mathcal{H}}}(C) > k,\ \forall C \in \mathcal{C}'$, then a minimum hitting set of $\widehat{\mathcal{H}}$ is also a minimum hitting set of $\mathcal{H}$.

First we show that each significant core intersects with $HS(\mathcal{H})$. Suppose there exists a significant core C that does not intersect with $HS(\mathcal{H})$. Let $\mathcal{S}$ be a r-sunflower in $\mathcal{H}$, $r > k$, such that C is the core of $\mathcal{S}$. Then each of the r petals of $\mathcal{S}$ must intersect with $HS(\mathcal{H})$. But the petals of any sunflower are disjoint. This implies $|HS(\mathcal{H})| \geq r > k$, which is a contradiction. So, each significant core intersects with $HS(\mathcal{H})$. As large cores are significant, each large core also intersects with $HS(\mathcal{H})$.

Let us consider a subhypergraph of $\mathcal{H}$, say $\tilde{\mathcal{H}}_1$, with the following definition. Take a large core C_1 in $\mathcal{H}$ that contains a significant core C_2 as a subset. Let $\mathcal{S}_1$ be a sunflower with core C_1. Let $\mathcal{S}_2$ be a sunflower with core C_2 that has more than k petals. Note that there can be at most one hyperedge F_1 of $\mathcal{S}_1$ that is also present in $\mathcal{S}_2$. We delete all hyperedges participating in $\mathcal{S}_1$ except F_1. The remaining hyperedges remain the same as in $\mathcal{H}$. Notice that a hitting set of $\tilde{\mathcal{H}}_1$ is also a hitting set of $\mathcal{H}$; the significant core C_2 remains significant in $\tilde{\mathcal{H}}_1$. Thus, any hitting set of $\tilde{\mathcal{H}}_1$ must intersect with C and therefore, must hit all the hyperedges of $\mathcal{S}_1$. We can think of this as a reduction rule, where the input hypergraph and the output hypergraph have the same sized minimum hitting sets. Let $\tilde{\mathcal{H}}$ be a hypergraph obtained after applying the above reduction rule exhaustively on $\mathcal{H}$. The following properties must hold for $\tilde{\mathcal{H}}$: (i) $HS(\mathcal{H}) = HS(\tilde{\mathcal{H}})$, (ii) all large cores in $\tilde{\mathcal{H}}$ do not contain significant cores as subsets. (iii) all hyperedges of $\mathcal{F}_s$ in $\mathcal{H}$ are still present in $\tilde{\mathcal{H}}$.

By Lemma 5.20, it is also true with high probability that $S_{\widehat{\mathcal{H}}}(C) > k$ when C is a large core of $\tilde{\mathcal{H}}$ that does not contain any significant core as a subset. Note that the arguments in Lemma 5.20 can be made for such large cores without significant cores in $\tilde{\mathcal{H}}$. Thus, we continue the arguments with the assumption that $S_{\widehat{\mathcal{H}}}(C) > k$ when C is a large core of $\tilde{\mathcal{H}}$ that does not contain any significant core as a subset.

Now we show that when $|HS(\mathcal{H})| \leq k$, $HS(\tilde{\mathcal{H}}) = HS(\widehat{\mathcal{H}})$. We know that $\mathcal{F}_s \subseteq \mathcal{F}(\tilde{\mathcal{H}})$. That is, any edge that does not contain any large core as a subset, is present in $\tilde{\mathcal{H}}$. Each hyperedge in $\mathcal{F}_s$ must be covered by any hitting set of $\mathcal{H}$, as well as any hitting set of $\tilde{\mathcal{H}}$ and $\widehat{\mathcal{H}}$. Now, it is enough to argue that a hyperedge $F \in \mathcal{F}(\tilde{\mathcal{H}}) \setminus \mathcal{F}_s$, must be

covered by any hitting set of $\widehat{\mathcal{H}}$. Note that each $F \in \mathcal{F}(\tilde{\mathcal{H}}) \setminus \mathcal{F}_s$ contains a large core, say $\widehat{C}$, which does not contain a significant core as a subset. By our assumption, $\widehat{C}$ is a significant core in $\widehat{\mathcal{H}}$ and therefore, must be hit by any hitting set of $\widehat{\mathcal{H}}$.

Putting everything together, when $|HS(\mathcal{H}|) \leq k$, each edge in $\mathcal{H}$ is covered by any hitting set of $\widehat{\mathcal{H}}$. Thus, $HS(\mathcal{H}) = HS(\widehat{\mathcal{H}})$. □

5.5 Algorithms for d-DECISION-HITTING-SET

In this Section, we will prove the following result.

Theorem 5.26. *d-DECISION-HITTING-SET can be solved with $\widetilde{\mathcal{O}}_d(k^{2d^2})$ GPIS queries.*

Note that the above result together with the algorithm for d-HITTING-SET that makes $\widetilde{\mathcal{O}}_d\left(k^d \log n\right)$ GPIS queries (Theorem 5.2(i)), implies an algorithm for d-DECISION-HITTING-SET that makes $\widetilde{\mathcal{O}}_d\left(\min\left\{k^d \log n, k^{2d^2}\right\}\right)$ GPIS queries proving the result in Theorem 5.2(ii).

Proof of Theorem 5.26. By Observation 5.4, it is enough to give an algorithm that solves d-DECISION-HITTING-SET with probability at least $2/3$ by using $\mathcal{O}\left(k^{2d^2}\right)$ GPIS queries.

We choose a random hash function $h : U(\mathcal{H}) \to [\gamma k^{2d}]$ [5], where $\gamma = 1009^d d^2$ (recall from Section 2.1 that choosing a said random hash function is about coloring the vertices uniformly and independently at random). Let $U_i = \{u \in U(\mathcal{H}) : h(u) = i\}$, where $i \in \left[\gamma k^{2d}\right]$. Note that U_is form a partition of $U(\mathcal{H})$, where some of the U_is can be empty. We make a GPIS query with input $(U_{i_1}, \ldots, U_{i_d})$ for each $1 \leq i_1 < \ldots < i_d \leq \gamma k^{2d}$ such that $U_{i_j} \neq \emptyset$ for all $j \in [d]$. Recall that the output of a GPIS query is Yes or No. We create a hypergraph $\widehat{\mathcal{H}}$ where we create a vertex for each part U_i, $i \in [\gamma k^{2d}]$. By abuse of notations, we will denote by

$$U(\widehat{\mathcal{H}}) = \left\{U_1, \ldots, U_{\gamma k^{2d}}\right\}$$

[5] For us (in this chapter) "choose a random hash function $h : V \to [N]$", means that each vertex in V is colored with one of the N colors uniformly and independently at random.

and

$$\mathcal{F}(\widehat{\mathcal{H}}) = \{(U_{i_1}, \ldots, U_{i_d}) : \text{ GPIS answers "yes" to the input } (U_{i_1}, \ldots, U_{i_d})\}.$$

Observe that we make $\mathcal{O}\left(k^{2d^2}\right)$ queries to the GPIS oracle. We find $HS(\widehat{\mathcal{H}})$ and report $|HS(\mathcal{H})| \leq k$ if and only if $\left|HS(\widehat{\mathcal{H}})\right| \leq k$.

For the hitting set $HS(\mathcal{H})$, consider the set $S' = \{U_i \mid \exists u \in HS(\mathcal{H}), h(u) = i\}$. Then S' is a hitting set for $\widehat{\mathcal{H}}$. So, $\left|HS(\widehat{\mathcal{H}})\right| \leq |HS(\mathcal{H})|$, and if $|HS(\mathcal{H})| \leq k$, then $\left|HS(\widehat{\mathcal{H}})\right| \leq k$. Now, the correctness of our query procedure follows directly from the following claim.

Claim 5.27. *If* $\left|HS(\widehat{\mathcal{H}})\right| \leq k$*, then* $|HS(\mathcal{H})| \leq k$ *with probability at least* $2/3$.

The remaining part of the proof will prove the above claim.

Let $\mathcal{R}$ be a fixed k-representative set corresponding to $\mathcal{H}$ obtained from Proposition 5.5 and let $\mathcal{H}'$ be a set system obtained from $\mathcal{R}$ as described in Corollary 5.6. Consider the set $U(\mathcal{H}')$. Note that $|\mathcal{F}(\mathcal{H}')| \leq \binom{k+d}{d}$ and $|U(\mathcal{H}')| \leq d \cdot \binom{k+d}{d}$. Let $\mathcal{E}_1$ be the event that all the vertices in $U(\mathcal{H}')$ are uniquely colored, i.e., $\mathcal{E}_1$: $h(u) = h(v)$ if and only if $u = v$, where $u, v \in U(\mathcal{H}')$.

Now we lower bound the probability of the event $\mathcal{E}_1$. As usual, let $\mathcal{E}_1^c$ denote the complement of the event $\mathcal{E}_1$. Therefore,

$$\mathbb{P}(\mathcal{E}_1^c) \leq \sum_{u,v \in U(\mathcal{H}')} \mathbb{P}(h(u) = h(v)) \leq \sum_{u,v \in U(\mathcal{H}')} \frac{1}{\gamma k^{2d}} \leq \frac{|U(\mathcal{H}')|^2}{\gamma k^{2d}} < \frac{1}{3}.$$

So, $\mathbb{P}(\mathcal{E}_1) \geq \frac{2}{3}$. Let Prop be the property that for each $F \in \mathcal{F}(\mathcal{H}')$, there is an "e

From the definition of the GPIS query oracle, observe that the property Prop is true whenever the event $\mathcal{E}_1$ occurs. If we show that the occurrence of Prop implies that $|HS(\mathcal{H})| \leq k$ if and only if $\left|HS(\widehat{\mathcal{H}})\right| \leq k$, we are done.

For the rest of the proof, assume that Prop holds. Let us define a function $f : U(\widehat{\mathcal{H}}) \to U(\mathcal{H}') \cup \{\psi\}$ as follows. For each $i \in [\gamma k^{2d}]$, if $h(u) = i$ and $u \in U(\mathcal{H}')$, then $f(U_i) = u$. Otherwise, $f(U_i) = \psi$.

Let $\left|HS(\widehat{\mathcal{H}})\right| = k' \leq k$. Let $HS(\widehat{\mathcal{H}}) = \{X_1, \ldots, X_{k'}\} \subseteq U(\widehat{\mathcal{H}})$. Consider the vertex set $U' = \{f(X_i) : i \in [k'], f(X_i) \neq \psi\} \subseteq U(\mathcal{H}')$ which is of size at most k. As $HS(\widehat{\mathcal{H}})$ is a hitting set of $\widehat{\mathcal{H}}$, U' covers all the hyperedges present in $\mathcal{F}(\mathcal{H}')$. Hence by Corollary 5.6, $|HS(\mathcal{H})| \leq k$. □

5.6 Lower bound for d-DECISION-HITTING-SET

We will prove the following result in this Section.

Theorem 5.28 (Restatement of Theorem 5.3)**.** *Let $n, k, d \in \mathbb{N}$ with $d \leq k \leq n$. Any algorithm, with* GPIS *query access to a hypergraph $\mathcal{H}$ having n vertices, that decides whether $HS(\mathcal{H}) \leq k$ or $HS(\mathcal{H}) \geq k+1$ with probability $2/3$, makes at least $\Omega\left(\binom{k+d}{d}\right)$ queries.*

We use the framework by Eden and Rosenbaum [ER18] to prove the above Theorem 5.28 via a reduction from DISJOINTNESS$_N$ problem in the Yao's two party communication model. Recall from Section 2.2, in the DISJOINTNESS$_N$ problem, we have two players Alice and Bob, where Alice has a vector $\mathbf{x} \in \{0,1\}^N$ and Bob has a vector $\mathbf{y} \in \{0,1\}^N$. The goal of the DISJOINTNESS$_N$ problem is for Alice and Bob to communicate bits between each other following a pre-decided protocol in order to decide if $\mathbf{x}$ and $\mathbf{y}$ intersect or not. Given two vectors $\mathbf{x}$ and $\mathbf{y}$ in $\{0,1\}^N$, we say $\mathbf{x}$ and $\mathbf{y}$ *intersect* if there exists $i \in [N]$ such that $x_i = y_i = 1$[6]. Otherwise, we say $\mathbf{x}$ and $\mathbf{y}$ are *disjoint*. From Theorem 2.10 in Section 2.2, the communication complexity of DISJOINTNESS$_N$ is $\Omega(N)$ [KN97]. The lower bound holds even if it is known from beforehand that either $\mathbf{x}$ and $\mathbf{y}$ are disjoint, or there exists exactly one $i \in [N]$ such that $x_i = y_i = 1$, see [KN97].

Proof of Theorem 5.28. Let $\mathbf{x} \in \{0,1\}^N$ and $\mathbf{y} \in \{0,1\}^N$, where $N = \binom{k+d}{d}$ be the inputs of Alice and Bob, respectively. Moreover, assume that either $\mathbf{x}$ and $\mathbf{y}$ are disjoint or there exists exactly one $i \in [N]$ such that $x_i = y_i = 1$. Fix a bijection $\phi : N \to \Sigma_d$,

[6]For a vector $\mathbf{z} \in \{0,1\}^N$, z_i denotes the i-th coordinate of the vector $\mathbf{z}$.

where Σ_d denote the collection of all d-sized subsets of $[k+d]$. Let $\mathcal{H}(\mathbf{x}, \mathbf{y})$ be the hypergraph (with $[n]$ as the vertex set), that can be uniquely determined from $\mathbf{x}$ and $\mathbf{y}$, having hyperedges according to the following rule: for each $i \in [N]$, the particular combination of $\binom{k+d}{d}$ indicated by $\phi(i)$ is a hyperedge in $\mathcal{H}(\mathbf{x}, \mathbf{y})$ if at least one of x_i and y_i is 0. Note that no hyperedge in $\mathcal{H}(\mathbf{x}, \mathbf{y})$ contains a vertex from $[n] \setminus [k+d]$. Moreover, $\mathcal{H}(\mathbf{x}, \mathbf{y})$ satisfies the following observation because of the particular nature of $\mathbf{x}$ and $\mathbf{y}$.

Observation 5.29. (i) There exists at most one d-sized subset of $[k+d]$ that is not a hyperedge in $\mathcal{H}(\mathbf{x}, \mathbf{y})$.

(ii) If $\mathbf{x}$ and $\mathbf{y}$ are disjoint then each d-sized subsets of $[k+d]$ is a hyperedge in $\mathcal{H}(\mathbf{x}, \mathbf{y})$, and therefore the minimum size of any hitting set of $\mathcal{H}(\mathbf{x}, \mathbf{y})$ is $k+1$. Otherwise, there is exactly one d-sized subset of $[k+d]$ that is not a hyperedge in $\mathcal{H}(\mathbf{x}, \mathbf{y})$, and therefore $\mathcal{H}(\mathbf{x}, \mathbf{y})$ has a hitting set of size k.

The above observation follows from the construction of $\mathcal{H}(\mathbf{x}, \mathbf{y})$ along with the fact that either $\mathbf{x}$ and $\mathbf{y}$ are disjoint or there exists exactly one $i \in [N]$ such that $x_i = y_i = 1$.

To reach a contradiction, assume that there exists an algorithm ALG that makes $o\left(\binom{k+d}{d}\right)$ GPIS query access to $\mathcal{H}(\mathbf{x}, \mathbf{y})$ and decides whether $HS(\mathcal{H}(\mathbf{x}, \mathbf{y})) \leq k$ or $HS(\mathcal{H}(\mathbf{x}, \mathbf{y})) = k+1$. Now we give a protocol for DISJOINTNESS$_N$ with $o\left(\binom{k+d}{d}\right) = o(N)$ bits of communication. Alice and Bob run ALG on $\mathcal{H}(\mathbf{x}, \mathbf{y})$. Let ALG ask for GPIS query with some input instance $A_1, \ldots, A_d$. Note that $A_1, \ldots, A_d$ are non-empty and pairwise disjoint. Without loss of generality, we can assume that $A_1, \ldots, A_d \subset [k+d]$ as no hyperedge in $\mathcal{H}(\mathbf{x}, \mathbf{y})$ contains any vertex from $[n] \setminus [k+d]$. Now, we describe how Alice and Bob simulate each GPIS query by communicating at most 2 bits.

At least one A_i has at least two vertices from $[k+d]$: By Observation 5.29 (i), in this case, there exists a hyperedge having a vertex in each A_i. So, Alice and Bob can answer to any such GPIS query without any communication.

Each A_i is a set of singleton vertex from $[k+d]$: In this case, Alice and Bob need to determine whether the vertices in $A = \bigcup_{i=1}^{d} A_i \subseteq [k+d]$ form a hyperedge in

$\mathcal{H}(\mathbf{x}, \mathbf{y})$. Let $j = \phi^{-1}(A)$. From the description of $\mathcal{H}(\mathbf{x}, \mathbf{y})$, A is a hyperedge if and only if at least one of x_j and y_j is 0. So, Alice and Bob can know the answer to any such GPIS query by communicating their bits at j-th index, which is 2 bits of communication.

Hence, Alice and Bob can simulate algorithm ALG by using $o(N)$ bits of communication. After simulating ALG, Alice and Bob reports $\mathbf{x}$ and $\mathbf{y}$ intersect if ALG reports that $HS(\mathcal{H}(\mathbf{x}, \mathbf{y})) \leq k$. Otherwise, if ALG reports that $HS(\mathcal{H}(\mathbf{x}, \mathbf{y})) = k+1$, Alice and Bob report $\mathbf{x}$ and $\mathbf{y}$ are disjoint. The correctness of the protocol for DISJOINTNESS$_N$ follows from the existence of algorithm ALG and Observation 5.29 (ii). □

5.7 Discussion

In this chapter, we proved that the query complexities of d-DECISION-HITTING-SET and d-HITTING-SET problems, using GPIS query, to be $\widetilde{\mathcal{O}}_d\left(\min\left\{k^d \log n, k^{2d^2}\right\}\right)$ and $\widetilde{\mathcal{O}}_d\left(k^d \log n\right)$ respectively. We proved an almost matching lower bound of $\Omega\left(\binom{k+d}{d}\right)$ for both of these problems.

We think that the $\log n$ term in the query complexity of d-DECISION-HITTING-SET is not required, and therefore, we believe that the query complexity of d-DECISION-HITTING-SET using GPIS query should be $\widetilde{\Theta}_d\left(k^d\right)$. Unlike the d-DECISION-HITTING-SET problem, we believe that the query complexity of d-HITTING-SET should be $\widetilde{\Theta}_d\left(k^d \log n\right)$.

Chapter 6

Streaming Algorithm for Graph Deletion Problems

Contents

6.1 Introduction

We have discussed the streaming models for graph problems in Chapter 1 (see Section 1.1.2). In streaming algorithms, a graph is presented as a sequence of edges. In the simplest of this model, we have a stream of edge arrivals, where each edge adds to the graph seen so far, or may include a dynamic mixture of arrivals and departures of edges. In either case, the primary objective is to quickly answer some basic questions over the current state of the graph, such as finding a (maximal) matching over the current graph edges, or finding a (minimum) vertex cover, while storing only a small amount of information. In the most restrictive model, we only allow $\mathcal{O}(\log^{O(1)} n)$ bits of space for storage. However, using standard techniques from communication complexity one can show that most problems do not admit such algorithms. Thus one relaxes this notion and defines what is called a *semi-streaming model*, which allows $\mathcal{O}(n \log^{O(1)} n)$ bits of space. This model has been extremely successful for graph streaming algorithms and a plethora of non-trivial algorithms have been designed in this model [AKL16, GVV17, KKSV17]. There is a vast literature on graph streaming and we refer to the survey by McGregor [McG14a] for more details.

6.1.1 The parameterization problems

The theme of this chapter is parameterized streaming algorithms. Recall the basic notions of parameterized complexity and FPT that we discussed in Section 5.1. Given the definition of FPT for parameterized problems, it is natural to seek an efficient algorithm

for the corresponding parameterized streaming versions to allow $\mathcal{O}(f(k)\log^{O(1)} n)$ bits of space, where f is an arbitrary computable function depending on the parameter k. This chapter discusses about the following parameterized problems.

$\mathcal{F}$-SUBGRAPH DELETION
Input: A graph G, a family $\mathcal{F}$ of connected graphs, and a non-negative integer k.
Output: Does there exist a set $X \subset V(G)$ of k vertices such that $G \setminus X$ does not contain any graph from $\mathcal{F}$ as a subgraph?

$\mathcal{F}$-MINOR DELETION
Input: A graph G, a family $\mathcal{F}$ of connected graphs, and a non-negative integer k.
Output: Does there exist a set $X \subset V(G)$ of k vertices such that $G \setminus X$ does not contain any graph from $\mathcal{F}$ as a minor?

FVS
Input: A graph G and a non-negative integer k.
Output: Does there exist a set $X \subset V(G)$ of k vertices such that $G \setminus X$ does not contain any cycle?

ECT
Input: A graph G and a non-negative integer k.
Output: Does there exist a set $X \subset V(G)$ of k vertices such that $G \setminus X$ does not contain any cycle of even length?

OCT
Input: A graph G and a non-negative integer k.
Output: Does there exist a set $X \subset V(G)$ of k vertices such that $G \setminus X$ does not contain any cycle of odd length, i.e., $G \setminus X$ is bipartite?

TD
Input: A graph G and a non-negative integer k.
Output: Does there exist a set $X \subset V(G)$ of k vertices such that $G \setminus X$ does not contain any triangle?

CVD
Input: A graph G and a non-negative integer k.
Output: Does there exist a set $X \subset V(G)$ of k vertices such that $G \setminus X$ is a cluster graph, i.e., $G \setminus X$ does not contain any induced P_3?

6.1.2 Parametrized streaming algorithm

There are several ways to formalize the parameterized streaming question, and in literature certain natural models are considered. The basic case is when the input of a given problem consists of a sequence of edge arrivals only, for which one seeks a parameterized streaming algorithm (PSA). It is more challenging when the input stream is dynamic, and contains both deletions and insertions of edges. In this case one seeks a dynamic parameterized streaming algorithm (DPSA). Notice that when an edge in the matching is deleted, we sometimes need substantial work to repair the solution and have to ensure that the algorithm has enough information to do so, while keeping only a bounded amount of working space. If we are promised that at every timestamp there is a solution of cost k, then we seek a promised dynamic parameterized streaming algorithm (PDPSA). These notions were formalized in the following two papers [CCHM15, CCE$^+$16] and several results for VERTEX COVER and MAXIMUM MATCHING were presented there. Unfortunately, this relaxation to $\mathcal{O}(f(k)\log^{O(1)} n)$ bits of space does not buy us too many new results. Most of the problems for which parameterized streaming algorithms are known are "local problems". Other local problems like CLUSTER VERTEX DELETION and TRIANGLE DELETION do not have positive results. Also, problems that require some global checking – such as FEEDBACK VERTEX SET, EVEN CYCLE TRANSVERSAL, ODD CYCLE TRANSVERSAL etc. remain elusive. In fact, one can show that, when edges of the graph arrive in an arbitrary order, using reductions from communication complexity all of the above problems will require $\Omega(n)$ space even if we allow a constant number of passes over the data stream [CCE$^+$16].

The starting point of this work is the above mentioned $\Omega(n)$ lower bounds on basic graph problems. We ask the most natural question – how do we deconstruct these

intractability results? When we look deeper we realize that, to the best of our knowledge, the only parameter that has been used in parameterized streaming algorithms is *the size of the solution that we are seeking* in most of the works. Indeed this is the most well-studied parameter, but there is no reason to only use solution size as a parameter. The only works that came to our notice using structural parameters in parameterized streaming were [EHL$^+$18, CCE$^+$16] – they used bounded arboricity.

In parameterized complexity, when faced with obstacles, we either study a problem with respect to parameters larger than the solution size or consider some structural parameters. We export this approach to parameterized streaming algorithms with focus to graph deletion problems. This is our main conceptual contribution, that is, to introduce the concept of structural parameterizations, in our case vertex cover size, to the study of parameterized streaming algorithms.

What parameters to use? In parameterized complexity, after solution size and treewidth, arguably the most notable structural parameter is *vertex cover size* K [CFK$^+$15, FJP14]. For all the vertex deletion problems that we consider in this thesis, a vertex cover is also a solution. Thus, the vertex cover size K is always larger than the solution size k for all the above problems. We do a thorough study of vertex deletion problems from the view point of parameterized streaming in all known models and show dichotomy when moving across parameters and streaming models. The main *conceptual* contribution of this work is to carry forward the use of structural parameters in parameterized streaming algorithms as done earlier in [EHL$^+$18, CCE$^+$16].

Streaming models considered in this chapter The models that we consider are: (1) EDGE ARRIVAL (EA) model; (2) DYNAMIC EDGE ARRIVAL (DEA) model; (3) VERTEX ARRIVAL (VA) model; and (4) ADJACENCY LIST (AL) model. For formal definitions of these models, see Section 1.1.2.

What problems to study? We study the streaming complexity of parameterized versions of $\mathcal{F}$-SUBGRAPH DELETION, $\mathcal{F}$-MINOR DELETION and CLUSTER VERTEX DELETION (CVD). These problems are one of the most well studied ones in parameterized complexity [Cai96, CM15, CM16, FJP14, FLM$^+$16, FLMS12, KLP$^+$16, KP14, Mar10,

RSV04, Tho10] and have led to development of the field. The parameters we consider in this thesis are (i) the solution size k and (ii) the size K of the vertex cover of the input graph G. In $\mathcal{F}$-SUBGRAPH DELETION, $\mathcal{F}$-MINOR DELETION and CVD, the objective is to decide whether there exists $X \subset V(G)$ of size at most k such that $G \setminus X$ has no graphs in $\mathcal{F}$ as a subgraph, has no graphs in $\mathcal{F}$ as a minor and has no induced P_3, respectively. $\mathcal{F}$-SUBGRAPH DELETION, $\mathcal{F}$-MINOR DELETION and CVD are interesting due to the following reasons. FEEDBACK VERTEX SET (FVS), EVEN CYCLE TRANSVERSAL (ECT), ODD CYCLE TRANSVERSAL (OCT) and TRIANGLE DELETION (TD) are special cases of $\mathcal{F}$-SUBGRAPH DELETION when $\mathcal{F} = \{C_3, C_4, C_5, \ldots\}$, $\mathcal{F} = \{C_3, C_5, \ldots\}$, $\mathcal{F} = \{C_4, C_6, \ldots\}$ and $\mathcal{F} = \{C_3\}$, respectively. FVS is also a special case of $\mathcal{F}$-MINOR DELETION when $\mathcal{F} = \{C_3\}$. CVD is different as we are looking for induced structures.

6.1.3 Our results

Let a graph G and a non-negative integer k be the inputs to the graph problems we consider. Notice that for $\mathcal{F}$-SUBGRAPH DELETION, $\mathcal{F}$-MINOR DELETION and CVD, $K \geq k$. Interestingly, the *parameter K also has different effects on the above mentioned problems in the different streaming models*. We show that structural parameters help to obtain efficient parameterized streaming algorithms for some of the problems, while no such effect is observed for other problems. This throws up the more general and deeper question in parameterized streaming complexity of classification of problems based on the different graph streaming models and different parameterization. We believe that our results and concepts will be instrumental in opening up the avenue for such studies in future.

In particular, we obtain a range of streaming algorithms as well as lower bounds on streaming complexity for the problems we consider. Informally, for a streaming model $\mathcal{M}$ and a parameterized problem Π, if there is a p-pass randomized streaming algorithm for Π that uses $\mathcal{O}(\ell)$ space then we say that Π is $(\mathcal{M}, \ell, p)$-streamable. Similarly, if there

is no p-pass algorithm using $o(\ell)$ bits[1] of storage then Π is said to be $(\mathcal{M}, \ell, p)$-hard. For formal definitions please refer to Section 6.2. When we omit p, it means we are considering one pass of the input stream. The highlight of our results are captured by the $\mathcal{F}$-SUBGRAPH DELETION, $\mathcal{F}$-MINOR DELETION and CVD problems.

Theorem 6.1. *Consider $\mathcal{F}$-SUBGRAPH DELETION in the AL model. Parameterized by solution size k, $\mathcal{F}$-SUBGRAPH DELETION is $(\text{AL}, \Omega(n \log n))$-hard. However, when parameterized by vertex cover K, $\mathcal{F}$-SUBGRAPH DELETION is $\left(\text{AL}, \mathcal{O}\left(\Delta(\mathcal{F}) \cdot K^{\Delta(\mathcal{F})+1}\right)\right)$-streamable. Here $\Delta(\mathcal{F})$ is the maximum degree of any graph in $\mathcal{F}$.*

The above Theorem is in contrast to results shown in [CCE+16]. First, we would like to point out that to the best of our knowledge this is the first set of results on hardness in the AL model. The results in [CCE+16] showed that $\mathcal{F}$-SUBGRAPH DELETION is $(\text{EA}, \Omega(n))$-hard. A hardness result in the AL model implies one in the EA model (Refer to Section 6.2). Thus, our result (Proof in Theorem 6.19) implies a stronger lower bound for $\mathcal{F}$-SUBGRAPH DELETION particularly in the EA model. On the positive side, we show that $\mathcal{F}$-SUBGRAPH DELETION parameterized by the vertex cover size K, is $\left(\text{AL}, \Delta(\mathcal{F}) \cdot K^{\Delta(\mathcal{F})+1}\right)$-streamable (Proof in Theorem 6.12).

Our hardness results are obtained from reductions from well- known problems in *communication complexity*. The problems we reduced from are INDEX_n, DISJ_n and PERM_n (Please refer to Section 6.5.1 for details). In order to obtain the algorithm, one of the main technical contributions of this work is the introduction of the COMMON NEIGHBOR problem which plays a crucial role in designing streaming algorithms in this chapter. We show that $\mathcal{F}$-SUBGRAPH DELETION and many of the other considered problems, like $\mathcal{F}$-MINOR DELETION parameterized by vertex cover size K, have a unifying structure that can be solved via COMMON NEIGHBOR, when the edges of the graph are arriving in the AL model. In COMMON NEIGHBOR, the objective is to obtain a subgraph H of the input graph G such that the subgraph contains a maximal matching M of G. Also, for each pair of vertices $a, b \in V(M)$ [2], the edge (a, b) is present

[1] It is standard in streaming that lower bound results are in bits, and the upper bound results are in words.

[2] $V(M)$ denotes the set of all vertices present in the matching M

in H if and only if $(a, b) \in E(G)$, and *enough* [3] common neighbors of all subsets of at most $\Delta(\mathcal{F})$ vertices of $V(M)$ are retained in H. Using structural properties of such a subgraph, called the *common neighbor subgraph*, we show that it is enough to solve $\mathcal{F}$-SUBGRAPH DELETION on the common neighbor subgraph. Similar algorithmic and lower bound results can be obtained for $\mathcal{F}$-MINOR DELETION. The following theorem can be proven using Theorem 6.15 in Section 6.3 and Theorem 6.19 in Section 6.5.

Theorem 6.2. *Consider $\mathcal{F}$-MINOR DELETION in the AL model. Parameterized by solution size k, $\mathcal{F}$-MINOR DELETION is $(\text{AL}, \Omega(n \log n))$-hard. However, when parameterized by vertex cover K, $\mathcal{F}$-MINOR DELETION is $\left(\text{AL}, \mathcal{O}\left(\Delta(\mathcal{F}) \cdot K^{\Delta(\mathcal{F})+1}\right)\right)$-streamable. Here $\Delta(\mathcal{F})$ is the maximum degree of any graph in $\mathcal{F}$.*

The result on CVD is stated in the following Theorem.

Theorem 6.3. *Parameterized by solution size k,* CVD *is $(\text{VA}, \Omega(n))$-hard. However, when parameterized by vertex cover K,* CVD *is $\left(\text{DEA}, \mathcal{O}\left(K^2 \log^4 n\right)\right)$-streamable.*

The CVD problem behaves very differently from the above two problems. We show that the problem is (VA, n)-hard (Theorem 6.21). In contrast, in [CCE+16] the (EA, n)-hardness for the problem was shown, and we are able to extend this result to the VA model (Refer to Section 6.2 for relations between the models considered). Surprisingly, when we parameterize by K, CVD is $(\text{DEA}, K^2 \log^4 n)$-streamable (Theorem 6.16). In fact, this implies $(\mathcal{M}, K^2 \log^4 n)$-streamability for $\mathcal{M} \in \{\text{AL}, \text{VA}, \text{EA}\}$. To design our algorithm, we build on the sampling technique for VERTEX COVER [CCE+16] to solve CVD in DEA model. Our analysis of the sampling technique exploits the structure of a cluster graph.

Though we have mentioned the main algorithmic and lower bound results in the above theorems, we have a list of other algorithmic and lower bound results in the different streaming models. The full list of results are summed up in Table 6.1. To understand the full strength of our contribution, we request the reader to go to Section 6.2 to see the relations between different streaming models and the notion of *hardness* and *streamability.*

[3] By enough, we mean $\mathcal{O}(K)$ in this case.

Problem	Parameter	AL model	VA model	EA/DEA model
$\mathcal{F}$-SUBGRAPH DELETION	k	(AL, $n \log n$)-hard (AL, $n/p, p$)-hard	(VA, $n \log n$)-hard (VA, $n/p, p$)-hard	(EA, $n \log n$)-hard (EA, $n/p, p$)-hard†
	K	(AL, $\Delta(\mathcal{F}) \cdot K^{\Delta(\mathcal{F})+1}$)-str.* (Theorem 6.12)	(VA, $n/p, p$)-hard	(EA, $n/p, p$)-hard
$\mathcal{F}$-MINOR DELETION	k	(AL, $n \log n$)-hard (AL, $n/p, p$)-hard	(VA, $n \log n$)-hard (VA, $n/p, p$)-hard	(EA, $n \log n$)-hard (EA, $n/p, p$)-hard
	K	(AL, $\Delta(\mathcal{F}) \cdot K^{\Delta(\mathcal{F})+1}$)-str.* (Theorem 6.15)	(VA, $n/p, p$)-hard	(EA, $n/p, p$)-hard
FVS, ECT, OCT	k	(AL, $n \log n$)-hard (AL, $n/p, p$)-hard	(VA, $n \log n$)-hard (VA, $n/p, p$)-hard	(EA, $n \log n$)-hard (EA, $n/p, p$)-hard†
	K	(AL, K^3)-str.* (Corollary 6.13)	(VA, $n/p, p$)-hard	(EA, $n/p, p$)-hard
TD	k	OPEN	(VA, $n \log n$)-hard (VA, $n/p, p$)-hard	(EA, $n \log n$)-hard (EA, $n/p, p$)-hard†
	K	(AL, K^3)-str.* (Corollary 6.13)	(VA, $n/p, p$)-hard	(EA, $n/p, p$)-hard
CVD	k	OPEN	(VA, $n/p, p$)-hard	(EA, $n/p, p$)-hard†
	K	(AL, $K^2 \log^4 n$)-str. (Theorem 6.16)	(VA, $K^2 \log^4 n$)-str.	(DEA, $K^2 \log^4 n$)-str.

Table 6.1: A summary of our results. "str." means streamable. The results marked with † in Table 6.1 are lower bound results of Chitnis et al. [CCE+16]. The other lower bound results are ours, some of them being improvements over the lower bound results of Chitnis et al. [CCE+16]. The full set of lower bound results for FVS, ECT, OCT are proven in Theorem 6.19. The lower bound results for TD and CVD are proven in Theorem 6.20 and Theorem 6.21, respectively. Notice that the lower bound results depend only on n. The hardness results are even stronger than what is mentioned in the above table. The nuances are mentioned in respective Theorems 6.19, 6.20, 6.21 in Section 6.5. Algorithmic results marked $*$ are deterministic.

6.1.4 Other related works

Problems in class P have been extensively studied in streaming complexity [McG14a] in the last decade. Recently, there has been a lot of interest in studying streaming complexity of NP-hard problems like HITTING SET, SET COVER, MAX CUT and MAX CSP [GVV17, KKSV17, AKL16]. Structural parameters have been considered to study MATCHING in streaming [BGM+19,EHL+18,MV18,MV16,CJMM17,CCE+16]. Fafianie and Kratsch [FK14] were the first to study parameterized streaming complexity of NP-hard problems like d-HITTING SET and EDGE DOMINATING SET in graphs. Chitnis *et al.* [CCHM15, CCE+16] developed a sampling technique to design efficient parame-

terized streaming algorithms for promised variants of VERTEX COVER, d-HITTING SET problem, b-MATCHING etc. They also proved lower bounds for problems like $\mathcal{G}$-FREE DELETION, $\mathcal{G}$-EDITING, CLUSTER VERTEX DELETION etc. [CCE$^+$16].

Organisation of the chapter

Section 6.2 contains preliminary definitions. The algorithms for COMMON NEIGHBOR, $\mathcal{F}$-SUBGRAPH DELETION and $\mathcal{F}$-MINOR DELETION are given in Section 6.3. Our algorithm for CVD is described in Section 6.4. The lower bound results are mentioned in Section 6.5. Section 6.6 concludes this chapter with a discussion.

6.2 Preliminaries

In this section, we explore the relative order of the power of the graph streaming models in use in this chapter. This information allows us to prove our result for one model so that it carries over to other models. Apart from that, we mention some preliminary notations that we make use of.

6.2.1 Notion of streamability and hardness

Let Π be a parameterized graph problem that takes as input a graph on n vertices and a parameter k. Let $f : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ be a *computable* function. For a model $\mathcal{M} \in \{\text{DEA, EA, VA, AL}\}$, whenever we say that *an algorithm $\mathcal{A}$ solves Π with complexity $f(n,k)$ in model $\mathcal{M}$*, we mean $\mathcal{A}$ is a randomized algorithm that for any input instance of Π in model $\mathcal{M}$ gives the correct output with probability $2/3$ and has streaming complexity $f(n,k)$.

Definition 6.4. A parameterized graph problem Π, that takes an n-vertex graph and a parameter k as input, is $\Omega(f)$ *p-pass hard* in the EDGE ARRIVAL model, or in short Π is (EA, f, p)-*hard*, if there does not exist any p-pass streaming algorithm of streaming complexity $\mathcal{O}(f(n,k))$ bits that can solve Π in model $\mathcal{M}$.

Analogously, (DEA, f, p)-*hard*, (VA, f, p)-*hard* and (AL, f, p)-*hard* are defined.

Definition 6.5. A graph problem Π, that takes an n-vertex graph and a parameter k as input, is $\mathcal{O}(f)$ p-*pass streamable* in EDGE ARRIVAL model, or in short Π is (EA, f, p)-*streamable* if there exists a p-pass streaming algorithm of streaming complexity $\mathcal{O}(f(n, k))$ words [4] that can solve Π in EDGE ARRIVAL model.

(DEA, f, p)-*streamable*, (VA, f, p)-*streamable* and (AL, f, p)-*streamable* are defined analogously. For simplicity, we refer to $(\mathcal{M}, f, 1)$-hard and $(\mathcal{M}, f, 1)$-streamable as $(\mathcal{M}, f)$-hard and $(\mathcal{M}, f)$-*streamable*, respectively, where $\mathcal{M} \in \{\text{DEA}, \text{EA}, \text{VA}, \text{AL}\}$.

6.2.2 Relation between streaming models

Definition 6.6. Let $\mathcal{M}_1, \mathcal{M}_2 \in \{\text{DEA,EA,VA,AL}\}$ be two streaming models, $f : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ be a computable function, and $p \in \mathbb{N}$.

(i) If for any parameterized graph problem Π, $(\mathcal{M}_1, f, p)$-hardness of Π implies $(\mathcal{M}_2, f, p)$-hardness of Π, then we say $\mathcal{M}_1 \leq_h \mathcal{M}_2$.

(ii) If for any parameterized graph problem Π, $(\mathcal{M}_1, f, p)$-streamability of Π implies $(\mathcal{M}_2, f, p)$-streamability of Π, then we say $\mathcal{M}_1 \leq_s \mathcal{M}_2$.

Now, from Definitions 6.4, 6.5 and 6.6, we have the following Observation.

Observation 6.7. $\text{AL} \leq_h \text{EA} \leq_h \text{DEA}$; $\text{VA} \leq_h \text{EA} \leq_h \text{DEA}$; $\text{DEA} \leq_s \text{EA} \leq_s \text{VA}$; $\text{DEA} \leq_s \text{EA} \leq_s \text{AL}$.

This observation has the following implication. If we prove a lower (upper) bound result for some problem Π in model $\mathcal{M}$, then it also holds in any model $\mathcal{M}'$ such that $\mathcal{M} \leq_h \mathcal{M}'$ ($\mathcal{M} \leq_s \mathcal{M}'$). For example, if we prove a lower bound result in AL or VA model, it also holds in EA and DEA model; if we prove an upper bound result in DEA model, it also holds in EA, VA and AL model. In general, there is no direct connection between AL and VA. In AL and VA, the vertices are exposed in an arbitrary order. However, we can say the following when the vertices arrive in a fixed (known) order.

[4] It is usual in streaming that the lower bound results are in bits, and the upper bound results are in words.

Observation 6.8. Let AL′ (VA′) be the restricted version of AL (VA), where the vertices are exposed in a fixed (known) order. Then AL′ $\leq_h$ VA′ and VA′ $\leq_s$ AL′.

Now, we remark the implication of the relation between different models discussed in this section to our results mentioned in Table 6.1.

Remark 6.1. In Table 6.1 (that gives our full set of results of this chapter), the lower bound results in VA and AL hold even if we know the sequence in which vertices are exposed, and the upper bound results hold even if the vertices arrive in an arbitrary order. In general, the lower bound in the AL model for some problem Π does not imply the lower bound in the VA model for Π. However, our lower bound proofs in the AL model hold even if we know the order in which vertices are exposed. So, the lower bounds for FVS, ECT, OCT in the AL model imply the lower bound in the VA model. By Observations 6.7 and 6.8, we will be done by showing a subset of the algorithmic and lower bound results mentioned in the Table 6.1.

6.2.3 Notations

The union of two graphs G_1 and G_2 with $V(G_1) = V(G_2)$, is $G_1 \cup G_2$, where $V(G_1 \cup G_2) = V(G_1) = V(G_2)$ and $E(G_1 \cup G_2) = E(G_1) \cup E(G_2)$. For $X \subseteq V(G)$, $G \setminus X$ is the subgraph of G induced by $V(G) \setminus X$. The degree of a vertex $u \in V(G)$, is denoted by $\deg_G(u)$. The maximum and average degrees of the vertices in G are denoted as $\Delta(G)$ and $\Delta_{av}(G)$, respectively. For a family of graphs $\mathcal{F}$, $\Delta(\mathcal{F}) = \max_{F \in \mathcal{F}} \Delta(F)$. A graph F is a *subgraph* of a graph G if $V(F) \subseteq V(G)$ and $E(F) \subseteq E(G)$ be the set of edges that can be formed only between vertices of $V(F)$. A graph F is said to be a *minor* of a graph G if F can be obtained from G by deleting edges and vertices and by contracting edges. The neighborhood of a vertex $v \in V(G)$ is denoted by $N_G(v)$. For $S \subseteq V(G)$, $N_G(S)$ denotes the set of vertices in $V(G) \setminus S$ that are *neighbors of every vertex in* S. A vertex $v \in N_G(S)$ is said to be a common neighbor of S in G. The size of any minimum vertex cover in G is denoted by $\mathrm{VC}(G)$. A cycle on the sequence of vertices $v_1, \ldots, v_n$ is denoted as $\mathcal{C}(v_1, \ldots, v_n)$. For a matching M in G, the vertices in the matching are denoted by $V(M)$. C_t denotes a cycle of length t. P_t denotes a path having t vertices.

A graph G is said to a *cluster* graph if G is a disjoint union of cliques, that is, no three vertices of G can form an induced P_3.

6.3 Deterministic algorithms in the AL model

In this Section, we show that $\mathcal{F}$-SUBGRAPH DELETION is $(\text{AL}, \Delta(\mathcal{F}) \cdot K^{\Delta(\mathcal{F})+1})$-streamable when the vertex cover of the input graph is parameterized by K. This will imply that FVS, ECT, OCT and TD parameterized by vertex cover size K, are (AL, K^3)-streamable. This complements the results in Theorems 6.19 and 6.20 (in Section 6.5) that show that the problems parameterized by vertex cover size K are $(\text{VA}, n/p, p)$-hard (see also Table 1). Note that by Observation 6.7, this also implies that the problems parameterized by vertex cover size K are $(\mathcal{M}, n/p, p)$-hard when $\mathcal{M} \in \{\text{EA},\text{DEA}\}$. Finally, we design an algorithm for $\mathcal{F}$-MINOR DELETION that is inspired by the algorithm for $\mathcal{F}$-SUBGRAPH DELETION.

For the algorithm for $\mathcal{F}$-SUBGRAPH DELETION, we define an auxiliary problem COMMON NEIGHBOR and a streaming algorithm for it. This works as a subroutine for our algorithm for $\mathcal{F}$-SUBGRAPH DELETION.

6.3.1 COMMON NEIGHBOR problem

For a graph G and a parameter $\ell \in \mathbb{N}$, H will be called *a common neighbor subgraph* for G if

(i) $V(H) \subseteq V(G)$ such that H has no isolated vertex.

(ii) $E(H)$ contains the edges of a maximal matching M of G along with the edges where both the endpoints are from $V(M)$, such that for all subsets $S \subseteq V(M)$, with $|S| \leq d$, we have $|N_H(S) \setminus V(M)| = \min\{|N_G(S) \setminus V(M)|, \ell\}$. That is, $E(H)$ contains edges to at most ℓ common neighbors of S in $N_G(S) \setminus V(M)$.

In simple words, a common neighbor subgraph H of G contains the subgraph of G induced by $V(M)$ as a subgraph of H for some maximal matching M in G. Also, for

each subset S of at most d vertices in $V(M)$, H contains edges to *sufficiently many* common neighbors of S in G. The parameters $d \leq K$ and ℓ are referred to as the degree parameter and common neighbor parameter, respectively.

The COMMON NEIGHBOR problem is formally defined as follows. It takes as input a graph G with $\text{VC}(G) \leq K$, degree parameter $d \leq K$ and common neighbor parameter ℓ and produces a common neighbor subgraph of G as the output. COMMON NEIGHBOR parameterized by vertex cover size K, admits Algorithm 6.1.

Algorithm 6.1: COMMON NEIGHBOR

Input: A graph G, with $\text{VC}(G) \leq K$, in the AL model, a degree parameter d (which is at most K), and a common neighbor parameter ℓ.
Output: A common neighbor subgraph H of G.

```
begin
    Initialize M = ∅ and V(M) = ∅, where M denotes the current maximal matching.
    Initialize a temporary storage T = ∅.
    for (each vertex u ∈ V(G) exposed in the stream) do
        for (each (u, x) ∈ E(G) in the stream) do
            if (u ∉ V(M) and x ∉ V(M)) then
                Add (u, x) to M and both u, x to V(M).
            if (x ∈ V(M)) then
                Add (u, x) to T.
        if ( If u is added to V(M) during the exposure of u) then
            Add all the edges present in T to E(H).
        else
            for (each S ⊆ V(M) such that |S| ≤ d and (u, z) ∈ T ∀z ∈ S) do
                if (N_H(S) is less than ℓ ) then
                    Add the edges (u, z) ∀z ∈ S to E(H).
        Reset T to ∅.
```

Lemma 6.9. COMMON NEIGHBOR*, with a commmon neighbor parameter ℓ and parameterized by vertex cover size K, is* $(\text{AL}, K^2\ell)$*-streamable.*

Proof. We start our algorithm by initializing $M = \emptyset$ and construct a matching in G that

is maximal under inclusion; See Algorithm 6.1. As $|\text{VC}(G)| \leq K$, $|M| \leq K$. Recall that we are considering the AL model here. Let M_u and M'_u be the maximal matchings just before and after the exposure of the vertex u (including the processing of the edges adjacent to u), respectively. Note that, by construction these partial matchings M_u and M'_u are also maximal matchings in the subgraph exposed so far. The following Lemma will be useful for the proof.

Claim 6.10. *Let $u \in N_G(S) \setminus V(M)$ for some $S \subseteq V(M)$. Then $S \subseteq V(M_u)$, that is, u is exposed, after all the vertices in S are declared as vertices of $V(M)$.*

Proof. Observe that if there exists $x \in S$ such that $x \notin V(M_u)$, then after u is exposed, there exists $y \in N_G(u)$ such that (u, y) is present in M'_u. This implies $u \in V(M'_u) \subseteq V(M)$, which is a contradiction to $u \in N_G(S) \setminus V(M)$. □

Now, we describe what our algorithm does when a vertex u is exposed. The complete pseudocode of our algorithm for COMMON NEIGHBOR is given in Algorithm 6.1. When a vertex u is exposed in the stream, we try to extend the maximal matching M_u. Also, we store all the edges of the form (u, x) such that $x \in V(M_u)$, in a temporary memory T. As $|M_u| \leq K$, we are storing at most $2K$ edges in T. Now, there are the following possibilities.

- If $u \in V(M'_u)$, that is, either $u \in V(M_u)$ or the matching M_u is extended by one of the edges stored in T, then we add all the edges stored in T to $E(H)$.
- Otherwise, for each $S \subseteq V(M_u)$ such that $|S| \leq d$ and $S \subseteq N_G(u)$, we check whether the number of common neighbors of the vertices present in S, that are already stored, is less than ℓ. If yes, we add all the edges of the form (u, z) such that $z \in S$ to $E(H)$; else, we do nothing. Now, we reset T to $\emptyset$.

As $|M| \leq K$, $|V(M)| \leq 2K$. We are storing at most ℓ common neighbors for each $S \subseteq V(M)$ with $|S| \leq d$ and the number of edges having both the endpoints in M is at most $\mathcal{O}(K^2)$, the total amount of space used is at most $\mathcal{O}(K^d \ell)$. □

We call our algorithm described in the proof of Lemma 6.9 and given in Algorithm 6.1, as $\mathcal{A}_{cn}$. The following structural Lemma of the common neighbor subgraph

of G, obtained by algorithm $\mathcal{A}_{cn}$ is important for the design and analysis of streaming algorithms for $\mathcal{F}$-SUBGRAPH DELETION.

Lemma 6.11. *Let G be a graph with $\mathrm{VC}(G) \leq K$ and let F be a connected graph with $\Delta(F) \leq d \leq K$. Let H be the common neighbor subgraph of G with degree parameter d and common neighbor parameter $(d+2)K$, obtained by running the algorithm $\mathcal{A}_{cn}$. Then the following holds in H: For any subset $X \subseteq V(H)$, where $|X| \leq K$, F is a subgraph of $G \setminus X$ if and only if F' is a subgraph of $H \setminus X$, such that F and F' are* isomorphic.

Proof. Let the common neighbor subgraph H, obtained by algorithm $\mathcal{A}_{cn}$, contain a maximal matching M of G. First, observe that since $\mathrm{VC}(G) \leq K$, the size of a subgraph F in G is at most dK. Now let us consider a subset $X \subseteq V(H)$ such that $|X| \leq K$. First, suppose that F' is a subgraph of $H \setminus X$ and F' is isomorphic to F. Then since H is a subgraph of G, F' is also a subgraph of $G \setminus X$. Therefore, $F = F'$ and we are done.

Conversely, suppose F is a subgraph of $G \setminus X$ that is not a subgraph in $H \setminus X$. We show that there is a subgraph F' of $H \setminus X$ such that F' is isomorphic to F. Consider an arbitrary ordering $\{e_1, e_2, \ldots, e_s\} \subseteq (E(G) \setminus E(H)) \cap E(F)$; note that $s \leq |E(F)|$. We describe an iterative subroutine that converts the subgraph F to F' through s steps, or equivalently, through a sequence of isomorphic subgraphs $F_0, F_1, F_2, \ldots F_s$ in G such that $F_0 = F$ and $F_s = F'$.

Let us discuss the consequence of such an iterative routine. Just before the starting of step $i \in [s]$, we have the subgraph F_{i-1} such that F_{i-1} is isomorphic to F and the set of edges in $(E(G) \setminus E(H)) \cap E(F_{i-1})$ is a subset of $\{e_i, e_{i+1}, \ldots, e_s\}$. In step i, we convert the subgraph F_{i-1} into F_i such that F_{i-1} is isomorphic to F_i. Just after the step $i \in [s]$, we have the subgraph F_i such that F_i is isomorphic to F and the set of edges in $(E(G) \setminus E(H)) \cap E(F_i)$ is a subset of $\{e_{i+1}, e_{i+2}, \ldots, e_s\}$. In particular, in the end $F_s = F'$ is a subgraph both in G and H.

Now consider the instance just before step i. We show how we select the subgraph F_i from F_{i-1}. Let $e_i = (u, v)$. Note that $e_i \notin E(H)$. By the definition of the maximal matching M in G, it must be the case that $|\{u, v\} \cap V(M)| \geq 1$. From the construction

of the common neighbor subgraph H, if both u and v are in $V(M)$, then $e_i = (u, v) \in E(H)$. So, exactly one of u and v is present in $V(M)$. Without loss of generality, let $u \in V(M)$. Observe that v is a common neighbor of $N_G(v)$ in G. Because of the maximality of M, each vertex in $N_G(v)$ is present in $V(M)$. Now, as $(u, v) \notin E(H)$, v is not a common neighbor of $N_G(v)$ in H. From the construction of the common neighbor subgraph, H contains $(d + 2)K$ common neighbors of all the vertices present in $N_G(v)$. Of these common neighbors, at most $(d + 1)K$ common neighbors can be vertices in $X \cup F_i$. Thus, there is a vertex v' that is a common neighbor of all the vertices present in $N_G(v)$ in H such that F_{i+1} is a subgraph that is isomorphic to F_i. Moreover, $(E(G) \setminus E(H)) \cap E(F_{i+1}) \subseteq \{e_{i+2}, e_{i+3} \ldots, e_s\}$. Thus, this leads to the fact that there is a subgraph F' in $H \setminus X$ that is isomorphic to the subgraph F in $G \setminus X$. □

6.3.2 Streamability results for $\mathcal{F}$-SUBGRAPH DELETION and $\mathcal{F}$-MINOR DELETION

Our result on COMMON NEIGHBOR leads us to the following streamability result for $\mathcal{F}$-SUBGRAPH DELETION and $\mathcal{F}$-MINOR DELETION. We first discuss the result on $\mathcal{F}$-SUBGRAPH DELETION, which is stated in the following theorem.

Theorem 6.12. *$\mathcal{F}$-SUBGRAPH DELETION parameterized by vertex cover size K is $(\text{AL}, d \cdot K^{d+1})$-streamable, where $d = \Delta(\mathcal{F}) \leq K$.*

Proof. Let (G, k, K) be an input for $\mathcal{F}$-SUBGRAPH DELETION, where G is the input graph, $k \leq K$ is the size of the solution of $\mathcal{F}$-SUBGRAPH DELETION, and the parameter K is at least $\text{VC}(G)$.

Now, we describe the streaming algorithm for $\mathcal{F}$-SUBGRAPH DELETION. First, we run the COMMON NEIGHBOR streaming algorithm described in Lemma 6.9 (and given in Algorithm 6.1) with degree parameter d and common neighbor parameter $(d + 2)K$, and let the common neighbor subgraph obtained be H. We run a traditional FPT algorithm for $\mathcal{F}$-SUBGRAPH DELETION [CFK+15] on H and output YES if and only if the output on H is YES.

Let us argue the correctness of this algorithm. By Lemma 6.11, for any subset $X \subseteq V(H)$, where $|X| \leq K$, $F \in \mathcal{F}$ is a subgraph of $G \setminus X$ if and only if F', such that F' is isomorphic to F', is a subgraph of $H \setminus X$. In particular, let X be a k-sized vertex set of G. As mentioned before, $k \leq K$. Thus, by Lemma 6.11, X is a solution of $\mathcal{F}$-SUBGRAPH DELETION in H if and only if X is a solution of $\mathcal{F}$-SUBGRAPH DELETION in G. Therefore, we are done with the correctness of the streaming algorithm for $\mathcal{F}$-SUBGRAPH DELETION.

The streaming complexity of $\mathcal{F}$-SUBGRAPH DELETION is same as the streaming complexity for the algorithm $\mathcal{A}_{cn}$ from Lemma 6.9 with degree parameter $d = \Delta(\mathcal{F})$ and common neighbor parameter $(d+2)K$. Therefore, the streaming complexity of $\mathcal{F}$-SUBGRAPH DELETION is $\mathcal{O}(d \cdot K^{d+1})$. □

Corollary 6.13. FVS, ECT, OCT *and* TD *parameterized by vertex cover size K are* (AL, K^3)*-streamable due to deterministic algorithms.*

6.3.3 Algorithm for $\mathcal{F}$-MINOR DELETION

Finally, we describe a streaming algorithm for $\mathcal{F}$-MINOR DELETION that works similar to that of $\mathcal{F}$-SUBGRAPH DELETION due to the following proposition and the result is stated in Theorem 6.15.

Proposition 6.14 ([FJP14])**.** *Let G be a graph with F as a minor and* $\text{VC}(G) \leq K$*. Then there exists a subgraph G^* of G that has F as a minor such that $\Delta(G^*) \leq \Delta(F)$ and $V(G^*) \leq V(F) + K(\Delta(F)+1)$.*

Theorem 6.15. $\mathcal{F}$-MINOR DELETION *parameterized by vertex cover size K are* $(\text{AL}, d \cdot K^{d+1})$*-streamable, where $d = \Delta(\mathcal{F}) \leq K$.*

Proof. Let (G, k, K) be an input for $\mathcal{F}$-MINOR DELETION, where G is the input graph, k is the size of the solution of $\mathcal{F}$-MINOR DELETION we are looking for, and the parameter K is such that $\text{VC}(G) \leq K$. Note that, $k \leq K$.

Now, we describe the streaming algorithm for $\mathcal{F}$-MINOR DELETION. First, we run the COMMON NEIGHBOR streaming algorithm described in Lemma 6.9 with degree pa-

rameter d and common neighbor parameter $(d+2)K$, and let the common neighbor subgraph obtained be H. We run a traditional FPT algorithm for $\mathcal{F}$-MINOR DELETION [CFK$^+$15] and output YES if and only if the output on H is YES.

Let us argue the correctness of this algorithm, that is, we prove the following for any $F \in \mathcal{F}$. $G \setminus X$ contains F as a minor if and only if $H \setminus X$ contains F' as a minor such that F and F' are isomorphic, where $X \subseteq V(G)$ is of size at most K. For the only if part, suppose $H \setminus X$ contains F' as a minor. Then since H is a subgraph of G, $G \setminus X$ contains F' as a minor. For the if part, let $G \setminus X$ contains F as a minor. By Proposition 6.14, $G \setminus X$ conatins a subgraph G^* such that G^* contains F as a minor and $\Delta(G^*) \leq \Delta(F)$. Now, Lemma 6.11 implies that $H \setminus X$ also contains a subgraph $\widehat{G^*}$ that is isomorphic to G^*. Hence, $H \setminus X$ contains F' as a monor such that F' is isomorphic to F.

The streaming complexity of the streaming algorithm for $\mathcal{F}$-MINOR DELETION is same as the streaming complexity for the algorithm $\mathcal{A}_{cn}$ from Lemma 6.9 with degree parameter $d = \Delta(\mathcal{F})$ and common neighbor parameter $(d+2)K$. Therefore, the streaming complexity for $\mathcal{F}$-MINOR DELETION is $\mathcal{O}(d \cdot K^{d+1})$. □

6.4 CVD in the DEA model

In this Section, we show that CVD parameterized by vertex cover size K, is $(\text{DEA}, K^2 \log^4 n)$-streamable. By Observation 6.7, this implies $(\mathcal{M}, K^2 \log^4 n)$-streamability for all $\mathcal{M} \in \{\text{EA,VA,AL}\}$. The sketch of the algorithm for CVD parameterized by vertex cover size K in the DEA model is in Algorithm 6.2. The algorithm is inspired by the streaming algorithm for VERTEX COVER [CCE$^+$16]. Before discussing the algorithm, let us discuss some terms.

A family of hash functions of the form $h : [n] \to [m]$ is said to be *pairwise independent hash family* if for a pair $i, j \in [n]$ and a randomly chosen h from the family, $\mathbb{P}(h(i) = h(j)) \leq \frac{1}{m}$. Such a hash function h can be stored efficiently by using $\mathcal{O}(\log n)$ bits [MR95].

ℓ_0-sampler [CF14]: Given a dynamic graph stream, an ℓ_0-sampler does the following: with probability at least $1 - \frac{1}{n^c}$, where c is a positive constant, it produces an edge uniformly at random from the set of edges that have been inserted so far but not deleted. If no such edge exists, ℓ_0-sampler reports NULL. The total space used by the sampler is $\mathcal{O}(\log^3 n)$.

Algorithm 6.2: CVD

Input: A graph G having n vertices in the DEA model, with vertex cover size at most $K \in \mathbb{N}$, solution parameter $k \in \mathbb{N}$, such that $k \leq K$.

Output: A set $X \subset V(G)$ of k vertices such that $G \setminus X$ is a cluster graph if such a set exists. Otherwise, the output is NULL

begin
- From a pairwise independent family of hash functions that map $V(G)$ to $[\beta K]$, choose $h_1, \ldots, h_{\alpha \log n}$ such that each h_i is chosen uniformly and independently at random, where α and β are suitable *large* constants.
- For each $i \in [\alpha \log n]$ and $r, s \in [\beta K]$, initiate an ℓ_0 sampler $L^i_{r,s}$.
- **for** *(each (u, v) in the stream)* **do**
  - Irrespective of (u, v) being inserted or deleted, give the respective input to the ℓ_0-samplers $L^i_{h_i(u),h_i(v)}$ for each $i \in [\alpha \log n]$.
- For each $i \in [\alpha \log n]$, construct a subgraph H_i by taking the outputs of all the ℓ_0-samplers corresponding to the hash function h_i.
- Construct $H = H_1 \cup \cdots \cup H_{\alpha \log n}$.
- Run the classical FPT algorithm for CVD on the subgraph H and solution size bound k [CFK$^+$15].
- **if** *(H has a solution S of size at most k)* **then**
  - Report S as the solution to G.
- **else**
  - Report NULL

Theorem 6.16. CVD*, parameterized by vertex cover size K, is* (DEA, $K^2 \log^4 n$)*-streamable.*

Proof. Let G be the input graph of the streaming algorithm and by assumption $\mathrm{VC}(G) \leq K$. Let $h_1, \ldots, h_{\alpha \log n}$ be a set of $\alpha \log n$ pairwise independent hash functions such that each h_i chosen uniformly and independently at random from a pairwise independent family of hash functions, where $h : V(G) \to [\beta K]$, α and β are suitable constants.

For each hash function h_i and pair $r, s \in [\beta K]$, let $G^i_{r,s}$ be the subgraph of G induced by the vertex set $\{v \in V(G) : h_i(v) \in \{r, s\}\}$. For the hash function h_i and for each pair $r, s \in [\beta K]$, we initiate an ℓ_0 sampler for the dynamic stream restricted to the subgraph $G^i_{r,s}$. Therefore, there is a set of $\mathcal{O}(K^2)$ ℓ_0-samplers $\{L^i_{r,s} : r, s \in [\beta K]\}$ corresponding to the hash function h_i. Now, we describe what our algorithm does when an edge is either inserted or deleted. A pseudocode of our algorithm for CVD is given in Algorithm 6.2. When an edge (u, v) arrives in the stream, that is (u, v) is inserted or deleted, we give the respective input to $L^i_{h_i(u),h_i(v)}$, where $i \in [\alpha \log n]$. At the end of the stream, for each $i \in [\alpha \log n]$, we construct a subgraph H_i by taking the outputs of all the ℓ_0-samplers corresponding to the hash function h_i. Let $H = H_1 \cup \cdots \cup H_{\alpha \log n}$. We run the classical FPT algorithm for CVD on the subgraph H and solution size bound k [CFK+15], and report YES to CVD if and only if we get YES as answer from the above FPT algorithm on H. If we output YES , then we also give the solution on H as our solution to G.

The correctness of the algorithm needs an existential structural result on G (Claim 6.17) and the fact that if there exists a set $X \subset V(G)$ whose deletion turns H into a cluster graph, then the same X deleted from G will turn it into a cluster graph with high probability (Claim 6.18).

Claim 6.17. *There exists a partition $\mathcal{P}$ of $V(G)$ into $Z_1, \ldots, Z_t, I$ such that the subgraph induced in G by each Z_i, is a clique with at least 2 vertices, and the subgraph induced by I is the empty graph.*

Proof. We start with a partition which may not have the properties of the claim and modify it iteratively such that the final partition does have all the properties of the Claim. Let us start with a partition $\mathcal{P}$ that does not satisfy the given condition. First, if there exists a part Z_i having one vertex v, we create a new partition by adding v to I. Next, if there exists a part Z_i having at least two vertices and the subgraph induced by Z_i is not a clique, then we partition Z_i into smaller parts such that each smaller part is either a clique having at least two vertices or a singleton vertex. We create a new partition by replacing Z_i with the smaller cliques of size at least 2 and adding all the singleton vertices to I.

Now, let $\mathcal{P}'$ be the new partition of $V(G)$ obtained after all the above modifications. In $\mathcal{P}'$, each part except I is a clique of at least two vertices. If the subgraph induced by I has no edges, $\mathcal{P}'$ satisfies the properties in the Claim and we are done. Otherwise, there exists $u, v \in I$ such that $(u, v) \in E(G)$. In this case, we create a new part with $\{u, v\}$, and remove both u and v from I. Note that in the above iterative description, each vertex goes to a new part at most 2 times - (i) it can move at most once from a part Z_i to a smaller part Z_j that is a clique on at least 2 vertices and such a vertex will remain in the same part in all steps afterwards, or it can move at most once from a Z_i to I, and (ii) a vertex can move at most once from I to become a part of a clique Z_i with at least 2 vertices and such a vertex will remain in the same part in all steps after that. Therefore, this process is finite and there is a final partition that we obtain in the end. This final partition has all the properties of the claim. □

Claim 6.18. *Let $X \subset V(H)$ be such that $H \setminus X$ is a cluster graph. Then $G \setminus X$ is a cluster graph with high probability.*

Proof. Consider a partition $\mathcal{P}$ of $V(G)$ into $Z_1, \ldots, Z_t, I$ as mentioned in Claim 6.17. Note that our algorithm does not need to find such a partition. The existence of $\mathcal{P}$ will be used only for the analysis purpose. Let $\mathcal{Z} = \cup_{i=1}^{t} Z_i$. Note that since $\text{VC}(G) \leq K$, each Z_i can have at most $K + 1$ vertices, and it must be true that $t \leq \text{VC}(G) \leq K$. In fact, we can obtain the following stronger bound that $|\mathcal{Z}| \leq 2K$. The total number of vertices in $\mathcal{Z}$ is at most $\text{VC}(G) + t$. Since $t \leq \text{VC}(G) \leq K$, the total number of vertices in $\mathcal{Z}$ is at most $2K$.

A vertex $u \in V(G)$, is said to be of *high degree* if $\deg_G(u) \geq 40K$, and *low degree*, otherwise. Let $V_h \subseteq V(G)$ be the set of all high degree vertices and V_ℓ be the set of low degree vertices in G. Let E_ℓ be the set of edges in G having both the endpoints in V_ℓ. It can be shown [CCE+16] that

(i) **Fact-1:** $|V_h| \leq K$, $E_\ell = \mathcal{O}(K^2)$;

(ii) **Fact-2:** $E_\ell \subseteq E(H)$, and $\deg_H(u) \geq 4K$ for each $u \in V_h$, with probability at least $1 - \frac{1}{n^{\mathcal{O}(1)}}$.

Note that Fact-2 makes our algorithmic result for CVD probabilistic.

Let $\text{CVD}(G) \subset V(G)$ denote a minimum set of vertices such that $G \setminus \text{CVD}(G)$ is a cluster graph. Our parametric assumption says that $|\text{CVD}(G)| \leq \text{VC}(G) \leq K$. Now consider the fact that a graph is a cluster graph if and only if it does not have any induced P_3. First, we show that the high degree vertices in G surely need to be deleted to make it a cluster graph, i.e., $V_h \subseteq \text{CVD}(G)$. Let us consider a vertex $u \in V_h$. As the subgraph induced by I has no edges and $|\mathcal{Z}| \leq 2K$, each vertex in I is of degree at most $|\mathcal{Z}| \leq 2K$. So, u must be in some Z_i in the partition $\mathcal{P}$. As $\deg_G(u) \geq 40K$, using $|\mathcal{Z}| \leq 2K$, u must have at least $38K$ vertices from I as its neighbors in G. Thus, there are at least $19K$ edge disjoint induced P_3's that are formed with u and its neighbors in I. If $u \notin \text{CVD}(G)$, then more than K neighbors of u that are in I must be present in $\text{CVD}(G)$. It will contradict the fact that $|\text{CVD}(G)| \leq \text{VC}(G) \leq K$. Similarly, we can also argue that $V_h \subseteq \text{CVD}(H) = X$ as $\deg_H(u) \geq 4K$ by Fact-2.

Next, we show that an induced P_3 is present in $G \setminus V_h$ if and only if it is present in $H \setminus V_h$. Removal of V_h from G (or H) removes all the induced P_3's in G (or H) having at least one vertex in V_h. Any induced P_3 in $G \setminus V_h$ (or $H \setminus V_h$) must have all of its vertices as low degree vertices. Now, using Fact-2, note that all the edges, in G, between low degree vertices are in H. In other words, an induced P_3 is present in $G \setminus V_h$ if and only if it is present in $H \setminus V_h$. Thus for a set $X \subseteq V(G)$, if $(H \setminus V_h) \setminus X$ is a cluster graph then $(G \setminus V_h) \setminus X$ is also a cluster graph.

Putting everything together, if $X \subseteq V(G)$ is such that $H \setminus X$ is a cluster graph, then $G \setminus X$ is also a cluster graph. □

Coming back to the proof of Theorem 6.16, we are using $\mathcal{O}(\log n)$ hash functions, and each hash function requires a storage of $\mathcal{O}(\log n)$ bits. There are $\mathcal{O}(K^2)$ ℓ_0-samplers for each hash function and each ℓ_0-sampler needs $\mathcal{O}(\log^3 n)$ bits of storage. Thus, the total space used by our algorithm is $\mathcal{O}(K^2 \log^4 n)$. □

6.5 The lower bounds

We finish with lower bounds that complement the upper bound results and complete the picture presented in Table 6.1. Before we explicitly give the statements of the stated lower bound results presented in Table 6.1, we want to note that a lower bound on FEEDBACK VERTEX SET is also a lower bound for $\mathcal{F}$-SUBGRAPH DELETION (deletion of cycles as subgraphs) and $\mathcal{F}$-MINOR DELETION (deletion of 3-cycles as minors). Observe that we will be done by proving the following theorems, and the rest of the lower bound results will follow from Observations 6.7 and 6.8.

Theorem 6.19. FEEDBACK VERTEX SET, EVEN CYCLE TRANSVERSAL *and* ODD CYCLE TRANSVERSAL *are*

(I) $(\text{AL}, n \log n)$*-hard parameterized by solution size* k *and even if* $\Delta_{av}(G) = \mathcal{O}(1)$*,*

(II) $(\text{AL}, n/p, p)$*-hard parameterized by solution size* k *and even if* $\Delta(G) = \mathcal{O}(1)$*, and*

(III) $(\text{VA}, n/p, p)$*-hard parameterized by vertex cover size* K *and even if* $\Delta_{av}(G) = \mathcal{O}(1)$*.*

Theorem 6.20. TD *is*

(I) $(\text{VA}, n \log n)$*-hard parameterized by solution size* k *and even if* $\Delta_{av}(G) = \mathcal{O}(1)$*,*

(II) $(\text{VA}, n/p, p)$*-hard parameterized by solution size* k *and even if* $\Delta(G) = \mathcal{O}(1)$*, and*

(III) $(\text{VA}, n/p, p)$*-hard parameterized by vertex cover size* K *and even if* $\Delta_{av}(G) = \mathcal{O}(1)$*.*

Theorem 6.21. CVD *is* $(\text{VA}, n/p, p)$*-hard parameterized by solution size* k *and even if* $\Delta(G) = \mathcal{O}(1)$*.*

Remark 6.2. (i) The proofs of part (I) of Theorems 6.19 and 6.20 use the lower bound constructions given in [SW15].

(ii) To the best of our knowledge, this is the first set of results on hardness in the AL model.

(iii) The proofs of parts (II) and (III) of Theorems 6.19 and 6.20 use the lower bound constructions given in [CCE$^+$16].

(iv) The proof of Theorem 6.21 uses the lower bound constructions given in [CCE$^+$16].

The following theorems will be proved by using reductions from communication complexity. We need some more results on communication complexity than those discussed in Section 2.2.

6.5.1 A discussion on communication complexity

We have discussed about INDEX$_n$ and DISJ$_n$ (along with their communication complexities) in Section 2.2. Here, we define another problem known as PERMUTATION (PERM$_n$), and state its communication complexity. PERM$_n$ along with INDEX$_n$ and DISJ$_n$ will be useful to establish the lower bounds.

- PERM$_n$ [SW15] : Alice gets a permutation $\pi : [n] \rightarrow [n]$ and Bob gets an index $j \in [n \log n]$. The objective of Bob is to decide the value of PERM$_n(\pi, j)$, defined as the j-th bit in the string of 0's and 1's obtained by concatenating the bit expansions of $\pi(1) \ldots \pi(n)$. In other words, let $\Phi : [n \log n] \rightarrow [n] \times [\log n]$ be a bijective function defined as $\Phi(j) = \left(\lceil \frac{j}{\log n} \rceil, j + \log n - \lceil \frac{j}{\log n} \rceil \times \log n\right)$. For a permutation $\pi : [n] \rightarrow [n]$, Bob needs to determine the value of the γ-th bit of $\pi\left(\lceil \frac{j}{\log n} \rceil\right)$, where $\gamma = \left(j + \log n - \lceil \frac{j}{\log n} \rceil \times \log n\right)$.

Proposition 6.22 ([SW15]). *The one way communication complexity of* PERM$_n$ *is* $\Omega(n \log n)$.

Here, we outline the meaning of reduction from the communication complexity problems to streaming problems.

A note on reduction from INDEX_n, DISJ_n, PERM_n: A reduction from a problem Π_1 in one/two way communication complexity to a problem Π_2 in streaming algorithms is typically as follows: The two players Alice and Bob device a communication protocol for Π_1 that uses a streaming algorithm for Π_2 as a subroutine. Typically in a round of communication, a player gives inputs to the input stream of the streaming algorithm, obtains the compact sketch produced by the streaming algorithm and communicates this sketch to the other player. This implies that a lower bound on the communication complexity of Π_1 also gives a lower bound on the streaming complexity of Π_2.

The following Proposition summarizes a few important consequences of reductions from problems in communication complexity to problems for streaming algorithms:

Proposition 6.23. *(i) If we can show a reduction from* INDEX_n *to a problem* Π *in model* $\mathcal{M}$ *such that the reduction uses a* 1*-pass streaming algorithm of* Π *as a subroutine, then* Π *is* $(\mathcal{M}, n)$*-hard.*

(ii) If we can show a reduction from DISJ_n *to a problem* Π *in model* $\mathcal{M}$ *such that the reduction uses a* 1*-pass streaming algorithm of* Π *as a subroutine, then* Π *is* $(\mathcal{M}, n/p, p)$*-hard, for any* $p \in \mathbb{N}$ *[CCE+16, BGMS18, AMP+06].*

(iii) If we can show a reduction from PERM_n *to a problem* Π *in model* $\mathcal{M}$ *such that the reduction uses a* 1*-pass streaming algorithm of* Π *as a subroutine, then* Π *is* $(\mathcal{M}, n \log n)$*-hard.*

6.5.2 Proofs of Theorems 6.19, 6.20 and 6.21

Proof of Theorem 6.19. The proofs for all three problems are similar. We first consider FEEDBACK VERTEX SET. To begin with, we show the hardness results of FVS for solution size $k = 0$.

Proof of Theorem 6.19 (I). We give a reduction from PERM_n to FVS in the AL model when the solution size parameter $k = 0$. The idea is to build a graph G with $\Delta_{av}(G) =$

[4] Recall that we take n as a power of 2. For $1 \leq i \leq n-1$, the bit expansion of i is the usual bit notation of i using $\log_2 n$ bits; the bit expansion of n is $\log_2 n$ consecutive zeros. For example: Take $n = 32$. The bit expansion of 32 is 100000. We ignore the bit 1 and say that the bit expansion of 32 is 00000.

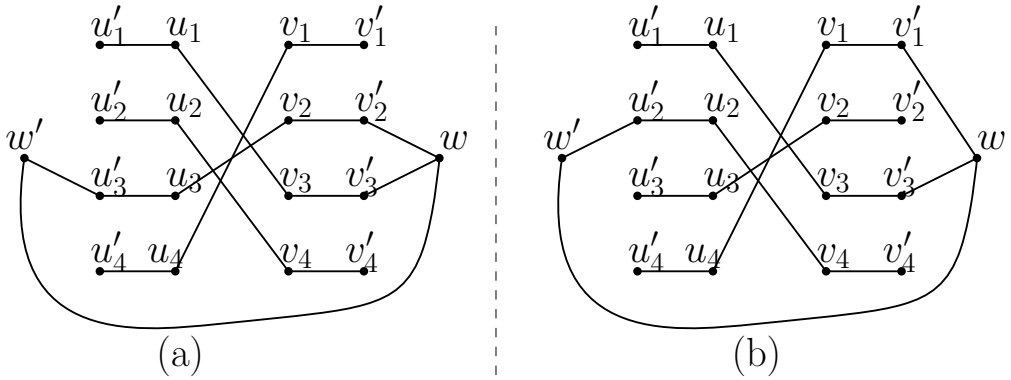

Figure 6.1: Illustration of Proof of Theorem 6.19 (I). Consider $n = 4$. Let $\pi : [4] \to [4]$ such that $\pi(1) = 3, \pi(2) = 4, \pi(3) = 2$ and $\pi(4) = 1$. So the concatenated bit string is 11001001^2. In (a), $j = 5$, $\Phi(j) = (\psi, \gamma) = (3, 1)$, $\text{PERM}_n(\pi, j) = 1$, and G contains a cycle. In (b), $j = 4$, $\Phi(j) = (\psi, \gamma) = (2, 2)$, $\text{PERM}_n(\pi, j) = 0$, and G does not contain a cycle.

$\mathcal{O}(1)$ and construct edges according to the input of PERM_n, such that the output of PERM_n is 0 if and only if G is cycle-free.

Let $\mathcal{A}$ be a one pass streaming algorithm that solves FVS in AL model using $o(n \log n)$ space. Let G be a graph with $4n+2$ vertices $u_1, \ldots, u_n, v_1, \ldots, v_n, u'_1, \ldots, u'_n, v'_1, \ldots, v'_n$, w, w'. Let π be the input of Alice for PERM_n. See Figure 6.1 for an illustration.

Alice's input to $\mathcal{A}$: Alice inputs the graph G first by exposing the vertices $u_1, \ldots, u_n$, $v_1, \ldots, v_n$, sequentially. (i) While exposing the vertex u_i, Alice gives as input to $\mathcal{A}$ the edges $(u_i, u'_i), (u_i, v_{\pi(i)})$; (ii) while exposing the vertex v_i, Alice gives the edges $(v_i, v'_i), (v_i, u_{\pi^{-1}(i)})$ to the input stream of $\mathcal{A}$.

After the exposure of $u_1, \ldots, u_n, v_1, \ldots, v_n$ as per the AL model, Alice sends the current memory state of $\mathcal{A}$, i.e the sketch generated by $\mathcal{A}$, to Bob. Let $j \in [n \log n]$ be the input of Bob and let $(\psi, \gamma) = \Phi(j)$.

Bob's input to $\mathcal{A}$: Bob exposes the vertices $u'_1 \ldots, u'_n, v'_1, \ldots, v'_n, w, w'$, sequentially. (i) While exposing a vertex u'_i where $i \neq \psi$, Bob gives the edge (u'_i, u_i) to the input stream of $\mathcal{A}$; (ii) while exposing u'_ψ, Bob gives the edges (u'_ψ, u_ψ) and (u'_ψ, w'); (iii) while exposing a vertex v'_i, Bob gives the edge (v'_i, v_i), and the edge (v'_i, w) if and only

if bit$(i, \gamma) = 1$; (iv) while exposing w, Bob gives the edge (w, w'), and the edge (w, v'_i) if and only if bit$(i, \gamma) = 1$; (v) while exposing w', Bob gives the edges (w', w) and (w', u'_ψ).

Observe that $\Delta_{av}(G) = \mathcal{O}(1)$. Now we show that the output of FVS is NO if and only if $\text{PERM}_n(\pi, j) = 1$. Recall that $k = 0$.

From the construction, observe that $(w, w'), (w', u'_\psi), (u'_\psi, u_\psi), (u_\psi, v_{\pi(\psi)}), (v_{\pi(\psi)}, v'_{\pi(\psi)}) \in E(G)$. When $\text{PERM}_n(\pi, j) = 1$, the edge $(v'_{\pi(\psi)}, w)$ is present in G. So, G contains the cycle $\mathcal{C}(w, w', u'_\psi, u_\psi, v_{\pi(\psi)}, v'_{\pi(\psi)})$, that is, the output of FVS is NO.

On the other hand, if the output of FVS is NO, then there is a cycle in G. From the construction, the cycle is $\mathcal{C}(w, w', u'_\psi, u_\psi, v_{\pi(\psi)}, v'_{\pi(\psi)})$. As $(v'_{\pi(\psi)}, w)$ is an edge, the γ-th bit of $\pi(\psi)$ is 1, that is $\text{PERM}_n(\pi, j) = 1$. Now by Propositions 6.22 and 6.23(iii), we obtain that FEEDBACK VERTEX SET is $(\text{AL}, n \log n)$-hard even if $\Delta_{av}(G) = \mathcal{O}(1)$ and when $k = 0$. □

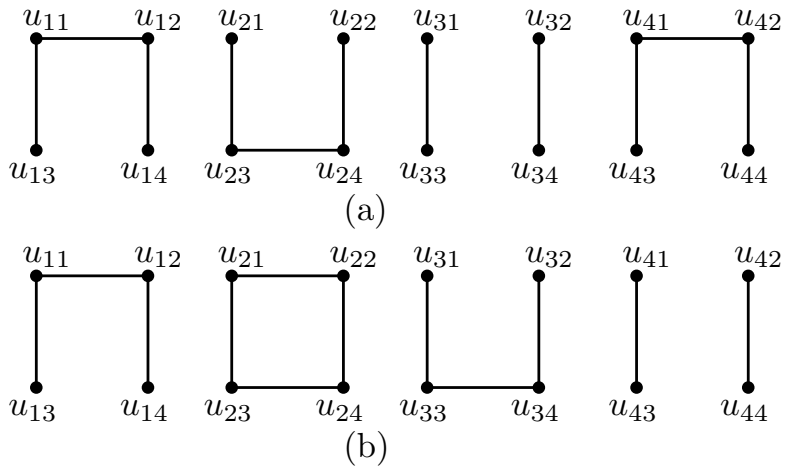

Figure 6.2: Illustration of Proof of Theorem 6.19 (II). Consider $n = 4$. In (a), $\mathbf{x} = 1001$ and $\mathbf{y} = 0100$, that is, $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 1$, and G does not contain a cycle. In (b), $\mathbf{x} = 1100$ and $\mathbf{y} = 0110$, that is, $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$, and G contains a cycle.

Proof of Theorem 6.19 (II). We give a reduction from DISJ_n to FVS in the AL model when the solution size parameter $k = 0$. The idea is to build a graph G with $\Delta(G) =$

$\mathcal{O}(1)$ and construct edges according to the input of DISJ_n, such that the output of DISJ_n is 1 if and only if G is cycle-free.

Let $\mathcal{A}$ be a one pass streaming algorithm that solves FVS in AL model, such that $\Delta(G) = \mathcal{O}(1)$, and the space used is $o(n)$. Let G be a graph with $4n$ vertices $u_{11}, u_{12}, u_{13}, u_{14}, \ldots, u_{n1}, u_{n2}, u_{n3}, u_{n4}$. Let $\mathbf{x}, \mathbf{y}$ be the input of Alice and Bob for DISJ_n, respectively. See Figure 6.2 for an illustration.

Alice's input to $\mathcal{A}$: Alice inputs the graph G by exposing the vertices $u_{11}, u_{12}, u_{21}, u_{22} \ldots, u_{n1}, u_{n2}$, sequentially. (i) While exposing u_{i1}, Alice gives as input to $\mathcal{A}$ the edge (u_{i1}, u_{i3}). Also, Alice gives the edge (u_{i1}, u_{i2}) as input to $\mathcal{A}$ if and only if $x_i = 1$; (ii) while exposing u_{i2}, Alice gives the edge (u_{i2}, u_{i4}) as input to $\mathcal{A}$. Also, Alice gives the edge (u_{i2}, u_{i1}) as input to $\mathcal{A}$ if and only if $x_i = 1$.

After the exposure of $u_{11}, u_{12}, u_{21}, u_{22} \ldots, u_{n1}, u_{n2}$ as per the AL model, Alice sends current memory state of $\mathcal{A}$, i.e. the sketch generated by $\mathcal{A}$, to Bob.

Bob's input to $\mathcal{A}$: Bob exposes the vertices $u_{13}, u_{14}, u_{23}, u_{24} \ldots, u_{n3}, u_{n4}$ sequentially. (i) While exposing u_{i3}, Bob gives the edge (u_{i3}, u_{i1}) as input to $\mathcal{A}$, and gives the edge (u_{i3}, u_{i4}) if and only if $y_i = 1$; (ii) while exposing u_{i4}, Bob gives the edge (u_{i4}, u_{i2}) as input to $\mathcal{A}$, and gives the edge (u_{i4}, u_{i3}) if and only if $y_i = 1$.

Observe that $\Delta(G) \leq 4$. Recall that $k = 0$. Now we show that the output of FVS is NO if and only if $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$.

From the construction, $(u_{i1}, u_{i3}), (u_{i2}, u_{i4}) \in E(G)$, for each $i \in [n]$. If $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$, there exists $i \in [n]$ such that $x_i = y_i = 1$. This implies the edges (u_{i1}, u_{i2}) and (u_{i3}, u_{i4}) are present in G. So, the cycle $\mathcal{C}(u_{i1}, u_{i2}, u_{i3}, u_{i4})$ is present in G, that is, the output of FVS is NO.

Conversely, if the output of FVS is NO, there exists a cycle in G. From the construction, the cycle must be $\mathcal{C}(u_{i1}, u_{i2}, u_{i3}, u_{i4})$ for some $i \in [n]$. As the edges (u_{i1}, u_{i2}) and (u_{i3}, u_{i4}) are present in G, $x_i = y_i = 1$, that is, $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$.

Now by Propositions 6.22 and 6.23(ii), we obtain that FEEDBACK VERTEX SET is $(\text{AL}, n/p, p)$-hard even if $\Delta(G) = \mathcal{O}(1)$ and when $k = 0$. □

Proof of Theorem 6.19 (III). We give a reduction from DISJ_n to FVS in the VA model when the solution size parameter $k = 0$. The idea is to build a graph G with vertex cover size bounded by K and $\Delta(G) = \mathcal{O}(1)$, and construct edges according to the input of DISJ_n, such that the output of DISJ_n is 1 if and only if G is cycle-free.

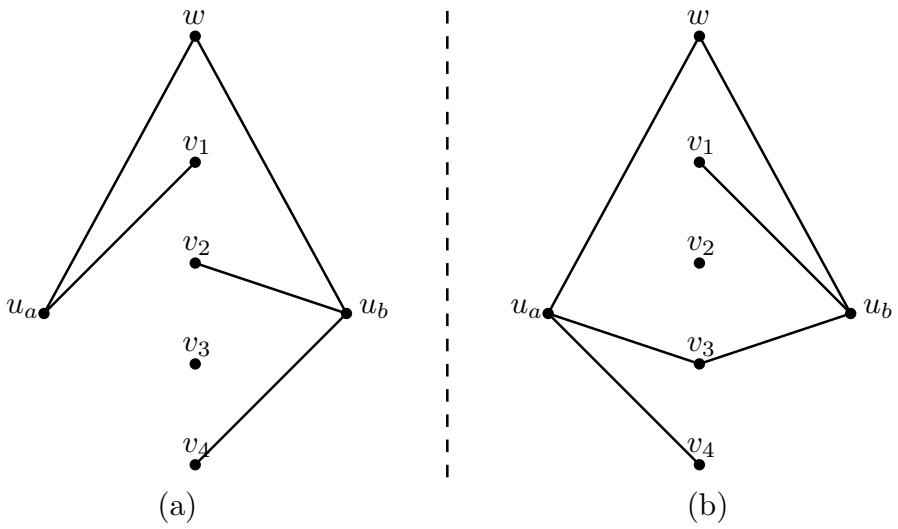

Figure 6.3: Illustration of Proof of Theorem 6.19 (III). Consider $n = 4$. In (a), $\mathbf{x} = 1000$ and $\mathbf{y} = 0101$, that is, $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 1$, and G does not contain a cycle. In (b), $\mathbf{x} = 0011$ and $\mathbf{y} = 1010$, that is, $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$, and G contains a cycle.

Let $\mathcal{A}$ be a one pass streaming algorithm that solves FVS in VA model, such that $\text{VC}(G) \leq K$ and $\Delta_{av}(G) = \mathcal{O}(1)$, and the space used is $o(n)$. Let G be a graph with $n + 3$ vertices $u_a, v_1, \ldots, v_n, u_b, w$. Let $\mathbf{x}, \mathbf{y}$ be the input of Alice and Bob for DISJ_n, respectively. See Figure 6.3 for an illustration.

Alice's input to $\mathcal{A}$: Alice inputs the graph G first by exposing the vertices $u_a, v_1, \ldots, v_n$, sequentially. (i) While exposing u_a, Alice does not give any edge; (ii) while exposing v_i, Alice gives the edge (v_i, u_a), as input to $\mathcal{A}$, if and only if $x_i = 1$.

After the exposure of $u_a, v_1, \ldots, v_n$ as per VA model, Alice sends the current memory state of $\mathcal{A}$, i.e., the sketch generated by $\mathcal{A}$, to Bob.

Bob's input to $\mathcal{A}$: Bob first exposes u_b and then exposes w. (i) While exposing u_b, Bob gives the edge (u_b, v_i) if and only if $y_i = 1$; (ii) while exposing w, Bob gives the edges (w, u_a) and (w, u_b), as inputs to $\mathcal{A}$.

From the construction, observe that $\text{VC}(G) \leq 2 \leq K$ and $\Delta_{av}(G) = \mathcal{O}(1)$. Recall that $k = 0$. Now we show that the output of FVS is NO if and only if $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$.

From the construction, $(u_a, w), (u_b, w) \in E(G)$. If $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$, there exists $i \in [n]$ such that $x_i = y_i = 1$. This implies the edges (u_a, v_i) and (u_b, v_i) are present in G. So, the cycle $\mathcal{C}(u_a, v_i, u_b, w)$ is present in G, that is, the output of FVS is NO.

Conversely, if the output of FVS is NO, there exists a cycle in G. From the construction, the cycle must be $\mathcal{C}(u_a, v_i, u_b, w)$ for some $i \in [n]$. As the edges (u_a, v_i) and (u_b, v_i) are present in G, $x_i = y_i = 1$, that is, $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$.

Now by Propositions 6.22 and 6.23(ii), we obtain that FEEDBACK VERTEX SET parameterized by vertex cover size K is $(\text{VA}, n/p, p)$-hard even if $\Delta_{av}(G) = \mathcal{O}(1)$, and when $k = 0$. □

In each of the above three cases, we can make the reduction work for any k, by adding k vertex disjoint cycles of length 4, i.e. C_4's, to G. In Theorem 6.19 (III), the vertex cover must be bounded. In the given reduction for Theorem 6.19 (III), the vertex cover of the constructed graph is at most 2. Note that by the addition of k edge disjoint C_4's, the vertex cover of the constructed graph in the modified reduction is at most $2k+2$, and is therefore still a parameter independent of the input instance size.

This completes the proof of the Theorem 6.19 with respect to FVS.

If the graph constructed in the reduction, in any of the above three cases for FEEDBACK VERTEX SET, contains a cycle, then it is of even length. Otherwise, the graph is cycle free. Hence, the proof of this Theorem with respect to ECT is same as the proof for FVS.

Similarly, a slight modification can be made to the constructed graph, in all three of the above cases, such that a cycle in the graph is of odd length if a cycle exists. Thereby, the proof of this Theorem with respect to OCT also is very similar to the proof for FVS. □

Proof of Theorem 6.20. We first show the hardness results of TD for $k = 0$ in all three cases.

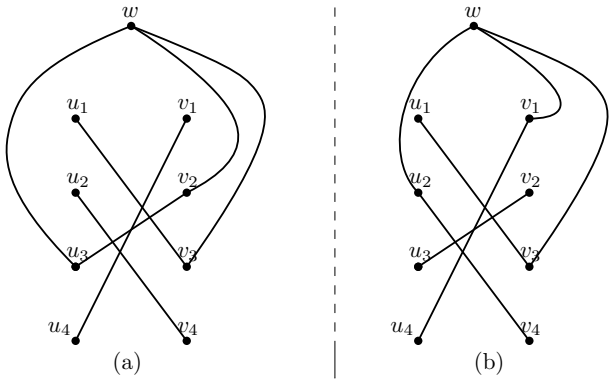

Figure 6.4: Illustration of Proof of Theorem 6.20 (I). Consider $n = 4$. Let $\pi : [4] \to [4]$ such that $\pi(1) = 3, \pi(2) = 4, \pi(3) = 2$, and $\pi(4) = 1$. So the concatenated bit string is 11001001. In (a), $j = 5$, $\Phi(j) = (\psi, \gamma) = (3, 1)$, $\textsc{Perm}_n(\pi, j) = 1$ and G contains a triangle. In (b), $j = 4$, $\Phi(j) = (\psi, \gamma) = (2, 2)$, $\textsc{Perm}_n(\pi, j) = 0$, and G does not contain any triangle.

Proof of Theorem 6.20 (I). We give a reduction from $\textsc{Perm}_n$ to TD when the solution size parameter $k = 0$. Let $\mathcal{A}$ be a one pass streaming algorithm that solves TD in VA model, such that $\Delta_{av}(G) = \mathcal{O}(1)$, and the space used is $o(n \log n)$. Let G be a graph with $2n + 1$ vertices $u_1, \ldots, u_n, v_1, \ldots, v_n, w$. Let π be the input of Alice for $\textsc{Perm}_n$. See Figure 6.4 for an illustration.

Alice's input to $\mathcal{A}$: Alice inputs the graph G by exposing the vertices $u_1, \ldots, u_n, v_1, \ldots, v_n$, sequentially. (i) While exposing the vertex u_i, Alice does not give any edge; (ii) while exposing the vertex v_i, Alice gives the edges $(v_{\pi(i)}, u_i)$ as an input to the stream of $\mathcal{A}$.

After the exposure of $u_1, \ldots, u_n, v_1, \ldots, v_n$ as per the VA model, Alice sends the current memory state of $\mathcal{A}$, i.e. the sketch generated by $\mathcal{A}$, to Bob. Let $j \in [n \log n]$ be the input of Bob and let $(\psi, \gamma) = \Phi(j)$.

Bob's input to $\mathcal{A}$: Bob exposes only the vertex w. Bob gives the edge (w, u_ψ), and the edge (w, v_i) if and only if $\text{bit}(i, \gamma) = 1$, as input to the stream of $\mathcal{A}$.

From the construction, note that $\Delta_{av}(G) = \mathcal{O}(1)$. Recall that $k = 0$. Now we show that, the output of TD is NO if and only if $\text{PERM}_n(\pi, j) = 1$.

From the construction, the edges $(u_\psi, v_{\pi(\psi)})$ and (w, u_ψ) are present in G. If $\text{PERM}_n(\pi, j) = 1$, then $(v_{\pi(\psi)}, w) \in E(G)$. So, there exists a triangle in G, that is, the output of TD is NO.

On the other hand, if the output of TD is NO, then there exists a triangle in G. From the construction, the triangle is formed with the vertices $u_\psi, v_{\pi(\psi)}$ and w. As $(v_{\pi(\psi)}, w) \in E(G)$, the γ-th bit of $\pi(\psi)$ is 1, that is, $\text{PERM}_n(\pi, j) = 1$.

Now by Propositions 6.22 and 6.23(iii), we obtain that TD is $(\text{VA}, n \log n)$-hard even if $\Delta_{av}(G) = \mathcal{O}(1)$, and when $k = 0$. □

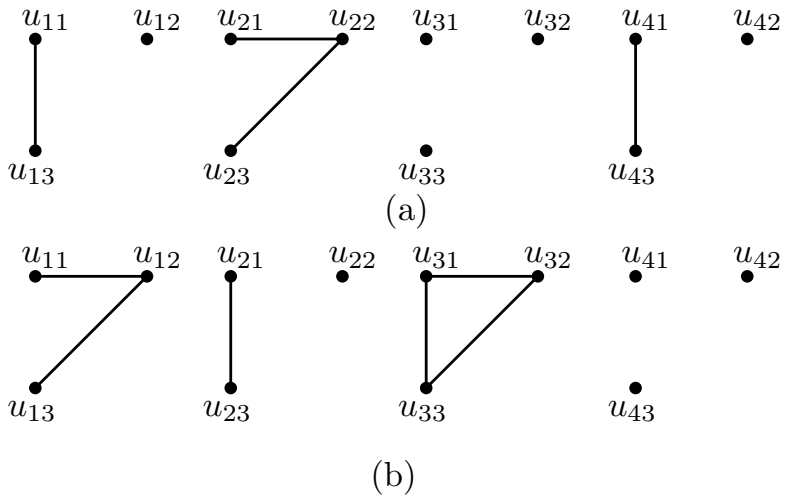

Figure 6.5: Illustration of Proof of Theorem 6.20 (II). Consider $n = 4$. In (a), $\mathbf{x} = 1001$ and $\mathbf{y} = 0100$, that is, $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 1$, and G does not contain any triangle. In (b), $\mathbf{x} = 0110$ and $\mathbf{y} = 1010$, that is, $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$, and G contains a triangle.

Proof of Theorem 6.20 (II). We give a reduction from DISJ_n to TD when the solution size parameter $k = 0$. Let $\mathcal{A}$ be a one pass streaming algorithm that solves TD in VA model, such that $\Delta(G) = \mathcal{O}(1)$, and the space used is $o(n)$. Let G be a graph with $3n$ vertices $u_{11}, u_{12}, u_{13}, \ldots, u_{n1}, u_{n2}, u_{n3}$. Let $\mathbf{x}, \mathbf{y}$ be the input of Alice and Bob for DISJ_n. See Figure 6.5 for an illustration.

Alice's input to $\mathcal{A}$: Alice inputs the graph G first by exposing the vertices u_{11}, u_{12}, u_{21}, u_{22}, ..., u_{n1}, u_{n2}, sequentially. (i) While exposing u_{i1}, Alice does not give any edge; (ii) while exposing u_{i2}, Alice gives the edge (u_{i2}, u_{i1}), if and only if $x_i = 1$, as inputs to $\mathcal{A}$.

After the exposure of $u_{11}, u_{12}, u_{21}, u_{22} \ldots, u_{n1}, u_{n2}$ as per the VA model, Alice sends current memory state of $\mathcal{A}$, i.e. the sketch generated by $\mathcal{A}$, to Bob.

Bob's input to $\mathcal{A}$: Bob exposes the vertices $u_{13}, \ldots, u_{n3}$, sequentially. While exposing u_{i3}, Bob gives the edges (u_{i3}, u_{i1}) and (u_{i3}, u_{i2}) as two inputs to $\mathcal{A}$ if and only if $y_i = 1$.

From the construction, note that $\Delta(G) \leq 2$. Recall that $k = 0$. Now we show that the output of TD is NO if and only if $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$.

If $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$, there exists $i \in [n]$ such that $x_i = y_i = 1$. From the construction, the edges (u_{i2}, u_{i1}), (u_{i3}, u_{i1}) and (u_{i3}, u_{i2}) are present in G. So, there exists a triangle in G, that is, the output of TD is NO.

Conversely, if the output of TD is NO, there exists a triangle in G. From the construction, the triangle is (u_{i1}, u_{i2}, u_{i3}) for some $i \in [n]$. As the edge $(u_{i2}, u_{i1}) \in E(G)$, $x_i = 1$; and as the edges (u_{i3}, u_{i1}) and (u_{i3}, u_{i2}) are in G, $y_i = 1$. So, $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$.

Now by Propositions 6.22 and 6.23(ii), we obtain that TD is $(\text{VA}, n/p, p)$-hard even if $\Delta(G) = \mathcal{O}(1)$, and when $k = 0$. □

Proof of Theorem 6.20 (III). We give a reduction from DISJ_n to TD parameterized by vertex cover size K, where $\mathcal{A}$ is a one pass streaming algorithm that solves TD parameterized by K in VA model such that $\Delta_{av}(G) = \mathcal{O}(1)$, and the space used is $o(n)$. Let G be a graph with $n + 2$ vertices $u_a, v_1, \ldots, v_n, u_b$. Let $\mathbf{x}, \mathbf{y}$ be the input of Alice and Bob for DISJ_n. See Figure 6.6 for an illustration.

Alice's input to $\mathcal{A}$: Alice inputs the graph G first by exposing the vertices $u_a, v_1, \ldots, v_n$ sequentially. (i) While exposing u_a, Alice does not give any edge; (ii) while exposing v_i, Alice gives the edge (v_i, u_a) as input to $\mathcal{A}$ if and only if $x_i = 1$.

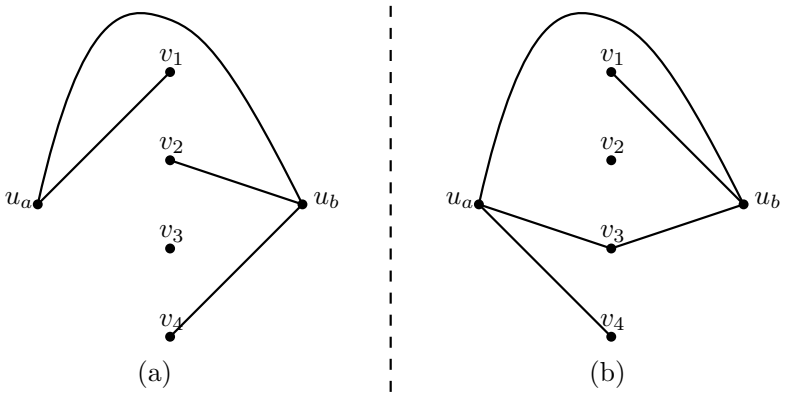

Figure 6.6: Illustration of Proof of Theorem 6.20 (III). Consider $n = 4$. In (a), $\mathbf{x} = 1000$ and $\mathbf{y} = 0101$, that is, $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 1$, and G does not contain any triangle. In (b), $\mathbf{x} = 0011$ and $\mathbf{y} = 1010$, that is, $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$, and G contains a triangle.

After the exposure of $u_a, v_1, \ldots, v_n$ as per the VA model, Alice sends current memory state of $\mathcal{A}$, i.e. the sketch generated by $\mathcal{A}$, to Bob.

Bob's input to $\mathcal{A}$: Bob exposes u_b only. Bob gives the edge (u_b, u_a) unconditionally, and an edge (u_b, v_i) as input to $\mathcal{A}$ if and only if $y_i = 1$.

From the construction, observe that $\text{VC}(G) \leq 2 \leq K$ and $\Delta_{av}(G) = \mathcal{O}(1)$. Recall that $k = 0$. Now we show that the output of TD is NO if and only if $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$.

Observe that $(u_a, u_b) \in E(G)$. If $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$, there exists an $i \in [n]$ such that $x_i = y_i = 1$. From the construction, the edges (v_i, u_a) and (u_b, v_i) are present in G. So, G contains the triangle with vertices u_a, u_b and w, i.e., the output of TD is NO.

On the other hand, if the output of TD is NO, there exists a triangle in G. From the construction, the triangle is formed with the vertices u_a, u_b and v_i. As $(v_i, u_a) \in E(G)$ implies $x_i = 1$, and $(v_i, u_a) \in E(G)$ implies $y_i = 1$. So, $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$.

Now by Propositions 6.22 and 6.23(ii), we obtain that TD parameterized by vertex cover size K is $(\text{VA}, n/p, p)$-hard even if $\Delta_{av}(G) = \mathcal{O}(1)$, and when $k = 0$. □

In each of the above cases, we can make the reductions work for any k, by adding k vertex disjoint triangles to G. In Theorem 6.20 (III), the vertex cover must be bounded. In the given reduction for Theorem 6.20 (III), the vertex cover of the constructed graph

is at most 2. Note that by the addition of k edge disjoint C_4's, the vertex cover of the constructed graph in the modified reduction is at most $2k + 2$, and is therefore still a parameter independent of the input instance size.

Hence, we are done with the proof of the Theorem 6.20. □

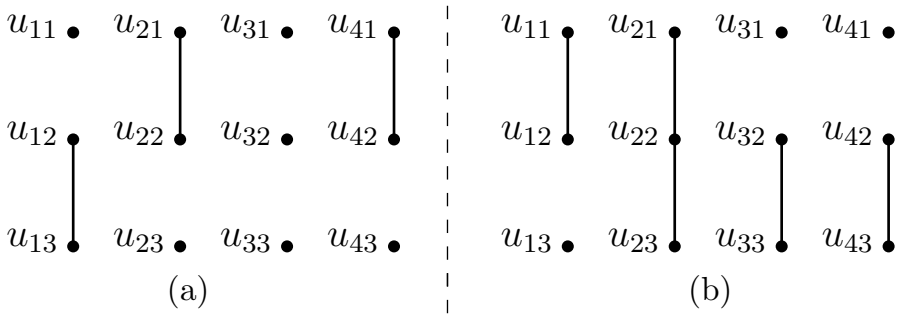

Figure 6.7: Illustration of Proof of Theorem 6.21. Consider $n = 4$. In (a), $\mathbf{x} = 0101$ and $\mathbf{y} = 1000$, that is, $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 1$, and G does not have any induced P_3. In (b), $\mathbf{x} = 1100$ and $\mathbf{y} = 0112$, that is, $\text{DISJ}_n(\mathbf{x}, \mathbf{y}) = 0$, and G contains an induced P_3.

Proof of Theorem 6.21. We give a reduction from DISJ_n to CVD for solution size parameter $k = 0$. Let $\mathcal{A}$ be a one pass streaming algorithm that solves CVD in VA model, such that $\Delta(G) = \mathcal{O}(1)$, and the space used is $o(n)$. Consider a graph G with $3n$ vertices $u_{11}, u_{12}, u_{13}, \ldots, u_{n1}, u_{n2}, u_{n3}$. Let $\mathbf{x}, \mathbf{y}$ be the input of Alice and Bob for DISJ_n. See Figure 6.7 for an illustration.

Alice's input to $\mathcal{A}$: Alice inputs the graph G by exposing the vertices $u_{11}, u_{12}, u_{21}, u_{22}$ $\ldots, u_{n1}, u_{n2}$, sequentially. (i) While exposing u_{i1}, Alice does not give any edge; (ii) while exposing u_{i2}, Alice gives the edge (u_{i2}, u_{i1}) as input to $\mathcal{A}$ if and only if $x_i = 1$.

After the exposure of $u_{11}, u_{12}, u_{21}, u_{22} \ldots, u_{n1}, u_{n2}$ as per the VA model, Alice sends current memory state of $\mathcal{A}$, i.e., the sketch generated by $\mathcal{A}$, to Bob.

Bob's input to $\mathcal{A}$: Bob exposes the vertices $u_{13}, \ldots, u_{n3}$, sequentially. While exposing u_{i3}, Bob gives the edges (u_{i3}, u_{i2}) as an input to $\mathcal{A}$ if and only if $y_i = 1$.

From the construction, note that $\Delta(G) \leq 2$. Observe that, there exists a P_3 in G if and only if there exists an $i \in [n]$ such that $x_i = y_i = 1$. Hence, the output of CVD is NO if and only if $\textsc{Disj}_n(\mathbf{x}, \mathbf{y}) = 0$.

Now by Propositions 6.22 and 6.23(ii), we obtain that CVD is $(\textsc{Va}, n/p, p)$-hard even if $\Delta(G) = \mathcal{O}(1)$, and when $k = 0$.

We can make the reduction work for any k, by adding k vertex disjoint P_3's to G. □

6.6 Conclusion

In this chapter, we initiated the study of parameterized streaming complexity with structural parameters for graph deletion problems. Our study also compares the parameterized streaming complexity of several graph deletion problems in the different streaming models. In future, we wish to investigate why such a classification exists for seemingly similar graph deletion problems, and conduct a systematic study of other graph deletion problems as well.

Chapter 7

Monochromatic Edge Estimation when the Coloring Function also Streams

Contents

7.1 Brief description of the problem and related works

The *chromatic number* $\chi(G)$ of an n-vertex graph $G = (V, E)$ is the minimum number of colors needed to color the vertices of V so that no two adjacent vertices get the same color. The *chromatic number* problem is NP-hard and even hard to approximate within a factor of $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$ [FK98, Zuc07, KP06]. For any connected undirected graph G with maximum degree Δ, $\chi(G)$ is at most $\Delta+1$ [Viz64]. This existential coloring scheme can be made constructive across different models of computation. A seminal result of recent vintage is that the $\Delta + 1$ coloring can be done in the streaming model [ACK19]. Of late, there has been interest in graph coloring problems in the sub-linear regime across a variety of models [AA20a, ACK19, BDH$^+$19, BG18, BCG19]. Keeping with the trend of coloring problems, these works look at assigning colors to vertices. Since the size of the output will be as large as the number of vertices, reseachers study the semi-streaming model [McG14b] for streaming graphs. In the semi-streaming model, $\widetilde{\mathcal{O}}(n)$[1] space is allowed.

In a marked departure from the above works that look at the classical coloring problem, the starting point of our work is (inarguably?) the easiest question one can ask in graph coloring – given a coloring function $f : V \to \{1, \ldots, C\}$ on the vertex set V of a graph $G = (V, E)$, is f a valid coloring, i.e., does every edge have its two endpoints colored with different colors? This is the problem one encounters while proving that the problem of chromatic number belongs to the class NP [GJ79]. CONFLICT-EST, the problem of estimating the number of monochromatic (or, conflicting) edges for a graph G given a coloring function f, remains a simple problem in the RAM model; it even remains simple in the one-pass streaming model if the coloring function f is marked on a *public board*, readable at all times. We show that the problem throws up interesting consequences if the coloring function f on a vertex is revealed only when the vertex is exposed in the stream.

Our work can also be viewed from a contrary perspective as follows. For a streaming graph, if the vertices are assigned colors arbitrarily or randomly on-the-fly while the

[1] $\widetilde{\mathcal{O}}(\cdot)$ hides a polylogarithmic factor.

vertex is exposed, our results can also be used to estimate the number of conflicting edges. These problems also find their use in estimating the number of conflicts in a job schedule and verifying a given job schedule in a streaming setting. This can also be extended to problems in various domains like frequency assignment in wireless mobile networks and register allocation [EHKR09]. As the problem, by its definition, finds an estimate, we can try for space efficient algorithms in the conventional graph streaming models like VERTEX ARRIVAL [CDK19]. We also note in passing that many of the trend setting problems in streaming, like frequency moments, distinct elements, majority, etc. have been simple problems in the ubiquitous RAM model as the coloring problem we solve here.

Recall the definitions of VERTEX ARRIVAL (VA), VERTEX ARRIVAL WITH DEGREE ORACLE (VADEG), VERTEX ARRIVAL IN RANDOM ORDER (VARAND), EDGE ARRIVAL (EA) and ADJACENCY LIST (AL) models for graph streaming discussed in Section 1.1.2. As the conflicts can be checked easily in the EA model in $O(1)$ space, a logarithmic counter is enough to count the number of monochromatic edges. Note that we can count the number of monochromatic edges in a graph by using $\widetilde{\mathcal{O}}(n)$ space in VA, VADEG and VARAND model, by storing each vertex and its color. The AL model works almost the same as the VADEG model. So, we focus on the three models – VA, VADEG and VARAND in this work and show that they have a clear separation in their power vis-a-vis the problem we solve. A crucial takeaway from our work is that the random order assumption on exposure of vertices has huge improvements in space complexity.

7.1.1 Notations, problem definition, results and the ideas

Notations. $G(V(G), E(G))$ denotes a graph where $V(G)$ and $E(G)$ denote the set of vertices and edges of G, respectively; $|V| = n$ and $|E| = m$. We will write only V and E for vertices and edges when the graph is clear from the context. We denote $E_M \subseteq E$ as the set of monochromatic edges. The set of neighbors of a vertex $u \in V(G)$ is denoted by $N_G(u)$ and the degree of a vertex $u \in V(G)$ is denoted by $d_G(u)$. Let $N_G(u) = N_G^-(u) \uplus N_G^+(u)$ where $N_G^-(u)$ and $N_G^+(u)$ denote the set of neighbors of u

that have been exposed already and are yet to be exposed, respectively in the stream. Also, $d_G(u) = d_G^-(u) + d_G^+(u)$ where $d_G^-(u) = \left|N_G^-(u)\right|$ and $d_G^+(u) = \left|N_G^+(u)\right|$. For a monochromatic edge $(u, v) \in E_M$, we refer to u and v as monochromatic neighbors of each other. We define $d_M(u)$ to be the number of monochromatic neighbors of u and hence, the monochromatic degree of u.

Problem definition. Let the vertices of G be colored with a function $f : V(G) \to [C]$, for $C \in \mathbb{N}$. An edge $(u, v) \in E(G)$ is said to be *monochromatic* or *conflicting* with respect to f if $f(u) = f(v)$. A coloring function f is called *valid* if no edge in $E(G)$ is monochromatic with respect to f. For a given parameter $\varepsilon \in (0, 1)$, f is said to be ε-far from being *valid* if at least $\varepsilon \cdot |E(G)|$ edges are monochromatic with respect to f. We study the following problem.

Problem 7.1 (CONFLICT ESTIMATION aka CONFLICT-EST)**.** *A graph $G = (V, E)$ and a coloring function $f : V(G) \to [C]$ are streaming inputs. Given an input parameter $\varepsilon > 0$, the objective is to estimate the number of monochromatic edges in G within a $(1 \pm \varepsilon)$-factor.*

An easier variant of the above problem, CONFLICT-SEP (defined below), was considered by Upasana [Upa20]. Table 7.1 shows the results about CONFLICT-SEP (in different graph models) from the work by Upasana [Upa20].

Problem 7.2 (CONFLICT SEPARATION aka CONFLICT-SEP)**.** *A graph $G = (V, E)$ and a coloring function $f : V(G) \to [C]$ are streaming inputs. Given an input parameter $\varepsilon > 0$, the objective is to distinguish if the coloring function f is valid or is ε-far from being valid.*

Our results Table 7.2 states our results for the CONFLICT-EST problem, the main problem we solve in this work, across different variants of the VA model. The main thrust of our work is on estimating monochromatic edges under random order stream. For random order stream, we present both upper and lower bounds in Sections 7.2 and

Model	VA	VADEG	VARAND
Upper Bound	$\widetilde{\mathcal{O}}\left(\min\{\lvert V\rvert, \frac{\lvert V\rvert^2}{\varepsilon\lvert E\rvert}\}\right)$	$\widetilde{\mathcal{O}}\left(\min\{\lvert V\rvert, \frac{1}{\varepsilon}\}\right)$	$\widetilde{\mathcal{O}}\left(\frac{\lvert V\rvert}{\sqrt{\varepsilon\lvert E\rvert}}\right)$
Lower Bound	$\Omega\left(\frac{\lvert V\rvert}{\sqrt{\lvert E\rvert}}\right)$	$\Omega\left(\frac{1}{\varepsilon}\right)$	—

Table 7.1: This table shows results on CONFLICT-SEP presented in [Upa20]. All of the above upper and lower bounds on space are for one-pass streaming algorithms.

7.3, respectively. There is a gap between the upper and lower bounds in the VARAND model, though we have a strong hunch that our upper bound is tight. Lower bounds for VA and VADEG models have been presented in [Upa20] and [BBMU21]. We show matching upper bounds for the VA and VADEG models in Section 7.4.

Comparison of our results with [Upa20]: As noted earlier, the space complexity of CONFLICT-EST is at least as large as the space complexity of CONFLICT-SEP. We would like to note that both lower and upper bound results of [Upa20] on CONFLICT-SEP in VA and VADEG models can be suitably generalized to that for CONFLICT-EST in VA and VADEG model, respectively, as presented in Table 7.2. In Section 7.4 of this thesis, we present upper bound results on CONFLICT-EST for the VA and VADEG models that match the lower bounds presented in [Upa20] and [BBMU21]. These lower bound results do not form part of this thesis but are presented in Appendix A.1 for easy reference. However, the upper bound result on CONFLICT-SEP in VARAND model by [Upa20] cannot be easily generalized for CONFLICT-EST in VARAND model. Note that the algorithm for CONFLICT-EST in VARAND model is the main upper bound contribution in this chapter. Also, there was no lower bound for CONFLICT-SEP in VARAND model by [Upa20]. But we have a lower bound as presented in Table 7.2.

Model	VA	VADEG	VARAND
Upper Bound	$\widetilde{\mathcal{O}}\left(\min\{\lvert V\rvert, \frac{\lvert V\rvert^2}{T}\}\right)$ (Sec. 7.4, Thm. 7.11)	$\widetilde{\mathcal{O}}\left(\min\{\lvert V\rvert, \frac{\lvert E\rvert}{T}\}\right)$ (Sec. 7.4, Thm. 7.12)	$\widetilde{\mathcal{O}}\left(\frac{\lvert V\rvert}{\sqrt{T}}\right)$ (Sec. 7.2, Thm. 7.3)
Lower Bound	$\Omega\left(\min\{\lvert V\rvert, \frac{\lvert V\rvert^2}{T}\}\right)^{\dagger}$ (see [Upa20] and [BBMU21])	$\Omega\left(\min\{\lvert V\rvert, \frac{\lvert E\rvert}{T}\}\right)^{\dagger}$ (see [Upa20] and [BBMU21])	$\Omega\left(\frac{\lvert V\rvert}{T^2}\right)^{*}$ (Sec. 7.3, Thm. 7.10)

Table 7.2: This table shows our results on CONFLICT-EST on a graph $G(V, E)$ across different VERTEX ARRIVAL models. The lower bound results marked with † were presented in [Upa20] and [BBMU21]. We mention it here to give an overall picture of the bounds. Here, $T > 0$ denotes the promised lower bound on the number of monochromatic edges. The trivial algorithm has space complexity of $\mathcal{O}(n)$ in all the three models, we are also proving $\Omega(n)$ lower bound when $T = 1$ in all the three models. This justifies the assumption of a lower bound T in this work. The result marked with * is for constant pass streaming algorithms; the rest are for one pass streaming algorithm.

The ideas involved. Note that T in Table 7.2 denotes a promised lower bound on the number of monochromatic edges in the graph. The promise T (on the number of monochromatic edges) is a very standard assumption for estimating substructures in the world of graph streaming algorithm [KKP18, KMSS12, KMPV19, MVV16, BC17]. [2]

We now briefly mention the salient ideas involved. For the simpler variant of CONFLICT-EST in VA model, we first check if $|V| \leq T$. If yes, we store all the vertices and their colors in the stream to determine the exact value of the number of monochromatic edges. Otherwise, we sample each pair of vertices $\{u, v\}$ in $\binom{V}{2}$ [3], with probability $\widetilde{\mathcal{O}}(1/T)$ independently [4] before the stream starts. When the stream comes, we compute the number of monochromatic edges from this sample. The details are in Section 7.4. Though the algorithm looks extremely simple, it matches the lower bound result for CONFLICT-EST in VA model, presented in Appendix A.1. The VADEG model with its added power of

[2] Here we have cited a few. However, there are huge amount of relevant literature.
[3] $\binom{V}{2}$ denotes the set of all size 2 subsets of $V(G)$.
[4] Note that we might sample some pairs that are not forming edges in the graph.

a degree oracle, allows us to know $d_G(u)$ for a vertex u and as edges to pre-exposed vertices are revealed, we also know $d^-_G(u)$ and $d^+_G(u)$. This allows us to use sampling to store vertices and to use a technique which we call *sampling into the future* where indices of random neighbors, out of $d^+_G(u)$ neighbors, are selected for future checking. The one-pass algorithm has a dependence on $|E|$ but can work without the knowledge of $|E|$; and this has to be handled in a non-trivial way. The upper bound result, for CONFLICT-EST in VADEG model, is presented in Section 7.4, and it is tight as can be seen from a matching lower bound in Appendix A.1.

The algorithm for CONFLICT-EST in VARAND model is the mainstay of our work and is presented in Section 7.2. We redefine the degree in terms of the number of monochromatic neighbors a vertex has in the randomly sampled set. Here, we estimate the high monochromatic degree and low monochromatic degree vertices separately by sampling a random subset of vertices. While the monochromatic degree for the high degree vertices can be extrapolated from the sample, handling low monochromatic degree vertices individually in the same way does not work. To get around, we group such vertices having similar monochromatic degress and treat them as an entity. We also provide a lower bound for the VARAND model, in Section 7.3, using a reduction from *multi-party set disjointness*; though there is a gap in terms of the exponent in T.

7.1.2 Prior works on graph coloring in semi-streaming model.

Bera and Ghosh [BG18] commenced the study of vertex coloring in the semi-streaming model. They devise a randomized one-pass streaming algorithm that finds a $(1+\varepsilon)\Delta$ vertex coloring in $\widetilde{\mathcal{O}}(n)$ space. They do this in two phases by first randomly partitioning the vertex set into $\mathcal{O}(\frac{\Delta}{\log n})$ subsets where the subgraph induced by each subset has a maximum degree of $\log n$ with high probability. Then, every vertex of the random partitioning is colored independently and uniformly at random using a $\mathcal{O}(\frac{\Delta}{\log n})$ sized color palette. Assadi et al. [ACK19] find a proper vertex coloring using $\Delta+1$ colors via various classes of sublinear algorithms. Their state of the art contributions can be attributed to a key result called the *palette-sparsification theorem* which states that for an

n-vertex graph with maximum degree Δ, if $\mathcal{O}(\log n)$ colors are sampled independently and uniformly at random for each vertex from a list of $\Delta + 1$ colors, then with a high probability, a proper $\Delta + 1$ coloring exists for the graph. They design a randomized one-pass dynamic streaming algorithm for the $\Delta + 1$ coloring using $\widetilde{\mathcal{O}}(n)$ space. The algorithm takes post-processing $\widetilde{\mathcal{O}}(n\sqrt{\Delta})$ time and assumes a prior knowledge of Δ. Alon and Assadi [AA20b] improve the palette sparsification result of [ACK19]. They consider situations where the number of colors available is both more than and less than $\Delta + 1$ colors. They show that sampling $\mathcal{O}_\varepsilon(\sqrt{\log n})$ colors [5] per vertex is sufficient and necessary for a $(1 + \varepsilon)\Delta$ coloring. Bera et al. [BCG19] give a new graph coloring algorithm in the semi-streaming model where the number of colors used is parameterized by the degeneracy κ. The key idea is a *low degeneracy partition*, also employed in [BG18]. The numbers of colors used to properly color the graph is $\kappa + o(\kappa)$ and post-processing time of the algorithm is improved to $\widetilde{\mathcal{O}}(n)$, without any prior knowledge about κ. Behnezhad et al. [BDH$^+$19] were the first to give one-pass W-streaming algorithms (streaming algorithms where outputs are produced in a streaming fashion as opposed to outputs given finally at the end) for edge coloring both when the edges arrive in a random order or in an adversarial fashion.

7.2 CONFLICT-EST in VARAND model

In this Section, we mainly show that the power of randomness can be used to design a better solution for the CONFLICT-EST problem in the VARAND model. The CONFLICT-EST problem is the main highlight of our work. We feel that the crucial use of randomness in the input that is used to estimate a substructure (here, monochromatic edges) in a graph, will be of independent interest.

In this variant, we are given an $\varepsilon \in (0, 1)$ and a promised lower bound T on $|E_M|$, the number of monochromatic edges in G, as input and our objective is to determine a $(1 \pm \varepsilon)$-approximation to $|E_M|$.

[5] The constant in $\mathcal{O}(\cdot)$ depends on ε.

Theorem 7.3. *Given any graph $G = (V, E)$ and a coloring function $f : V(G) \to [C]$ as input in the stream, the* CONFLICT-EST *problem in the* VARAND *model can be solved with high probability in $\widetilde{\mathcal{O}}\left(\frac{|V|}{\sqrt{T}}\right)$ space, where T is a lower bound on the number of monochromatic edges in the graph.*

We discuss the algorithm and its proof idea in Section 7.2.1 and the formal proof is in Section 7.2.2.

7.2.1 The proof idea of Theorem 7.3 for CONFLICT-EST in VARAND model

A random sample comes for free – pick the first few vertices: Let $v_1, \ldots, v_n$ be the random ordering in which the vertices of V are revealed. Let R be a random subset of $\Gamma = \widetilde{\Theta}\left(\frac{n}{\sqrt{T}}\right)$ vertices of G sampled without replacement [6]. As we are dealing with a random order stream, consider the first Γ vertices in the stream; they can be treated as R, the random sample. We start by storing all the vertices in R as well as their colors. Observe that if the monochromatic degree of any vertex v_i is *large* (say roughly more than $\sqrt{T}$), then it can be well approximated by looking at the number of monochromatic neighbors that v_i has in R. As a vertex v_i streams past, there is no way we can figure out its monochromatic degree, unless we store its monochromatic neighbors that appear before it in the stream; if we could, we were done. Our only savior is the stored random subset R.

Classifying the vertices of the random sample R based on its monochromatic degree: Our algorithm proceeds by figuring out the influence of the color of v_i on the monochromatic degrees of vertices in R. To estimate this, let κ_{v_i} denote the number of monochromatic neighbors that v_i has in R. We set a threshold $\tau = \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}$, where $t = \lceil \log_{1+\frac{\varepsilon}{10}} n \rceil$. The significance of t will be clear from the discussion below. Any vertex v_i will be classified as a high-m_R or low-m_R degree vertex depending on its monochromatic degree within R, i.e., if $\kappa_{v_i} \geq \tau$, then v_i is a high-m_R vertex, else it

[6] $\widetilde{\Theta}(\cdot)$ hides a polynomial factor of $\log n$ and $\frac{1}{\varepsilon}$ in the upper bound.

is a low-m$_R$ vertex, respectively. (We use the subscripts m$_R$ to stress the fact that the monochromatic degrees are induced by the set R.) Let H and L be the partition of V into the set of high-m$_R$ and low-m$_R$ degree vertices in G. Let H_R and L_R denote the set of high-m$_R$ and low-m$_R$ degree vertices in R. Notice that, because of the definition of high-m$_R$ and low-m$_R$ degree vertices, not only the sets H_R, L_R are subsets of R, but they are determined by the vertices of R only.

Let m_h and m_ℓ denote the sum of the monochromatic degrees of all the high-m$_R$ degree vertices and low-m$_R$ degree vertices in G, respectively. So, $m_h = \sum_{v \in H} d_M(v)$ and $m_\ell = \sum_{v \in L} d_M(v)$. Note that

$$\widehat{m} = |E_M| = \frac{1}{2} \sum_{v \in V} d_M(v) = \frac{1}{2} \left(m_h + m_\ell \right). \tag{7.1}$$

We will describe how to approximate m_h and m_ℓ separately. The formal algorithm is described in Algorithm 7.1 as RANDOM-ORDER-EST(ε, T) that basically executes steps to approximate m_h and m_ℓ in parallel.

To approximate m_h, the random sample R comes to rescue: We can find $\widehat{m_h}$, that is, a $\left(1 \pm \frac{\varepsilon}{10}\right)$ approximation of m_h as described below. For each vertex $v_i \in R$ and each monochromatic edge (u, v_i), $u \in R$, we see in the stream, we increase the value of κ_u for u and κ_{v_i} for v_i. After all the vertices in R are revealed, we can determine H_R by checking whether $\kappa_{v_i} \geq \tau$ for each $v_i \in R$. For each vertex $v_i \in H_R$, we set its approximate monochromatic degree $\widehat{d_{v_i}}$ to be $\frac{n}{|R|}\kappa_{v_i}$. We initialize the estimated sum of the monochromatic degree of high degree vertices as $\widehat{m_h} = \sum_{v_i \in H_R} \widehat{d_{v_i}}$. For each vertex $v_i \notin R$ in the stream, we can determine κ_{v_i}, as we have stored all the vertices in R along with their colors, and hence we can also determine whether v_i is a high-m$_R$ degree vertex in G. If $v_i \notin R$ is a high-m$_R$ degree vertex, we determine $\widehat{d_{v_i}} = \frac{n}{|R|}\kappa_{v_i}$ and update $\widehat{m_h}$ by $\widehat{m_h} + \widehat{d_{v_i}}$. Observe that, at the end, $\widehat{m_h}$ is $\sum_{v_i \in H} \widehat{d_{v_i}}$. Recall that H is the set of all high-m$_R$ degree vertices in G. For each $v_i \in H$, we will show, as in Claim 7.5, that $\widehat{d_{v_i}}$ is a $\left(1 \pm \frac{\varepsilon}{10}\right)$-approximation to $d_M(v_i)$ with high probability. This implies that

Algorithm 7.1: RANDOM-ORDER-EST(ε, T): CONFLICT-EST in VARAND model

Input: $G = (V, E)$ and a coloring function f on V in the VARAND model, parameters T and ε.
Output: $\widehat{m}$, that is, a $(1 \pm \varepsilon)$ approximation to $|E_M|$.

$\Gamma = \widetilde{\Theta}\left(\frac{n}{\sqrt{T}}\right)$; $v_1, \ldots, v_n$ be the random ordering in which vertices are revealed and $R = \{v_1, \ldots, v_\Gamma\}$;

$\kappa_{v_i}, i \in [n]$, denotes the number of monochromatic neighbors of v_i in R,

$\widehat{d_{v_i}}, i \in [n]$, denotes the (estimated) monochromatic neighbors of vertices in G. H denotes the set of *high* degree vertex in R, i.e., $H = \{v_i : \kappa_{v_i} \geq \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}\}$ and $L = V \setminus H$; $L_R = L \cap R$ and $H_R = H \cap R$;

The vertices in L are partitioned into t buckets as follows:

$B_j = \{v_i \in L : \left(1 + \frac{\varepsilon}{10}\right)^{j-1} \leq d_M(v_i) < \left(1 + \frac{\varepsilon}{10}\right)^{j}\}$, where $j \in [t]$.

Set $t = \lceil \log_{1+\frac{\varepsilon}{10}} n \rceil$. If $T < 63t^2$, then store all the vertices in G along with their colors. At the end, report the exact value of $|E_M|$. Otherwise, we proceed through via three building blocks described below and marked as (1),(2), (3) and (4).

(1) Processing the vertices in R, the first Γ vertices, in the stream:

for *(each vertex $v_i \in R$ exposed in the stream)* **do**

    Store v_i as well as its color $f(v_i)$.

    For each edge $(v_{i'}, v_i)$ that arrives in the stream, increase the values of $\kappa_{v_{i'}}$ and κ_{v_i}.

(2)Computation of some parameters based on vertices in R and their colors:

for *(each $v_i \in R$ with $\kappa_{v_i} \geq \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}$)* **do**

    Add v_i to H_R, and set $\widehat{d_{v_i}} = \frac{n}{|R|}\kappa_{v_i}$.

$\widehat{m_h} = \sum_{v_i \in H} \widehat{d_{v_i}}$. Let $L_R = R \setminus H_R$.

for *(each $v_i \in L_R$)* **do**

    Set $\widehat{d_{v_i}} = \kappa_{v_i}$.

(3)Processing the vertices in $V(G) \setminus R$ in the stream:

for *(each vertex $v_i \notin R$ exposed in the stream)* **do**

    Determine the value of κ_{v_i}.

    If $\kappa_{v_i} \geq \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}$, find $\widehat{d_{v_i}} = \frac{n}{|R|}\kappa_{v_i}$ and add $\widehat{d_{v_i}}$ to the current $\widehat{m_h}$.

    Also, for each $v_{i'} \in L_R$, increase the value of $\widehat{d_{v_{i'}}}$ if $(v_{i'}, v_i)$ is an edge.

(4)Post processing, after the stream ends, to return the output:

From the values of $\widehat{d_{v_i}}$ for all $v_i \in L_R$, determine the buckets for each vertex in L_R. Also, for each $j \in [t]$, find $|A_j| = |L_R \cap B_j|$. Then determine

$$\widehat{m_\ell} = \frac{n}{|R|} \sum_{j \in [t]} |A_j| \left(1 + \frac{\varepsilon}{10}\right)^j.$$

Report $\widehat{m} = \frac{\widehat{m_h} + \widehat{m_\ell}}{2}$ as the final OUTPUT. // See Equation 7.1.

$$\left(1-\frac{\varepsilon}{10}\right)m_h \leq \widehat{m_h} \leq \left(1+\frac{\varepsilon}{10}\right)m_h \tag{7.2}$$

To approximate m_ℓ, group the vertices in L based on similar monochromatic degree: Recall that $m_\ell = \sum_{v_i \in L} d_M(v_i)$. Unlike the high-m$_R$ degree vertices, it is not possible to approximate the monochromatic degree of $v_i \in L$ from κ_{v_i}. To cope up with this problem, we partition the vertices of L into t buckets $B_1, \ldots, B_t$ such that all the vertices present in a bucket have *similar* monochromatic degrees, where $t = \lceil \log_{1+\frac{\varepsilon}{10}} n \rceil$. The bucket B_j is defined as follows: $B_j = \{v_i \in L : \left(1+\frac{\varepsilon}{10}\right)^{j-1} \leq d_M(v_i) < \left(1+\frac{\varepsilon}{10}\right)^{j}\}$.

Note that our algorithm will not find the buckets explicitly. It will be used for the analysis only. Observe that $\sum_{j\in[t]} |B_j| \left(1+\frac{\varepsilon}{10}\right)^{j-1} \leq m_\ell < \sum_{j\in[t]} |B_j| \left(1+\frac{\varepsilon}{10}\right)^{j}$. We can surely approximate m_ℓ by approximating $|B_j|$s suitably. We estimate $|B_j|$s as follows. After the stream of the vertices in R has gone past, we have the set of low-m$_R$ degree vertices L_R in R and $\widehat{d_{v_i}} = \kappa_{v_i}$ for each $v_i \in L_R$. For each $v_i \notin R$ in the stream, we determine the monochromatic neighbors of v_i in L_R. It is possible as we have stored all the vertices in R and their colors. For each monochromatic neighbor $v_{i'} \in L_R$ of v_i, we increase the value of $\widehat{d}_{v_{i'}}$ of $v_{i'}$. Observe that, at the end of the stream, $\widehat{d_{v_{i'}}} = d_M(v_{i'})$ for each $v_{i'} \in L_R$, i.e., we can accurately estimate the monochromatic degree of each $v_{i'} \in L_R$. So, we can determine the bucket where each vertex in L_R belongs. Let A_j $(= L_R \cap B_j)$ be the bucket B_j projected onto L_R in the random sample; note that as $B_j \subseteq L$ and $L_R = L \cap R$, $A_j = R \cap B_j$ also. We determine $\widehat{m_\ell} = \frac{n}{|R|} \sum_{j\in[t]} |A_j| \left(1+\frac{\varepsilon}{10}\right)^{j}$. We can show that $\frac{n}{|R|}|A_j|$ is a $\left(1+\frac{\varepsilon}{10}\right)$-approximation of $|B_j|$, with high probability, if $|B_j| \geq \frac{\sqrt{\varepsilon T}}{10t}$. Also, we can show that, if $|B_j| < \frac{\sqrt{\varepsilon T}}{10t}$, then $|A_j| \leq \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}$ with high probability. Now using the fact that we consider bucketing of only low-m$_R$ degree vertices (L_R), we can show that

$$\left(1-\frac{\varepsilon}{10}\right)\left(m_\ell - \frac{\varepsilon T}{63t}\right) \leq \widehat{m_\ell} \leq \left(1+\frac{\varepsilon}{10}\right)^2\left(m_\ell + \frac{\varepsilon T}{56t}\right). \tag{7.3}$$

Note that $\varepsilon \in (0,1)$ and $t = \lceil \log_{1+\frac{\varepsilon}{10}} n \rceil$. Assuming $T \geq 63t^2$, Equations 7.2 and 7.3

imply that $\widehat{m} = \frac{1}{2}(\widehat{m_h} + \widehat{m_\ell})$ is a $(1 \pm \varepsilon)$-approximation to $|E_M|$. If $T < 63t^2$, then note that $n = \widetilde{\mathcal{O}}\left(\frac{n}{\sqrt{T}}\right)$. So, in that case, we store all the vertices along with their colors and compute the exact value of $|E_M|$.

7.2.2 Proof of correctness

The correctness of the algorithm follows trivially if $T < 63t^2$. So, let us assume that $T \geq 63t^2$. In the VARAND model, we consider the first $\widetilde{\Theta}\left(\frac{n}{\sqrt{T}}\right)$ vertices as the random sample R without replacement. Using the Chernoff bound for sampling without replacement (See Lemma 2.6 in Section 2.1), we can have the following lemma, which will be useful for the correctness proof of Algorithm 7.1 (RANDOM-ORDER-EST(ε, T)) in case of $T \geq 63t^2$.

Lemma 7.4. *(i) For each $j \in [t]$ with $|B_j| \geq \frac{\sqrt{\varepsilon T}}{10t}$, $\mathbb{P}\left(\left||B_j \cap R| - \frac{|R||B_j|}{n}\right| \geq \frac{\varepsilon}{10}\frac{|R||B_j|}{n}\right) \leq \frac{1}{n^{10}}$.*

(ii) For each $j \in [t]$ with $|B_j| < \frac{\sqrt{\varepsilon T}}{10t}$, $\mathbb{P}\left(|B_j \cap R| \geq \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}\right) \leq \frac{1}{n^{10}}$.

(iii) For each vertex v_i with $d_M(v_i) \geq \frac{\sqrt{\varepsilon T}}{10t}$, $\mathbb{P}\left(\left|\kappa_{v_i} - \frac{|R|d_M(v_i)}{n}\right| \geq \frac{\varepsilon}{10}\frac{|R|d_M(v_i)}{n}\right) \leq \frac{1}{n^{10}}$.

(iv) For each vertex v_i with $d_M(v_i) < \frac{\sqrt{\varepsilon T}}{10t}$, $\mathbb{P}\left(\kappa_{v_i} \geq \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}\right) \leq \frac{1}{n^{10}}$.

Proof. Let us take $N = n, r = |R| = \Gamma = \widetilde{\Theta}\left(\frac{n}{\sqrt{T}}\right)$, $I = \{v_1, \ldots, v_n\}$ in Lemma 2.6.

(i) Setting $J = B_j$ and $\delta = \frac{\varepsilon}{10}$ in Lemma 2.6 (i) and (ii), we have

$$\mathbb{P}\left(\left||B_j \cap R| - \frac{|R|\,|B_j|}{n}\right| \geq \frac{\varepsilon}{10}\frac{|R|\,|B_j|}{n}\right) \leq 2\exp\left(-\frac{(\varepsilon/10)^2\,|B_j|\,\Gamma}{3n}\right) \leq \frac{1}{n^{10}}.$$

The last inequality holds as $|B_j| \geq \frac{\sqrt{\varepsilon T}}{10t}$, $t = \lceil \log_{1+\frac{\varepsilon}{10}} n \rceil = \Theta\left(\frac{\log n}{\varepsilon}\right)$ and $\Gamma = \widetilde{\Theta}\left(\frac{n}{\sqrt{T}}\right)$.

(ii) Set $J = B_j$, $k = \frac{\sqrt{\varepsilon T}}{10t}$, $\delta = \frac{1}{4}$ in Lemma 2.6 (iii). As $|B_j| \leq \frac{\sqrt{\varepsilon T}}{10t}$, $|J| \leq k$. Hence,

$$\mathbb{P}\left(|B_j \cap R| \geq \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}\right) \leq \exp\left(-\frac{(1/4)^2(\sqrt{\varepsilon T}/10t)\Gamma}{3n}\right) \leq \frac{1}{n^{10}}.$$

(iii) Setting J as the set of monochromatic neighbors of v_i in R and $\delta = \frac{\varepsilon}{10}$ in Lemma 2.6 (i) and (ii), we get

$$\mathbb{P}\left(\left|\kappa_{v_i} - \frac{|R|\, d_M(v_i)}{n}\right| \geq \frac{\varepsilon}{10}\frac{|R|\, d_M(v_i)}{n}\right) \leq \exp\left(-\frac{(\varepsilon/10)^2|J|\Gamma}{3n}\right) \leq \frac{1}{n^{10}}.$$

The last inequality holds as $|J| = d_M(v_i) \geq \frac{\sqrt{\varepsilon T}}{10t}$, $t = \lceil \log_{1+\frac{\varepsilon}{10}} n \rceil = \Theta\left(\frac{\log n}{\varepsilon}\right)$ and $\Gamma = \widetilde{\Theta}\left(\frac{n}{\sqrt{T}}\right)$.

(iv) Set J as the set of monochromatic neighbors of v_i in R, $k = \frac{\sqrt{\varepsilon T}}{10t}$, $\delta = \frac{1}{4}$ in Lemma 2.6 (iii). Note that $|J| = d_M(v_i) \leq \frac{\sqrt{\varepsilon T}}{10t} = k$. Hence,

$$\mathbb{P}\left(\kappa_{v_i} \geq \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}\right) \leq \exp\left(-\frac{(1/4)^2(\sqrt{\varepsilon T}/10t)\Gamma}{3n}\right) \leq \frac{1}{n^{10}}.$$

□

The correctness proof of the algorithm is divided into the following two claims.

Claim 7.5. $\left(1 - \frac{\varepsilon}{10}\right) m_h \leq \widehat{m_h} \leq \left(1 + \frac{\varepsilon}{10}\right) m_h$ *with probability at least* $1 - \frac{1}{n^9}$.

Claim 7.6. $\left(1 - \frac{\varepsilon}{10}\right)\left(m_\ell - \frac{\varepsilon T}{63t}\right) \leq \widehat{m_\ell} \leq \left(1 + \frac{\varepsilon}{10}\right)^2\left(m_\ell + \frac{\varepsilon T}{56t}\right)$ *with probability at least* $1 - \frac{1}{n^7}$.

Assuming the above two claims hold and taking $\varepsilon \in (0, 1)$, $t = \lceil \log_{1+\frac{\varepsilon}{10}} n \rceil$ and $T \geq 63t^2$, observe that $\widehat{m} = \frac{1}{2}(\widehat{m_h} + \widehat{m_\ell})$ is a $(1 \pm \varepsilon)$ approximation of $|E_M| = m_h + m_\ell$ with high probability. Thus, it remains to prove Claims 7.5 and 7.6.

Proof of Claim 7.5. Note that $m_h = \sum_{v_i : \kappa_{v_i} \geq \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}} d_M(v_i)$ and $\widehat{m_h} = \sum_{v_i : \kappa_{v_i} \geq \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}} \widehat{d_{v_i}}$.

From Lemma 7.4 (iv) and (iii), $\kappa_{v_i} \geq \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}$ implies that $\widehat{d_{v_i}}$ is an $\left(1 \pm \frac{\varepsilon}{10}\right)$ approximation to $d_M(v_i)$ with probability at least $1 - \frac{2}{n^{10}}$. Hence, we have $\left(1 - \frac{\varepsilon}{10}\right) m_h \leq \widehat{m_h} \leq \left(1 + \frac{\varepsilon}{10}\right) m_h$ with probability at least $1 - \frac{1}{n^9}$. □

Proof of Claim 7.6. Note that

$$m_\ell = \sum_{v_i \in L} d_M(v_i) = \sum_{v_i : \kappa_{v_i} < \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}} d_M(v_i)$$

and

$$\widehat{m_\ell} = \frac{n}{|R|} \sum_{j \in [t]} |A_j| \left(1 + \frac{\varepsilon}{10}\right)^j$$

Recall that the vertices in L are partitioned into t buckets as follows: $B_j = \{v_i \in L : \left(1 + \frac{\varepsilon}{10}\right)^{j-1} \leq d_M(v_i) < \left(1 + \frac{\varepsilon}{10}\right)^j\}$, where $j \in [t]$. By Lemma 7.4 (iv), $\kappa_{v_i} < \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}$ implies that $d_M(v_i) \leq \frac{\sqrt{\varepsilon T}}{7t}$ with probability $1 - \frac{1}{n^{10}}$. So, we have the following observation.

Observation 7.7. Let $j \in [t]$ be such that $|A_j| \neq 0$ ($|B_j| \neq 0$). Then, with probability at least $1 - \frac{1}{n^{10}}$, the monochromatic degree of each vertex in A_j as well as B_j is at most $\frac{\sqrt{\varepsilon T}}{7t}$, that is, $\left(1 + \frac{\varepsilon}{10}\right)^j \leq \frac{\sqrt{\varepsilon T}}{7t}$.

To upper and lower bound $\widehat{m_\ell}$ in terms of m_ℓ, we upper and lower bound m_ℓ in terms of $|B_j|$'s as follows; for the upper bound, we break the sum into two parts corresponding to large and small sized buckets:

$$\sum_{j \in [t]} |B_j| \left(1 + \frac{\varepsilon}{10}\right)^{j-1} \leq m_\ell < \sum_{j \in [t]} |B_j| \left(1 + \frac{\varepsilon}{10}\right)^j$$

$$\sum_{j \in [t]} |B_j| \left(1 + \frac{\varepsilon}{10}\right)^{j-1} \leq m_\ell < \sum_{j \in [t] : |B_j| \geq \frac{\sqrt{\varepsilon T}}{9t}} |B_j| \left(1 + \frac{\varepsilon}{10}\right)^j + \sum_{j \in [t] : |B_j| < \frac{\sqrt{\varepsilon T}}{9t}} |B_j| \left(1 + \frac{\varepsilon}{10}\right)^j$$

By Observation 7.7, we bound m_ℓ in terms of $|B_j|$'s with probability $1 - \frac{1}{n^9}$.

$$\sum_{j\in[t]} |B_j| \left(1+\frac{\varepsilon}{10}\right)^{j-1} \leq m_\ell < \sum_{j\in[t]:|B_j|\geq\frac{\sqrt{\varepsilon T}}{9t}} |B_j| \left(1+\frac{\varepsilon}{10}\right)^{j} + t \cdot \frac{\sqrt{\varepsilon T}}{9t}\frac{\sqrt{\varepsilon T}}{7t}$$

This implies the following Observation:

Observation 7.8. $\sum_{j\in[t]} |B_j| \left(1+\frac{\varepsilon}{10}\right)^{j-1} \leq m_\ell < \sum_{j\in[t]:|B_j|\geq\frac{\sqrt{\varepsilon T}}{9t}} |B_j| \left(1+\frac{\varepsilon}{10}\right)^{j} + \frac{\varepsilon T}{63t}$ holds with probability at least $1 - \frac{1}{n^9}$.

Now, we have all the ingredients to show that $\widehat{m_\ell}$ is a $(1 \pm \varepsilon)$ approximation of m_ℓ. To get to $\widehat{m_\ell}$, we need to focus on low-m$_R$ vertices of R, i.e., A_j's. Breaking $\widehat{m_\ell} = \frac{n}{|R|} \sum_{j\in[t]} |A_j| \left(1+\frac{\varepsilon}{10}\right)^{j}$ depending on small and large values of $|A_j|$'s (recall $A_j = L_R \cap B_j = R \cap B_j$), we have

$$\widehat{m_\ell} = \frac{n}{|R|} \left[\sum_{j\in[t]:|A_j|\geq\frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}} |A_j| \left(1+\frac{\varepsilon}{10}\right)^{j} + \sum_{j\in[t]:|A_j|<\frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}} |A_j| \left(1+\frac{\varepsilon}{10}\right)^{j} \right] \quad (7.4)$$

Note that $A_j = B_j \cap R$. By Lemma 7.4 (ii), $|A_j| \geq \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}$ implies $|B_j| \geq \frac{\sqrt{\varepsilon T}}{10t}$ with probability at least $1 - \frac{1}{n^{10}}$. Also, applying Lemma 7.4 (i), $|B_j| \geq \frac{\sqrt{\varepsilon T}}{10t}$ implies $|A_j|$ is an $\left(1 \pm \frac{\varepsilon}{10}\right)$-approximation to $\frac{|R||B_j|}{n}$ with probability at least $1 - \frac{1}{n^{10}}$. So, we have the following observation.

Observation 7.9. Let $j \in [t]$ be such that $|A_j| \geq \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}$. Then $|A_j|$ is an $\left(1 \pm \frac{\varepsilon}{10}\right)$-approximation to $\frac{|R||B_j|}{n}$ with probability at least $1 - \frac{2}{n^{10}}$, that is, $\frac{n}{|R|}|A_j|$ is an $\left(1 \pm \frac{\varepsilon}{10}\right)$-approximation to $|B_j|$ with probability at least $1 - \frac{2}{n^{10}}$

By the above observation along with Equation 7.4, we have the following upper bound on $\widehat{m_\ell}$ with probability at least $1 - \frac{1}{n^9}$.

$$\begin{aligned}\widehat{m_\ell} &\leq \sum_{j\in[t]:|A_j|\geq\frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}} \left(1+\frac{\varepsilon}{10}\right)|B_j|\left(1+\frac{\varepsilon}{10}\right)^j + \sum_{j\in[t]:|A_j|<\frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}} \frac{n}{|R|}|A_j|\left(1+\frac{\varepsilon}{10}\right)^j \\ &\leq \left(1+\frac{\varepsilon}{10}\right)^2\left[\sum_{j\in[t]:|A_j|\geq\frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}} |B_j|\left(1+\frac{\varepsilon}{10}\right)^{j-1} + \sum_{j\in[t]:|A_j|<\frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}} \frac{\sqrt{\varepsilon T}}{8t}\left(1+\frac{\varepsilon}{10}\right)^{j-2}\right]\end{aligned}$$

Now by Observations 7.8 and 7.7, we have the following with probability at least $1-\frac{1}{n^8}$.

$$\begin{aligned}\widehat{m_\ell} &\leq \left(1+\frac{\varepsilon}{10}\right)^2\left(m_\ell + t\cdot\frac{\sqrt{\varepsilon T}}{8t}\frac{\sqrt{\varepsilon T}}{7t}\right) \\ &= \left(1+\frac{\varepsilon}{10}\right)^2\left(m_\ell+\frac{\varepsilon T}{56t}\right)\end{aligned}$$

Now, we will lower bound $\widehat{m_\ell}$. From Equation 7.4, we have

$$\widehat{m_\ell} \geq \frac{n}{|R|}\sum_{j\in[t]:|A_j|\geq\frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}} |A_j|\left(1+\frac{\varepsilon}{10}\right)^j$$

By Observation 7.9, $|A_j| \geq \frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}$ implies $\frac{n}{|R|}|A_j|$ is an $\left(1\pm\frac{\varepsilon}{10}\right)$-approximation to $|B_j|$ with probability at least $1-\frac{2}{n^{10}}$. So, the following lower bound on $\widehat{m_\ell}$ holds with probability at least $1-\frac{1}{n^9}$.

$$\widehat{m_\ell} \geq \left(1-\frac{\varepsilon}{10}\right)\sum_{j\in[t]:|A_j|\geq\frac{|R|}{n}\frac{\sqrt{\varepsilon T}}{8t}} |B_j|\left(1+\frac{\varepsilon}{10}\right)^j$$

By Lemma 7.4 (i), if $|B_j| \geq \frac{\sqrt{\varepsilon T}}{9t}$, then $|A_j| \geq \frac{\sqrt{\varepsilon T}}{8t}$ with probability at least $1-\frac{1}{n^{10}}$. Hence, we have the following lower bound on m_ℓ with probability at least $1-\frac{1}{n^8}$.

$$\widehat{m_\ell} \geq \left(1-\frac{\varepsilon}{10}\right)\sum_{j\in[t]:|B_j|\geq\frac{\sqrt{\varepsilon T}}{9t}} |B_j|\left(1+\frac{\varepsilon}{10}\right)^j$$

Now by Observation 7.8, we have the following with probability at least $1 - \frac{1}{n^7}$.

$$\widehat{m_\ell} \;\geq\; \left(1 - \frac{\varepsilon}{10}\right)\left(m_\ell - \frac{\varepsilon T}{63t}\right).$$

□

7.3 Lower bound for CONFLICT-EST in VARAND model

In this Section, we show a lower bound of $\Omega\left(\frac{n}{T^2}\right)$ for CONFLICT-EST in VERTEX ARRIVAL IN RANDOM ORDER via a reduction from a variation of MULTIPARTY SET DISJOINTNESS problem called DISJOINTNESS$_R(t, n, p)$, played among p players: Consider a matrix of order $t \times n$ having t (rows) vectors $M_1, \ldots, M_t \in \{0,1\}^n$ such that each entry of matrix M is given to one of the p players chosen uniformly at random. The objective is to determine whether there exists a column where all the entries are 1s. If $t \geq 2$ and $p = \Omega(t^2)$, Chakrabarti et al. showed that any randomized protocol requires $\Omega\left(\frac{n}{t}\right)$ bits of communication [CCM16]. They showed that the lower bound holds under a promise called the UNIQUE INTERSECTION PROMISE which states that there exists at most a single column where all the entries are 1s and every other column of the matrix has Hamming weight either 0 or 1. Moreover, the lower bound holds even if all the p players know the random partition of the entries of matrix M.

Theorem 7.10. *Let $n, T \in \mathbb{N}$ be such that $4 \leq T \leq \binom{n}{2}$. Any constant pass streaming algorithm that takes the vertices and edges of a graph $G(V, E)$ (with $|V| = \Theta(n)$ and $|E| = \Theta(m)$) and a coloring function $f : V \to [C]$ in the* VARAND *model, and determines whether the number of monochromatic edges in G is 0 or $\Omega(T)$ with probability $2/3$, requires $\Omega\left(\frac{n}{T^2}\right)$ bits of space.*

Proof. Without loss of generality, assume that $\sqrt{T} \in \mathbb{N}$. Consider the DISJOINTNESS$_R$ $\left(\sqrt{T}, \frac{n}{\sqrt{T}}, p\right)$ problem with UNIQUE INTERSECTION PROMISE when all of the p players know the random partition of the entries of the relevant matrix M. Note that M is of order $[\sqrt{T}] \times \left[\frac{n}{\sqrt{T}}\right]$ and $p = AT$ for some suitable constant $A \in \mathbb{N}$. Also, consider a

graph G, with $V(G) = \{v_{ij} : i \in \left[\sqrt{T}\right], j \in \left[\frac{n}{\sqrt{T}}\right]\}$, having $\frac{n}{\sqrt{T}}$ vertex disjoint cliques such that $\{v_{1j}, \ldots, v_{\sqrt{T}j}\}$ forms a clique for each $j \in [n]$, i.e., a column of M forms a clique. Also, notice that each clique has $\Theta(T)$ edges. Let us assume that there is an r-pass streaming algorithm $\mathcal{S}$, with space complexity s bits, that solves CONFLICT-EST for the above graph G in the VARAND model. Now, we give a protocol $\mathcal{A}$ for DISJOINTNESS$_R\left(\sqrt{T}, \frac{n}{\sqrt{T}}, p\right)$ with communication cost $O(rsp)$. Using the fact that the lower bound of DISJOINTNESS $\left(\sqrt{T}, \frac{n}{\sqrt{T}}, p\right)$ is $\Omega\left(\frac{n/\sqrt{T}}{\sqrt{T}}\right)$ along with the fact that $p = AT$ and r is a constant, we get $s = \Omega\left(\frac{n}{T^2}\right)$.

Protocol $\mathcal{A}$ for DISJOINTNESS$_R\left(\sqrt{T}, \frac{n}{\sqrt{T}}, p\right)$**:** Let $P_1, \ldots, P_p$ denote the set of p players. For $k \in [p]$, $V_k = \{v_{ij} : M_{ij}$ is with $P_k\}$, where M_{ij} denotes the element present in the i-th row and j-th column of matrix M. Note that there is a one-to-one correspondence between the entries of M and the vertices in $V(G)$. Furthermore, there is a one-to-one correspondence between the columns of matrix M and the cliques in graph G. We assume that all the p players know the graph structure completely as well as both the one-to-one correspondences. The protocol proceeds as follows: for each $k \in [p]$, player P_k determines a random permutation π_k of the vertices in V_k. Also, for each $k \in [p]$, player P_k determines the colors of the vertices in V_k by the following rule: if $M_{ij} = 1$, then color vertex v_{ij} with color C_*. Otherwise, for $M_{ij} = 0$, color vertex v_{ij} with color C_i. Player P_1 initiates the streaming algorithm and it goes over r-rounds.

Rounds 1 **to** $r - 1$**:** For $k \in [p]$, each player resumes the streaming algorithm by exposing the vertices in V_k, along with their colors, in the order dictated by π_k. Also, P_k adds the respective edges to previously exposed vertices when the current vertex is exposed to satisfy the basic requirement of VA model. This is possible because all players know the graph G and the random partition of the entries of matrix M among p players. After exposing all the vertices in V_k, as described, P_k sends the current memory state to player P_{k+1}. Assume that $P_1 = P_{p+1}$.

Round r**:** All the players behave similarly as in the previous rounds, except that, the player P_p does not send the current memory state to P_1. Rather, P_p decides

whether there is a column in M with all 1s if the streaming algorithm $\mathcal{S}$ decides that there are $\Omega(T)$ monochromatic edges in G. Otherwise, if $\mathcal{S}$ decides that there is no monochromatic edge in G, then P_p decides that all the columns of M have weight either 0 or 1. Then P_p sends the output to all other players.

The vertices of graph G are indeed exposed randomly to the streaming algorithm. It is because the entries of matrix M are randomly partitioned among the players and each player also generates a random permutation of the vertices corresponding to the entries of matrix M available to them. From the description of the protocol $\mathcal{A}$, the memory state of the streaming algorithm (of space complexity s) is communicated $(r-1)p+(p-1)$ times and $p-1$ bits are communicated at the end by player P_p to broadcast the output. Hence, the communication cost of the protocol $\mathcal{A}$ is at most $O(rsp)$.

Now we are left to prove the correctness of the protocol $\mathcal{A}$. If there is a column in M with all 1s, then all the vertices corresponding to entries of that column are colored with color C_*. Recall that there is a one-to-one correspondence between the columns in matrix M and cliques in the graph G. So, all the vertices of the clique, corresponding to the column having all 1s, are colored with the color C_*. As the size of each clique in the graph G is $\sqrt{T}$, there are at least $\Omega(T)$ monochromatic edges. To prove the converse, assume that there is no column in the matrix M having all 1s. By UNIQUE INTERSECTION PROMISE, all the columns have Hamming weight at most 1. We will argue that there is no monochromatic edge in G. Consider an edge e in G. By the structure of G, the two vertices of e must be in the same clique, say the j-th clique, that is, let $e = \{v_{i_1 j}, v_{i_2 j}\}$. By the coloring scheme used by the protocols, $v_{i_1 j}$ and $v_{i_2 j}$ are colored according to the values of $M_{i_1 j}$ and $M_{i_2 j}$, respectively. Note that both $M_{i_1 j}$ and $M_{i_2 j}$ belong to j-th column. As the Hamming weight of every column is at most 1, there are three possibilities:

(i) $M_{i_1 j} = M_{i_2 j} = 0$, that is, $v_{i_1 j}$ and $v_{i_2 j}$ are colored with color C_{i_1} and C_{i_2}, respectively;

(ii) $M_{i_1 j} = 0$ and $M_{i_2 j} = 1$, that is, $v_{i_1 j}$ and $v_{i_2 j}$ are colored with color C_{i_1} and C_*, respectively;

(iii) $M_{i_1 j} = 1$ and $M_{i_2 j} = 0$, that is, $v_{i_1 j}$ and $v_{i_2 j}$ are colored with color C_* and C_{i_2}, respectively.

In any case, the edge $e = \{v_{i_1 j}, v_{i_2 j}\}$ is not monochromatic. This establishes the correctness of protocol $\mathcal{A}$ for DISJOINTNESS$_R\left(\sqrt{T}, \frac{n}{\sqrt{T}}, p\right)$. □

7.4 CONFLICT-EST in VA and VADEG models

In this Section, we design algorithms for CONFLICT-EST problem in the VA and VADEG models. Mainly, we prove the following two theorems here. We show matching lower bounds in Appendix A.1.

Theorem 7.11. *Given any graph $G = (V, E)$ and a coloring function $f : V \to [C]$ as input in the stream, there exists an algorithm that solves the* CONFLICT-EST *problem in the* VA *model with high probability in $\widetilde{\mathcal{O}}\left(\min\left(|V|, \frac{|V|^2}{T}\right)\right)$ space, where T is a lower bound on the number of monochromatic edges in the graph.*

Theorem 7.12. *Given any graph $G = (V, E)$ and a coloring function $f : V \to [C]$ as input in the stream, there exists an algorithm that solves the* CONFLICT-EST *problem in the* VADEG *model with high probability in $\widetilde{\mathcal{O}}\left(\min\{|V|, \frac{|E|}{T}\}\right)$ space, where T is a lower bound on the number of monochromatic edges in the graph.*

7.4.1 Motivating ideas for the algorithms

Before going to the algorithms for CONFLICT-EST problem in the VA and VADEG model, we discuss as a warm-up, a two-pass algorithm for CONFLICT-EST in the VA model that uses $\widetilde{\mathcal{O}}\left(\min\{|V|, \frac{|E|}{T}\}\right)$ space, where T is the promised lower bound on the number of monochromatic edges in the graph. Here we assume that $|E|$ is known to the algorithm. However, this assumption can be removed easily in a setting with two passes.

A two-pass algorithm for CONFLICT-EST **in** VA **model (an informal description):**

If $T \leq \frac{|E|}{|V|}$: Our algorithm stores all the vertices and their colors. Thus we can determine the number of monochromatic edges exactly. The algorithm in this case is one pass and uses $\widetilde{\mathcal{O}}(|V|)$ space.

If $T > \frac{|E|}{|V|}$: In the first pass, store each edge with probability $\widetilde{\mathcal{O}}\left(\frac{1}{T}\right)$. In the second pass, we check each edge stored in the first pass for conflict. In this way, we determine the number of monochromatic edges in the sample, from which, we can obtain a desired approximation of the number of monochromatic edges in the graph. The space complexity of our algorithm in this case is $\widetilde{\mathcal{O}}\left(\frac{|E|}{T}\right)$.

If only one pass is allowed, the above algorithm, when $T > \frac{|E|}{|V|}$, can not be simulated in the VA model because of the following reason. Consider an edge $(u, v) \in E_M$ such that u is exposed before v. Note that we will be able to know about the edge only when v is exposed but we will be able to check whether $(u, v) \in E_M$ only when we have stored u and its color. However, there is no clue about the edge (u, v) when u is exposed. So, to solve it in one-pass, we sample each pair of vertices (without bothering if there is an edge between them) with probability $\widetilde{\mathcal{O}}\left(\frac{1}{T}\right)$, before the start of the stream, and determine the number of monochromatic edges in the sample to get an estimate of the number of monochromatic edges in G. This implies that the space complexity of the algorithm for CONFLICT-EST in VA model is $\widetilde{\mathcal{O}}\left(\frac{|V|^2}{T}\right)$ as stated in Theorem 7.11. In the VADEG model, when u is exposed we learn $d_G(u)$ and hence $d^+_G(u)$. The degree information, when u is exposed, gives some statistics regarding how the vertex u might be useful in the future. We exploit this advantage of the VADEG model over the VA model to get an algorithm for CONFLICT-EST that has better space complexity (See Theorem 7.12).

7.4.2 Proof of Theorem 7.11

Our algorithm for CONFLICT-EST for the VA model- first checks if $T \leq |V|$. If yes, we store all the vertices along with their colors to estimate the number of monochromatic edges in the graph exactly. So, the space used by the algorithm is $\widetilde{\mathcal{O}}(|V|)$ when $T \leq |V|$. We will be done by giving an algorithm for CONFLICT-EST in VA model that uses $\widetilde{\mathcal{O}}\left(\frac{|V|^2}{T}\right)$ space. This algorithm will only be executed when $T > |V|$.

Let $V = \{v_1, \ldots, v_n\}$ be the vertices of the graph. Our algorithm starts by generating a sample Z of vertex pairs where each $\{v_i, v_j\}$ is added to Z, independently, with probability $\frac{30 \log n}{\varepsilon^2 T}$. Note that Z is obtained before the start of the stream. Over the stream, we check the following for each $\{v_i, v_j\} \in Z$: whether $(v_i, v_j) \in E$ and is monochromatic. Let $S \subseteq Z$ be the set of monochromatic edges in Z. Note that the expected value of $|S|$ is given by $\mathbb{E}[|S|] = \frac{30 \log n}{\varepsilon^2 T} |E_M|$.

We report $\widehat{m} = \frac{\varepsilon^2 T}{30 \log n} |S|$ as our estimate for $|E_M|$. Applying Chernoff bound (See Lemma 2.5 in Section 2.1), we guarantee that

$$\mathbb{P}\left(|\widehat{m} - |E_M|| \geq \varepsilon |E_M|\right) \leq \mathbb{P}\left(||S| - \mathbb{E}[|S|]| \geq \varepsilon \mathbb{E}[|S|]\right) \leq \exp\left(\frac{-\mathbb{E}[|S|]\varepsilon^2}{3}\right) \leq \frac{1}{n^{10}}.$$

Note that the last inequality holds as $\mathbb{E}[|S|] = \frac{30 \log n}{\varepsilon^2 T} |E_M|$ and $|E_M| \geq T$.

Observe that the space used by our algorithm is $\mathcal{O}(|Z|)$ when $T > \frac{|E|}{|V|}$. Notice that $\mathbb{E}[|Z|] = \frac{30 \log n}{\varepsilon^2 T} \binom{n}{2}$. Applying Chernoff bound (See Lemma 2.5 in Section 2.1), we can show that $|Z| = \widetilde{\mathcal{O}}\left(\frac{n^2}{T}\right)$ with high probability.

Putting together the space complexities of our algorithms for the case $T \leq |V|$ and $T > |V|$, we have the desired bound on the space.

7.4.3 Proof of Theorem 7.12

For simplicity of presentation, assume that we know the number of edges $|E|$ in the graph. We will discuss ways to remove this assumption later in Section 7.4.3.2.

7.4.3.1 Algorithm for CONFLICT-EST in VADEG model when $|E|$ is known

Our algorithm for CONFLICT-EST for the VADEG model first checks if $T \leq \frac{|E|}{|V|}$. If $T \leq \frac{|E|}{|V|}$, we store all the vertices along with their colors to estimate the number of monochromatic edges in the graph exactly. So, the space used by the algorithm is $\widetilde{\mathcal{O}}(|V|)$ when $T \leq \frac{|E|}{|V|}$. We will be done by giving an algorithm for CONFLICT-EST in VADEG model that uses $\widetilde{\mathcal{O}}\left(\frac{|E|}{T}\right)$ space. This algorithm will be executed only when $T > \frac{|E|}{|V|}$.

Let $V = \{v_1, \ldots, v_n\}$ and without loss of generality, the vertices are exposed in the order $v_1, \ldots, v_n$. However, our algorithm does not know about the ordering of the vertices in the stream. Our algorithm stores the following information.

- A random subset $Y \subset V \times [n]$ that will be generated over the stream;
- a subset A of vertices formed from the first elements in the pairs present in Y; the colors of the vertices are also stored;
- for each vertex $v \in A$, a number ℓ_v that denotes the number of neighbors in $N_G^+(v)$ that have been exposed. So, ℓ_v is initialized to 0 when v gets exposed in the stream and is at most $\left|N_G^+(v)\right|$ at any instance of the stream;
- a subset $S \subseteq E_M$ of the set of monochromatic edges in G.

When a vertex v_j is exposed, our algorithm performs the following steps:

(i) Get $d_G(v_j)$ from the degree oracle and $d_G^-(v_j)$ from the exposed edges and compute $d_G^+(v_j)$;

(ii) Add $(v_j, k), k \in \left[d_G^+(v_j)\right]$, with probability $\frac{30 \log n}{\varepsilon^2 T}$ to Y, independently;

(iii) Add v_j along with its color to A if at least one (v_j, k) is added to Y.

(iv) For each $v_i \in A$ such that $(v_i, v_j) \in E$, increment ℓ_{v_i} by 1.

(v) For each $v_i \in A$ such that $(v_i, \ell_{v_i}) \in Y$, check whether (v_i, v_j) forms a monochromatic edge. If yes, add (v_i, v_j) to S. (This step ensures independence so that Chernoff bounds can be used. See Remark 7.1 below.)

The main idea behind the algorithm for CONFLICT-EST in VADEG model is in Step-(ii). Due to the added power of degree oracle, we are able to sample edges that have not arrived explicitly in the stream. We referred to this phenomenon as *sampling into the future* in Section 7.1.1.

At the end of the stream, we report $\widehat{m} = \frac{\varepsilon^2 T}{30 \log n} |S|$ as the estimate of $|E_M|$. Now, we show that $\mathbb{P}\left(|\widehat{m} - |E_M|| \geq \varepsilon |E_M|\right) \leq \frac{1}{n^{10}}$. Consider a monochromatic edge $(v_i, v_j) \in$

E_M. W.l.o.g., assume that v_j is exposed sometime after v_i is exposed in the stream. Let $r \in \left[d^+_G(v_i)\right]$ be such that v_i has $r-1$ neighbors in $\{v_{i+1}, \ldots, v_{j-1}\}$. So, v_j is the r-th neighbor of v_i exposed after the exposure of v_i. From the description of the algorithm, (v_i, v_j) is added to S if and only if (v_i, r) is added to Y. Note that (v_i, r) can be added to Y only when the vertex v_i is exposed in the stream. Before calculating $E[|S|]$ and applying a Chernoff bound, we focus on the following remark.

Remark 7.1. At the first look, it might appear that the monochromatic edges are not independently added to S. For example, let us consider the following situation. Let (v_i, r'), with $r' \in \left[d^+_G(v_i)\right]$ and $r' \neq r$, is added to Y, that is, v_i is present in A and the color of v_i is stored. So, when v_j gets exposed along with its color, we can check whether (v_i, v_j) is monochromatic irrespective of (v_i, r) being added to Y. But the crucial point is that we add (v_i, v_j) to S only when (v_i, r) is added to Y. However, (v_k, ℓ)s, with $k \in [n]$ and $\ell \in \left[d^+_G(v_k)\right]$, are added to Y, independently. That is, each monochromatic edge in E_M is added to S, independently.

The probability that a monochromatic edge is added to S is $\frac{30 \log n}{\varepsilon^2 T}$. That is, $\mathbb{E}[|S|] = \frac{30 \log n}{\varepsilon^2 T} |E_M|$. Applying a Chernoff bound (See Lemma 2.5 in Section 2.1), we can guarantee that

$$\mathbb{P}\left(|\widehat{m} - |E_M|| \geq \varepsilon |E_M|\right) \leq \mathbb{P}\left(||S| - \mathbb{E}[|S|]| \geq \varepsilon \mathbb{E}[|S|]\right) \leq \exp\left(\frac{-\mathbb{E}[|S|]\varepsilon^2}{3}\right) \leq \frac{1}{n^{10}}.$$

Note that the last inequality holds as $|E_M| \geq T$. Observe that the space used by the algorithm is $\widetilde{\mathcal{O}}(|Y| + |A| + |S|) = \widetilde{\mathcal{O}}(|Y|)$. Note that $\mathbb{E}[|Y|] = \sum_{i=1}^{n} d^+_G(v_i) \cdot \frac{30 \log n}{\varepsilon^2 T} = \frac{30|E| \log n}{\varepsilon^2 T}$. Applying a Chernoff bound (See Lemma 2.5 in Section 2.1), we can say that $|Y| = \widetilde{\mathcal{O}}\left(\frac{|E|}{T}\right)$ with high probability. Putting together the space complexities of our algorithms for the case $T \leq \frac{|E|}{|V|}$ and $T > \frac{|E|}{|V|}$, we have the desired bound on the space.

7.4.3.2 Modifying the algorithm in Section 7.4.3.1 when $|E|$ is unknown

We now turn our attention to the real situation when $|E|$ is unknown by modifying the algorithm devised in Section 7.4.3.1. In the modified algorithm, we maintain a counter

defined as follows.

$$\text{CNT} := \sum_{v \text{ has been exposed}} d_G^+(v).$$

Consider the following observation about CNT that will be used in our analysis. As mentioned earlier, $|V| = n$.

Observation 7.13. At any point of the streaming algorithm, CNT is a lower bound on $|E|$, the number of edges in the graph. Moreover, at the end of the stream, CNT becomes $|E|$. Also, CNT is non-decreasing.

We process the stream by maintaining Y, A and S, as defined in the algorithm in Section 7.4.3.1, for the case $T > \frac{|E|}{|V|}$, until CNT reaches $\tau = 100\,|V|\,T \log n$, with a slight difference. Here, we add each (v_j, ℓ) to Y with probability $\frac{3000 \log n}{\varepsilon^3 T}$ instead of $\frac{30 \log n}{\varepsilon^2 T}$ as in Section 7.4.3.1, where v_j is a vertex exposed while CNT is less than τ and $\ell \in \left[d_G^+(v_j)\right]$. So, we have the following observation that will be used later in our analysis.

Observation 7.14. With high probability, $|Y| = \widetilde{\mathcal{O}}\left(|V|\right)$ for all the instances in the stream while CNT is less than τ.

Proof. Let v_k be the first exposed vertex in the stream when CNT is more than τ. Also, let $U = \sqcup_{j=1}^{k-1}\{(v_j, \ell) : \ell \in [d_G^+(v_j)]\}$, where $\sqcup$ denotes disjoint union. Observe that $|U| = \sum_{j=1}^{k-1} d_G^+(v_j) < \tau$. We construct Y by selecting independently each element of U with probability $\frac{3000 \log n}{\varepsilon^3 T}$. Recall that $\tau = 100|V|T \log n$. So, $\mathbb{E}\left[|Y|\right] = \frac{3000|U| \log n}{\varepsilon^3 T} < \frac{3000\tau \log n}{\varepsilon^3 T} = \frac{300000(\log n)^2|V|}{\varepsilon^3}$. The observation follows by applying Chernoff bound (see Lemma 2.5 (iii) in Section 2.1). □

However, the modified algorithm behaves differently once CNT is more than $\tau = 100\,|V|\,T \log n$. Let v_k be as defined earlier. We maintain two extra objects, as described below, after CNT crosses τ.

- The set of vertices $B = \{v_k, \ldots, v_n\}$ and their colors;

- A counter $C_{>\tau}$ that denotes the number of monochromatic edges having both the endpoints in B.

The formal description of the modified algorithm is presented in Algorithm 7.2.

Algorithm 7.2: CONFLICT-EST-DEG(ε, T): CONFLICT-EST problem in the VADEG model

Input: $G = (V, E)$ and a coloring function f on V in the VADEG model, parameters T and ε where $\varepsilon, T \geq 0$.

Output: $\widehat{m}$, that is, a $(1 \pm \varepsilon)$ approximation to $|E_M|$.

for *(each exposed vertex v_j)* **do**

  For each $v_i \in A$ such that $(v_i, v_j) \in E$, increment ℓ_{v_i} by 1;

  For each $v_i \in A$ such that $(v_i, \ell_{v_i}) \in Y$, check whether (v_i, v_j) forms a monochromatic edge. If yes, add (v_i, v_j) to S;

  Set $d^-_G(v_j)$ equals to the number of neighbors that v_j has in $\{v_1, \ldots, v_{j-1}\}$.

  Get $d_G(v_j)$ from the degree oracle and compute $d^+_G(v_j)$. Set CNT $=$ CNT $+ d^+_G(v_j)$.

  if *(*CNT $\leq \tau$*)* **then**

    (i) Add $(v_j, \ell), \ell \in \left[d^+_G(v_j)\right]$, with probability $\frac{3000 \log n}{\varepsilon^3 T}$ to Y, independently;

    (ii) Add v_j to A (with its color stored) if at least one (v_j, ℓ) is added to Y.

  else if *(*CNT $> \tau$*)* **then**

    (i) Add v_j to B (along with the color of v_j);

    (ii) For each $v_i \in B$, check whether (v_i, v_j) forms a monochromatic edge. If yes, increment $C_{>\tau}$ by 1.

If $|S| \leq \frac{60 \log n}{\varepsilon^2}$, then set $C_{\leq\tau} = 0$. Otherwise, set $C_{\leq\tau} = \frac{\varepsilon^3 T}{3000 \log n} |S|$.

Report $\widehat{m} = C_{\leq\tau} + C_{>\tau}$ as the OUTPUT.

We describe the algorithm and its analysis by breaking the range of $|E|$ into two cases, that is, $T \geq \frac{|E|}{100|V| \log n}$ (or $|E| \leq 100T|V| \log n = \tau$) and $T < \frac{|E|}{100|V| \log n}$ (or $|E| > 100T|V| \log n = \tau$). We show that the space complexity of the modified algorithm is

$\widetilde{\mathcal{O}}\left(\frac{|E|}{T}\right)$ in the first case and is $\widetilde{\mathcal{O}}(|V|)$ in the latter case with high probability. Observe that this will imply the desired result as claimed in Theorem 7.12.

$|E| \leq 100T|V|\log n$: In this case, by Observation 7.13, CNT never goes beyond $\tau = 100|V|T\log n$. That is, the algorithm behaves exactly same as that of the algorithm presented in Section 7.4.3.1 for the case $T > \frac{|E|}{|V|}$. Hence, the algorithm reports the desired output using $\widetilde{\mathcal{O}}\left(\frac{|E|}{|T|}\right)$ space, with high probability.

$|E| > 100T|V|\log n$: In this case, by Observation 7.13, there will be an instance (say when vertex v_k is exposed) such that CNT goes beyond τ for the first time. Then we start storing all the vertices and their colors in $B = \{v_k, \ldots, v_n\}$. We stop updating Y and A after v_k is exposed. However, we update S until end of the stream as we were doing previously in Section 7.4.3.1. Along with S, we maintain the number of monochromatic edges (say $C_{>\tau}$) having both the endpoints in $B = \{v_k, \ldots, v_n\}$. Note that $C_{>\tau}$ is maintained exactly. Finally, we report $\widehat{m} = C_{\leq\tau} + C_{>\tau}$ as the output, where 0 or $C_{\leq\tau} = \frac{\varepsilon^3 T}{3000\log n}|S|$ depending on whether $|S| \leq \frac{60\log n}{\varepsilon^2}$ or not, respectively. By Observation 7.14, with high probability, $|Y| = \widetilde{\mathcal{O}}(|V|)$ for all the instances when CNT is less than τ (that is before the exposure of v_k). Also, after the exposure of v_k, we are storing all the vertices along with their colors explicitly. So, the space used by the algorithm is $\widetilde{\mathcal{O}}(|V|)$, with high probability. To see the correctness of the algorithm, let E_M^B be the set of monochromatic edges having both the endpoints in $B = \{v_k \ldots, v_n\}$. Note that $\left|E_M^B\right| = C_{>\tau}$. Let $E_M^{V(G)\setminus B}$ be the set of monochromatic edges having at least one vertex in the set $V(G) \setminus B = \{v_1, \ldots, v_{k-1}\}$, that is, $E_M^{V(G)\setminus B} = E_M \setminus E_M^B$. Using Chernoff bound arguments (see Lemma 2.5 in Section 2.1), we have the following lemma.

Lemma 7.15. *(i) If $\left|E_M^{V(G)\setminus B}\right| \geq \frac{\varepsilon}{100}T$, then $\frac{\varepsilon^3 T}{3000\log n}|S|$ is a $\left(1 \pm \frac{\varepsilon}{100}\right)$ approximation to $\left|E_M^{V(G)\setminus B}\right|$ with probability at least $1 - \frac{1}{n^{10}}$.*

(ii) If $\left|E_M^{V(G)\setminus B}\right| \leq \frac{\varepsilon}{100}T$, $|S| \leq \frac{60\log n}{\varepsilon^2}$ with probability at least $1 - \frac{1}{n^{10}}$.

Proof. We use a similar argument as that of Section 7.4.3.1 to show $\widehat{m}$ is a $(1\pm\varepsilon)$-approximation of $|E_M|$.

Here, $\mu = \mathbb{E}[|S|] = \frac{3000\log n}{\varepsilon^3 T}|E_m^a|$. We prove (i) and (ii) separately.

(i) As $|E_M^a| \geq \frac{\varepsilon}{100}T$, $\mathbb{E}[|S|] \geq \frac{30\log n}{\varepsilon^2}$. Applying Lemma 2.5 (i) and (ii),

$$\begin{aligned}\mathbb{P}\left(||S| - \mathbb{E}[|S|]| \geq \varepsilon\mathbb{E}[|S|]\right) &\leq 2\exp\left(-\frac{\varepsilon^2\mathbb{E}[|S|]}{3}\right)\\ \mathbb{P}\left(\left|\frac{\varepsilon^3 T}{3000\log n}|S| - |E_M^a|\right| \geq \varepsilon|E_M^a|\right) &\leq \frac{1}{n^{10}}.\end{aligned}$$

Observe thet we are done with the claim.

(ii) As $|E_M^a| \leq \frac{\varepsilon}{100}T$, $\mathbb{E}[|S|] \leq \frac{30\log n}{\varepsilon^2}$. Applying Lemma 2.5 (iii), by taking $t = \frac{30\log n}{\varepsilon^2}$ and $\delta = 1$, we have

$$\begin{aligned}\mathbb{P}\left(|S| \geq (1+\delta)t|\right) &\leq \exp\left(-\frac{\delta^2 t}{3}\right)\\ \mathbb{P}\left(|S| \geq \frac{60\log n}{\varepsilon^2}\right) &\leq \frac{1}{n^{10}}.\end{aligned}$$

Observe that, we are done with the claim.

□

Now let us divide the analysis into two cases, that is, $|S| \geq \frac{60\log n}{\varepsilon^2}$ and $|S| < \frac{60\log n}{\varepsilon^2}$.

$|S| \leq \frac{60\log n}{\varepsilon^2}$: In this case, we set $C_{\leq\tau} = 0$. So, $\widehat{m} = C_{>\tau} = \left|E_M^B\right|$ is the output, which is always bounded above by $|E_M|$. By Lemma 7.15 (i), $|S| \leq \frac{60\log n}{\varepsilon^2}$ implies $\left|E_M^{V(G)\setminus B}\right| \leq \frac{\varepsilon}{25}T$ with probability at least $1 - \frac{1}{n^{10}}$. Note that $|E_M| = \left|E_M^{V(G)\setminus B}\right| + \left|E_M^B\right|$ and $|E_M| \geq T$. Putting everything together, $\widehat{m} = C_{\leq\tau} + C_{>\tau}$ lies between $\left(1 - \frac{\varepsilon}{25}\right)|E_M|$ and $|E_M|$, with probability at least $1 - \frac{1}{n^{10}}$.

$|S| > \frac{60\log n}{\varepsilon^2}$: In this case, we set $C_{\leq\tau} = \frac{\varepsilon^3 T}{3000\log n}|S|$. By Lemma 7.15 (ii), $|S| > \frac{60\log n}{\varepsilon^2}$ implies $\left|E_M^{V(G)\setminus B}\right| > \frac{\varepsilon}{100}T$ with probability at least $1 - \frac{1}{n^{10}}$.

Also, by Lemma 7.15 (i), $\left|E_M^{V(G)\setminus B}\right| > \frac{\varepsilon}{100}|T|$ implies $C_{\leq\tau} = \frac{\varepsilon^3 T}{3000\log n}|S|$ is a $\left(1 \pm \frac{\varepsilon}{100}\right)$ approximation to $\left|E_M^{V(G)\setminus B}\right|$ with probability at least $1 - \frac{1}{n^{10}}$. Combining it with the fact that $C_{>\tau} = \left|E_M^B\right|$, we have $\widehat{m} = C_{\leq\tau} + C_{>\tau}$ is an $(1 \pm \varepsilon)$-approximation to $|E_M|$, with probability at least $1 - \frac{2}{n^{10}}$.

This finishes the proof for the case $|E| > 100T|V|\log n$.

We have proved the correctness of Algorithm 7.2 by considering the cases $|E| \leq 100T|V|\log n$ and $|E| > 100T|V|\log n$ separately. We have also shown that the space complexity of Algorithm 7.2 is $\widetilde{\mathcal{O}}\left(\frac{|E|}{T}\right)$ in the former case and is $\widetilde{\mathcal{O}}(|V|)$ in the latter case with high probability. Hence, we are done with the proof of Theorem 7.12.

7.5 Conclusion and discussion

In this thesis, we introduced a graph coloring problem to streaming setting with a different flavor – the coloring function streams along with the graph. We study the problem of CONFLICT-EST in VA, VADEG, and VARAND models. Our algorithms for VA and VADEG are tight upto polylogarithmic factors. However, a matching lower bound on the space complexity for VARAND model is still elusive. There is a gap between our upper and lower bound results for VARAND model in terms of the exponent in T. Our hunch is that the upper bound is tight. Specifically, we obtained an upper bound of $\widetilde{\mathcal{O}}\left(\frac{n}{\sqrt{T}}\right)$ and the lower bound is $\Omega\left(\frac{n}{T^2}\right)$. However, we feel that our algorithm for CONFLICT-EST in VARAND model is tight upto polylogarithmic factors. Two questions naturally follow from our work. The first one is obvious – does CONFLICT-EST in VARAND model admit a lower bound of $\Omega\left(\frac{n}{\sqrt{T}}\right)$? The second one is more model centric. If the VARAND model is equipped with the degree oracle (let us call this model VADEGRAND), can we have an algorithm, for CONFLICT-EST with space complexity $o\left(\frac{n}{\sqrt{T}}\right)$? Though we do not have any answer to the above question, our hunch is in the negative direction, that is, CONFLICT-EST will in all possibility admit a lower bound of $\Omega\left(\frac{n}{\sqrt{T}}\right)$ space in the VADEGRAND model. Here, we would like to note that the lower bound of $\Omega\left(\frac{n}{T^2}\right)$

in VARAND model also holds in VADEGRAND. This is because all the vertices are of degree $\sqrt{T} - 1$ in the graph considered in Theorem 7.10 (see Section 7.3).

We also feel the *edge coloring* counterpart of the vertex coloring problem proposed in the work will be worthwhile to study. Let the edges of G be colored with a function $f : E(G) \rightarrow [C]$, for $C \in \mathbb{N}$. A vertex $u \in V(G)$ is said to be a *validly* colored vertex if no two edges incident on u have the same color. An edge coloring is valid if all vertices are validly colored. Consider the AL model for the edge coloring problem. As all edges incident on an exposed vertex u are revealed in the stream, if we can solve a duplicate element finding problem on the colors of the edges incident on u, then we are done, but duplicate finding is not an easy problem! It seems at a first glance that all the three models of VA, AL and EA will be difficult to handle for the edge coloring problem on streams of graph and edge colors. It would be interesting to see if the edge coloring variant of the problems we considered in this thesis admit efficient streaming algorithms. We plan to look at this problem next.

Chapter 8

Conclusion and Future Directions

In this thesis, we considered some estimation problems in the query complexity and streaming framework. The relevant open problems corresponding to each problem are discussed in the corresponding sections. Some more open problems are as follows:

BIS **and higher order structures:** BIS could estimate the number of edges in a graph, while TIS and GPIS estimated the number of triangles and hyperedges, respectively. The current trend seems to be that we had oracles specific to problems. We would obviously want to get into newer territory for the oracles by asking what is the query complexity of triangle estimation, clique estimation or in general subgraph estimation when we have BIS oracle access to the underlying graph?

Connection with other oracles: We would seek connections, if any, between BIS oracle and other global queries or other models of computation.

Lower bounds: We still do not have any framework for lower bound proofs for BIS. We would like to develop a general lower bound framework for global queries like BIS.

Parameterized streaming: We would like to do an exhaustive study of parameterized query complexity of NP-hard problems other than d-HITTING SET when we have BIS or GPIS access to underlying graph or hypergraph.

Subgraph counting in streams: One of the findings of this thesis for a particular coloring problem was that random order stream gains a lot over other streaming formats. We want to explore this further to find the streaming complexity of subgraph counting in different random order streaming models for graphs?

Hypergraph coloring in streaming: Extending the work in this thesis, we want to find the streaming complexity of monochromatic hyperedge estimation when hyperedges of a d-uniform hypergraph arrive in random order vertex arrival model.

Appendix A

Appendix for Chapter 7

A.1 Lower bounds for CONFLICT-EST in VA and VADEG models

We show a tight lower bound of $\Omega\left(\min\{|V|, \frac{|V|^2}{T}\}\right)$ for the CONFLICT-EST problem in the vertex arrival model in Section A.1.1. For the CONFLICT-EST problem in the vertex arrival with degree oracle model, we show a tight lower bound of $\Omega\left(\min\{|V|, \frac{|E|}{T}\}\right)$ in Section A.1.2. These bounds are proved using reductions from INDEX problem, discussed in Section 2.2, in the one-way communication complexity model to the CONFLICT-EST problem in graphs (in the vertex arrival streaming models). In INDEX problem on N bits, Alice gets $\mathbf{x} \in \{0,1\}^N$ and Bob gets $j \in [N]$, and the objective of Alice to send a single message to Bob and the objective of Bob is to determine x_j. Recall that (the one way communication complexity) of INDEX problem on N bits is $\Omega(N)$.

A.1.1 Lower bound for CONFLICT-EST in VA model

Theorem A.1. *Let $n, m, T \in \mathbb{N}$ be such that $1 \leq T \leq \binom{n}{2}$ and $m \geq T$. Any one pass streaming algorithm; that takes the vertices and edges of a graph $G(V, E)$ (with $|V| = \Theta(n)$ and $|E| = \Theta(m)$) and coloring function $f : V \to [C]$ on the vertices, in*

VA *model; and determines whether the number of monochromatic edges in* G *is* 0 *or* T *with probability* $2/3$*; requires* $\Omega\left(\min\{n, \frac{n^2}{T}\}\right)$ *bits of space.*

Proof. We show that the lower bound is $\Omega(n)$ when $T \leq \frac{n}{2}$ and $\Omega\left(\frac{n^2}{T}\right)$ when $T > \frac{n}{2}$, separately, to get the stated lower bound. We give a reduction from the INDEX problem on N bits (N to be fixed later) to the CONFLICT-EST problem in graphs with $\Theta(n)$ vertices, $\Theta(m)$ edges and having at least T conflicting edges, in the vertex arrival model. We show our reduction when $m = T$, but we can modify it for any $m \geq T$.

The reduction works as follows. For $T \leq \frac{n}{2}$, Alice has an N-bit input string $\mathbf{x} \in \{0,1\}^N$. For each input bit x_i, Alice creates a vertex p_i. If x_i equals 1, then vertex p_i is colored with color C_1, else it is colored with color C_0. After processing all bits of her input, Alice sends the current memory state to Bob. Let $j \in [N]$ be the input of Bob. Bob constructs a gadget Q which is an independent set of $(n - N)$ vertices and colors all the vertices in the gadget with color C_1. He adds all the edges from the vertex p_j to the gadget Q. The number of vertices in the graph is $(N) + (n - N) = n$ and the number of edges in the graph is $m = T$. We set $N = n - T$. If $x_j = 0$, then the color of p_j is C_0 and there are 0 conflicting edges, where as if $x_j = 1$, then the color of p_j is C_1 and there will be T conflicting edges. Therefore, for $N = n - T$, deciding whether the number of monochromatic edges in the graph is 0 or T, requires $\Omega(n - T)$ or $\Omega(n)$ space.

For $T \geq \frac{n}{2}$, Alice has an N-bit input string $\mathbf{x} \in \{0,1\}^N$. For each input bit x_i, Alice constructs an independent set P_i of size $\frac{2T}{n}$. If x_i equals 1, then the vertices of P_i are colored with color C_1, else the vertices are colored with color C_0. After processing all bits of her input, Alice sends the current memory state to Bob. Let $j \in [N]$ be the input of Bob. Let $j \in [N]$ be the input of Bob. Bob constructs a gadget Q which is an independent set of $\frac{n}{2}$ vertices and colors all the vertices with the color C_1. He adds all the edges from the gadget P_j to the gadget Q. We set $N = \frac{n^2}{T}$. The number of vertices in the graph is $\frac{2T}{n} \cdot (N) + \frac{n}{2} = \Theta(n)$ and the number of edges in the graph is $m = T$. If $x_j = 0$, then the color of vertices in P_j is C_0 and there are 0 conflicting edges, where as if $x_j = 1$, then the color of vertices in P_j is C_1 and there will be T conflicting edges. Therefore, for $N = \frac{n^2}{T}$, deciding whether the number of monochromatic edges in the

graph is 0 or T, requires $\Omega\left(\frac{n^2}{T}\right)$ space.

Recall that we are doing our reductions for $m = T$. We make the above constructions work for any $m \geq T$ by adding a complete subgraph on $\sqrt{m-T}$ vertices such that none of the edges of the complete subgraph are conflicting. [1] □

A.1.2 Lower bound for CONFLICT-EST in VADEG model

Theorem A.2. *Let $n, T \in \mathbb{N}$ be such that $1 \leq T \leq \binom{n}{2}$. Then there exists an m with $T \leq m \leq \binom{n}{2}$ such that the following happens. Any one pass streaming algorithm; that takes the vertices and edges of a graph $G(V, E)$ (with $|V| = \Theta(n)$ and $|E| = \Theta(m)$) and a coloring function $f : V \to [C]$ on the vertices, in* VADEG *model; and determines whether the number of monochromatic edges in G is 0 or T with probability $2/3$; requires $\Omega\left(\min\{n, \frac{m}{T}\}\right)$ bits of space.*

Proof. We show that the lower bound is $\Omega(n)$ when $m > nT$ and $\Omega\left(\frac{m}{T}\right)$ when $m \leq nT$, separately, to get the stated lower bound. We give a reduction from the INDEX problem on N bits (N to be fixed later) to the CONFLICT-EST problem in graphs with $\Theta(n)$ vertices, $\Theta(m)$ edges and having atleast T conflicting edges, in the vertex arrival model. The existence of m will be evident from the construction.

The reduction works as follows. For $m > nT$, Alice has an N-bit input string $\mathbf{x} \in \{0,1\}^N$. For each input bit x_i, Alice creates a vertex p_i. If x_i equals 1, then vertex p_i is colored with color C_{i1}, else it is colored with color C_{i0}. After processing all bits of her input, Alice sends the current memory state to Bob. Let $j \in [N]$ be the input of Bob. Bob constructs a gadget Q of $n - N$ vertices such that Q is an independent set of $(n - N)$ vertices. Bob colors all the vertices in the gadget Q with color C_{j1}. He adds all the edges from Q to all the vertices in $\{l_i : i \in [N]\}$. The number of vertices in the graph is $N + (n - N) = \Theta(n)$ and the number of edges in the graph is $m = NT$. We set $N = n - T$. If $x_j = 0$, then the color of p_j is C_{j0} and there are 0 conflicting edges, where as if $x_j = 1$, then the color of p_j is C_{j1} and there will be T conflicting edges.

[1] Note that $\sqrt{m-T} = O(n)$.

Therefore, for $N = n - T$, deciding whether the number of monochromatic edges in the graph is 0 or T, requires $\Omega(n - T)$ or $\Omega(n)$ space. Observe that, the degree of the vertices are independent of the inputs of Alice and Bob. In particular, the degree of every vertex in $\{p_i : i \in [N]\}$ is $|Q| = n - N$ and the degree of every vertex in Q is N. So, the availability of degree oracle will not help in the above construction.

For $m \leq nT$, Alice has an N-bit input string $\mathbf{x} \in \{0,1\}^N$. For each input bit x_i, Alice constructs an independent set P_i of size $\frac{2nT}{m}$. If x_i equals 1, then the vertices of P_i are colored with color C_{i1}, else the vertices are colored with color C_{i0}. After processing all bits of her input, Alice sends the current memory state to Bob. Let $j \in [N]$ be the input of Bob. Let $j \in [N]$be the input of Bob. Bob constructs a gadget Q where Q is an independent set of $\frac{m}{2n}$ vertices. Bob colors the vertices in Q with C_{j1} and he adds all the edges from Q to $(P_1 \cup \cdots \cup P_N)$. We set $N = \frac{m}{T}$. The number of vertices in the graph is $\frac{2nT}{m} \cdot N + \frac{m}{2n} = \Theta(n)$ as $T \geq \frac{n}{2}$ and $m \geq T$ and the number of edges in the graph is $m = NT$. If $x_j = 0$, then the color of vertices in P_j is C_{j0} and there are 0 conflicting edges, where as if $x_j = 1$, then the color of vertices in P_j is C_{j1} and there will be T conflicting edges. Therefore, for $N = \frac{m}{T}$, deciding whether the number of monochromatic edges in the graph is 0 or T, requires $\Omega\left(\frac{m}{T}\right)$ space. Observe that, the degree of the vertices are independent of the inputs of Alice and Bob. In particular, the degree of every vertex in $P_1 \cup \cdots \cup P_N$ is $|Q| = \frac{m}{2n}$, and the degree of every vertex in Q is $N \cdot \frac{2Tn}{m}$. So, the availability of degree oracle will not help in the above construction. $\square$

Bibliography

[AA20a] Noga Alon and Sepehr Assadi. Palette sparsification beyond (Δ+1) vertex coloring. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, pages 6:1–6:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. 10, 166

[AA20b] Noga Alon and Sepehr Assadi. Palette sparsification beyond (Δ+1) vertex coloring. *CoRR*, abs/2006.10456, 2020. 172

[ABG+18] Maryam Aliakbarpour, Amartya Shankha Biswas, Themis Gouleakis, John Peebles, Ronitt Rubinfeld, and Anak Yodpinyanee. Sublinear-time algorithms for counting star subgraphs via edge sampling. *Algorithmica*, 80(2):668–697, 2018. 3

[ACK19] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for (Δ + 1) vertex coloring. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 767–786. SIAM, 2019. 10, 166, 171, 172

[ADNK14] Nesreen K. Ahmed, Nick G. Duffield, Jennifer Neville, and Ramana Rao Kompella. Graph sample and hold: a framework for big-graph analytics. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference*

on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014, pages 1446–1455. ACM, 2014. 19

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 5–14. ACM, 2012. 19

[AKK19] Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 6:1–6:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 3

[AKL16] Sepehr Assadi, Sanjeev Khanna, and Yang Li. Tight bounds for single-pass streaming complexity of the set cover problem. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 698–711. ACM, 2016. 6, 128, 135

[AMP+06] Deepak Agarwal, Andrew McGregor, Jeff M. Phillips, Suresh Venkatasubramanian, and Zhengyuan Zhu. Spatial scan statistics: approximations and performance study. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 24–33. ACM, 2006. 152

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. 5

[AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. 19

[BBGM18] Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Triangle estimation using polylogarithmic queries. *CoRR*, abs/1808.00691, 2018. 23, 55, 67, 68, 71

[BBGM19a] Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Hyperedge estimation using polylogarithmic subset queries. *CoRR*, abs/1908.04196, 2019. 4, 5, 8, 50, 51, 57, 98

[BBGM19b] Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Triangle estimation using tripartite independent set queries. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPIcs*, pages 19:1–19:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 4, 7, 8, 23, 55, 57, 61, 67, 68, 71

[BBGM21] Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. On triangle estimation using tripartite independent set queries. *Theory Comput. Syst.*, 65(8):1165–1192, 2021. 7, 95

[BBMU21] Anup Bhattacharya, Arijit Bishnu, Gopinath Mishra, and Anannya Upasana. Even the easiest(?) graph coloring problem is not easy in streaming! In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 15:1–15:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. 11, 169, 170

[BC17] Suman K. Bera and Amit Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 11:1–11:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. 170

[BCG19] Suman K. Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. *CoRR*, abs/1905.00566, 2019. 10, 166, 172

[BDH+19] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Marina Knittel, and Hamed Saleh. Streaming and massively parallel algorithms for edge coloring. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 15:1–15:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 166, 172

[BFL+06] Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In Stijn Vansummeren, editor, *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 253–262. ACM, 2006. 19

[BG18] Suman Kalyan Bera and Prantar Ghosh. Coloring in graph streams. *CoRR*, abs/1807.07640, 2018. 166, 171, 172

[BGK+18a] Arijit Bishnu, Arijit Ghosh, Sudeshna Kolay, Gopinath Mishra, and Saket Saurabh. Parameterized query complexity of hitting set using stability of sunflowers. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao,

editors, *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, volume 123 of *LIPIcs*, pages 25:1–25:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 5, 54, 69

[BGK+18b] Arijit Bishnu, Arijit Ghosh, Sudeshna Kolay, Gopinath Mishra, and Saket Saurabh. Parameterized query complexity of hitting set using stability of sunflowers. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, volume 123 of *LIPIcs*, pages 25:1–25:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 9

[BGK+20] Arijit Bishnu, Arijit Ghosh, Sudeshna Kolay, Gopinath Mishra, and Saket Saurabh. Fixed parameter tractability of graph deletion problems over data streams. In Donghyun Kim, R. N. Uma, Zhipeng Cai, and Dong Hoon Lee, editors, *Computing and Combinatorics - 26th International Conference, COCOON 2020, Atlanta, GA, USA, August 29-31, 2020, Proceedings*, volume 12273 of *Lecture Notes in Computer Science*, pages 652–663. Springer, 2020. 10

[BGM+19] Marc Bury, Elena Grigorescu, Andrew McGregor, Morteza Monemizadeh, Chris Schwiegelshohn, Sofya Vorotnikova, and Samson Zhou. Structural Results on Matching Estimation with Applications to Streaming. *Algorithmica*, 81(1):367–392, 2019. 135

[BGMS18] Arijit Bishnu, Arijit Ghosh, Gopinath Mishra, and Sandeep Sen. On the streaming complexity of fundamental geometric problems. *CoRR*, abs/1803.06875, 2018. 152

[BHR+18] Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with inde-

pendent set oracles. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 38:1–38:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 4, 7, 8, 15, 23, 24, 27, 28, 50, 55, 57, 60, 61, 66, 67, 68, 70, 71, 89, 95

[BHR+20] Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Trans. Algorithms*, 16(4):52:1–52:27, 2020. 23, 95, 98

[BKS02] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 623–632. ACM/SIAM, 2002. 19

[BPWZ14] Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2014. 19

[BS20] Suman K. Bera and C. Seshadhri. How the degeneracy helps for triangle counting in graph streams. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 457–467. ACM, 2020. 6

[BT81] Béla Bollobás and Carsten Thomassen. The size of connected hypergraphs with prescribed covering number. *J. Comb. Theory, Ser. B*, 31(2):150–155, 1981. 103

[Cai96] Leizhen Cai. Fixed-Parameter Tractability of Graph Modification Problems for Hereditary Properties. *Inf. Process. Lett.*, 58(4):171–176, 1996. 131

[CCE+16] Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via Sampling with Applications to Finding Matchings and Related Problems in Dynamic Graph Streams. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1326–1344, 2016. 9, 108, 109, 113, 114, 130, 131, 133, 134, 135, 136, 145, 148, 151, 152

[CCHM15] Rajesh Hemant Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized Streaming: Maximal Matching and Vertex Cover. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1234–1251, 2015. 9, 130, 135

[CCM16] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. *Theory Comput.*, 12(1):1–35, 2016. 182

[CDK19] Graham Cormode, Jacques Dark, and Christian Konrad. Independent sets in vertex-arrival streams. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 45:1–45:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 6, 167

[CF14] Graham Cormode and Donatella Firmani. A Unifying Framework for ℓ_0-Sampling Algorithms. *Distributed and Parallel Databases*, 32(3):315–335, 2014. 146

[CFK+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015. 10, 99, 103, 131, 143, 145, 146, 147

[CGR+14] Artur Czumaj, Oded Goldreich, Dana Ron, C. Seshadhri, Asaf Shapira, and Christian Sohler. Finding cycles and trees in sublinear time. *Random Struct. Algorithms*, 45(2):139–184, 2014. 3

[CJ17] Graham Cormode and Hossein Jowhari. A second look at counting triangles in graph streams (corrected). *Theor. Comput. Sci.*, 683:22–30, 2017. 19

[CJMM17] Graham Cormode, Hossein Jowhari, Morteza Monemizadeh, and S. Muthukrishnan. The Sparse Awakens: Streaming Algorithms for Matching Size Estimation in Sparse Graphs. In *Proceedings of the 25th Annual European Symposium on Algorithms, ESA*, volume 87, pages 29:1–29:15, 2017. 135

[CLW20] Xi Chen, Amit Levi, and Erik Waingarten. Nearly optimal edge estimation with independent set queries. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2916–2935. SIAM, 2020. 4, 23, 98

[CM15] Yixin Cao and Dániel Marx. Interval Deletion Is Fixed-Parameter Tractable. *ACM Trans. Algorithms*, 11(3):21:1–21:35, 2015. 131

[CM16] Yixin Cao and Dániel Marx. Chordal Editing is Fixed-Parameter Tractable. *Algorithmica*, 75(1):118–137, 2016. 131

[DL18] Holger Dell and John Lapinskas. Fine-grained reductions from approximate counting to decision. In Ilias Diakonikolas, David Kempe, and

Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 281–288. ACM, 2018. 23

[DL21] Holger Dell and John Lapinskas. Fine-grained reductions from approximate counting to decision. volume 13, pages 8:1–8:24, 2021. 98

[DLM19] Holger Dell, John Lapinskas, and Kitty Meeks. Approximately counting and sampling small witnesses using a colourful decision oracle. *CoRR*, abs/1907.04826, 2019. 23, 55, 57, 69

[DLM20a] H. Dell, J. Lapinskas, and K. Meeks. Approximately Counting and Sampling Small Witnesses using a Colourful Decision Oracle. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2201–2211, 2020. 96

[DLM20b] Holger Dell, John Lapinskas, and Kitty Meeks. Approximately counting and sampling small witnesses using a colourful decision oracle. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2201–2211. SIAM, 2020. 5, 8, 50, 51, 98

[DP09] D.P. Dubhashi and Alessandro Panconesi. Concentration of measure for the analysis of randomized algoritms. In *Cambridge*, 2009. 13, 14, 15, 31, 33, 76

[EHKR09] Guy Even, Magnús M. Halldórsson, Lotem Kaplan, and Dana Ron. Scheduling with conflicts: online and offline algorithms. *J. Sched.*, 12(2):199–224, 2009. 167

[EHL+18] Hossein Esfandiari, Mohammadtaghi Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming Algorithms for Estimating the Matching Size in Planar Graphs and Beyond. *ACM Trans. Alg.*, 14(4):1–23, 2018. 131, 135

[ELRS15] Talya Eden, Amit Levi, Dana Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 614–633. IEEE Computer Society, 2015. 3, 7, 19, 22

[ER60] Paul Erdős and Richard Rado. Intersection Theorems for Systems of Sets. *Journal of the London Mathematical Society*, s1-35(1):85–90, 1960. 106, 110, 111

[ER18] T. Eden and W. Rosenbaum. Lower Bounds for Approximating Graph Parameters via Communication Complexity. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, volume 116, pages 11:1–11:18, 2018. 123

[ERS18] Talya Eden, Dana Ron, and C. Seshadhri. On approximating the number of k-cliques in sublinear time. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 722–734. ACM, 2018. 3, 22

[Fei06] Uriel Feige. On Sums of Independent Random Variables with Unbounded Variance and Estimating the Average Degree in a Graph. *SIAM J. Comput.*, 35(4):964–984, 2006. 3, 22

[FJP14] Fedor V. Fomin, Bart M. P. Jansen, and Michal Pilipczuk. Preprocessing subgraph and minor problems: When does a small vertex cover help? *Journal of Computer and System Sciences*, 80(2):468–495, 2014. 131, 144

[FK98] Uriel Feige and Joe Kilian. Zero knowledge and the chromatic number. *J. Comput. Syst. Sci.*, 57(2):187–199, 1998. 166

[FK14] Stefan Fafianie and Stefan Kratsch. Streaming kernelization. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science, MFCS*, pages 275–286, 2014. 9, 135

[FLM+16] Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting Forbidden Minors: Approximation and Kernelization. *SIAM J. Discrete Math.*, 30(1):383–410, 2016. 131

[FLMS12] Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 470–479, 2012. 131

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 166

[Gol17] Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017. 3, 17, 23, 98

[GPRW20] Andrei Graur, Tristan Pollner, Vidhya Ramaswamy, and S. Matthew Weinberg. New query lower bounds for submodular function minimization. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 64:1–64:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. 98

[GR08] Oded Goldreich and Dana Ron. Approximating Average Parameters of Graphs. *Random Struct. Algorithms*, 32(4):473–493, 2008. 3, 22, 23

[GRS11] Mira Gonen, Dana Ron, and Yuval Shavitt. Counting stars and other small subgraphs in sublinear-time. *SIAM J. Discret. Math.*, 25(3):1365–1411, 2011. 3, 19

[GVV17] Venkatesan Guruswami, Ameya Velingker, and Santhoshini Velusamy. Streaming Complexity of Approximating Max 2CSP and Max Acyclic Subgraph. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 8:1–8:19, 2017. 6, 128, 135

[IMR+18] Piotr Indyk, Sepideh Mahabadi, Ronitt Rubinfeld, Ali Vakilian, and Anak Yodpinyanee. Set cover in sub-linear time. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2467–2486. SIAM, 2018. 98

[IR78] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978. 19

[IY18] Kazuo Iwama and Yuichi Yoshida. Parameterized testability. *ACM Trans. Comput. Theory*, 9(4):16:1–16:16, 2018. 98, 99, 100, 101

[Jan04] Svante Janson. Large deviations for sums of partly dependent random variables. *Random Struct. Algorithms*, 24(3):234–248, 2004. 24, 25

[JG05] Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In Lusheng Wang, editor, *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, volume 3595 of *Lecture Notes in Computer Science*, pages 710–716. Springer, 2005. 19

[JSP13] Madhav Jha, C. Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthurusamy, editors, *The 19th ACM SIGKDD International Conference on*

Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013, pages 589–597. ACM, 2013. 19

[KKP18] John Kallaugher, Michael Kapralov, and Eric Price. The sketching complexity of graph and hypergraph counting. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 556–567. IEEE Computer Society, 2018. 170

[KKSV17] Michael Kapralov, Sanjeev Khanna, Madhu Sudan, and Ameya Velingker. $1 + \Omega(1)$ approximation to MAX-CUT Requires Linear Space. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1703–1722, 2017. 6, 128, 135

[KLP+16] Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear Kernels and Single-Exponential Algorithms Via Protrusion Decompositions. *ACM Trans. Algorithms*, 12(2):21:1–21:41, 2016. 131

[KMPV19] 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, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 119–133. ACM, 2019. 170

[KMSS12] 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, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 598–609. Springer, 2012. 19, 170

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997. 16, 18, 123

[KP06] Subhash Khot and Ashok Kumar Ponnuswami. Better inapproximability results for maxclique, chromatic number and min-3lin-deletion. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 226–237. Springer, 2006. 166

[KP14] Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic Feedback Vertex Set. *Inf. Process. Lett.*, 114(10):556–560, 2014. 131

[KP17] John Kallaugher and Eric Price. A hybrid sampling scheme for triangle counting. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1778–1797. SIAM, 2017. 19

[Mar10] Dániel Marx. Chordal Deletion is Fixed-Parameter Tractable. *Algorithmica*, 57(4):747–768, 2010. 131

[McG14a] Andrew McGregor. Graph Stream Algorithms: A Survey. *SIGMOD Record*, 43(1):9–20, 2014. 6, 128, 135

[McG14b] Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Rec.*, 43(1):9–20, 2014. 166

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995. 145

[Mul18] Wolfgang Mulzer. Five proofs of chernoff's bound with applications. *Bull. EATCS*, 124, 2018. 14

[MV16] Andrew McGregor and Sofya Vorotnikova. Planar Matching in Streams Revisited. In *APPROX/RANDOM*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016. 135

[MV18] Andrew McGregor and Sofya Vorotnikova. A Simple, Space-Efficient, Streaming Algorithm for Matchings in Low Arboricity Graphs. In *SOSA*, 2018. 135

[MVV16] Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In Tova Milo and Wang-Chiew Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 401–411. ACM, 2016. 6, 170

[ORRR12] Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1123–1131. SIAM, 2012. 3, 98

[PTTW13] Aduri Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proc. VLDB Endow.*, 6(14):1870–1881, 2013. 19

[Rou16] Tim Roughgarden. Communication Complexity (for Algorithm Designers). *Foundations and Trends in Theoretical Computer Science*, 11(3-4):217–404, 2016. 16, 17

[RSV04] Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. 131

[RSW18] Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In Anna R. Karlin, editor,

9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA, volume 94 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 3, 98

[RT16] Dana Ron and Gilad Tsur. The power of an example: Hidden set size approximation using group queries and conditional sampling. *ACM Trans. Comput. Theory*, 8(4):15:1–15:19, 2016. 4, 61

[Rub20] Ronitt Rubinfeld. Sublinear time algorithms. *Lecture Notes*, 2020. 3

[RY20] Anup Rao and Amir Yehudayoff. *Communication Complexity: and Applications*. Cambridge University Press, 2020. 17

[SK12] Isabelle Stanton and Gabriel Kliot. Streaming graph partitioning for large distributed graphs. In Qiang Yang, Deepak Agarwal, and Jian Pei, editors, *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 1222–1230. ACM, 2012. 7

[Sto83] Larry J. Stockmeyer. The complexity of approximate counting (preliminary version). In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 118–126. ACM, 1983. 4, 61

[Sto85] Larry J. Stockmeyer. On approximation algorithms for #p. *SIAM J. Comput.*, 14(4):849–861, 1985. 4, 61

[SW15] Xiaoming Sun and David P. Woodruff. Tight Bounds for Graph Problems in Insertion Streams. In *Proceedings of the 18th International Workshop*

on Approximation Algorithms for Combinatorial Optimization Problems, APPROX, pages 435–448, 2015. 150, 151

[TGRV14] Charalampos E. Tsourakakis, Christos Gkantsidis, Bozidar Radunovic, and Milan Vojnovic. FENNEL: streaming graph partitioning for massive scale graphs. In Ben Carterette, Fernando Diaz, Carlos Castillo, and Donald Metzler, editors, *Seventh ACM International Conference on Web Search and Data Mining, WSDM 2014, New York, NY, USA, February 24-28, 2014*, pages 333–342. ACM, 2014. 7

[Tho10] Stéphan Thomassé. A $4k^2$ Kernel for Feedback Vertex Set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010. 131

[TPT13] Kanat Tangwongsan, Aduri Pavan, and Srikanta Tirthapura. Parallel triangle counting in massive streaming graphs. In Qi He, Arun Iyengar, Wolfgang Nejdl, Jian Pei, and Rajeev Rastogi, editors, *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 781–786. ACM, 2013. 19

[Upa20] Anannya Upasana. The study of rainbow coloring of graphs and graph coloring in streaming. In *MTech Thesis*, 2020. 168, 169, 170

[Viz64] Vadim G. Vizing. On an estimate of the chromatic class of a p-graph. 1964. 166

[Yao79] Andrew Chi-Chih Yao. Some Complexity Questions Related to Distributive Computing (Preliminary Report). In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, STOC*, pages 209–213, 1979. 16

[Zuc07] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007. 166

List of Publications (Based on content of the thesis)

1. **Parameterized Query Complexity of Hitting Set using Stability of Sunflowers**, with Arijit Bishnu, Arijit Ghosh, Sudeshna Kolay and Saket Saurabh.

   Proceedings of the 29th International Symposium on Algorithms and Computation, ISAAC, Volume 123, pp. 25:1 – 25:12, 2018.

   Full version submitted to a *journal*.

2. **Triangle Estimation using Tripartite Independent Set Queries**, with Anup Bhattacharya, Arijit Bishnu and Arijit Ghosh.

   Theory of Computing Systems, online, 2021

   Proceedings of the 30th International Symposium on Algorithms and Computation, ISAAC, Volume 149, pp. 19:1 – 19:17, 2019.

3. **Hyperedge Estimation using Polylogarithmic Subset Queries**, with Anup Bhattacharya, Arijit Bishnu, and Arijit Ghosh.

   CoRR abs/1908.04196, 2019. An improved version of this has recently been accepted in STACS'22.

4. **Fixed-Parameter Tractability of Graph Deletion Problems over Data Streams**, with Arijit Bishnu, Arijit Ghosh, Sudeshna Kolay, and Saket Saurabh

   In *Proceedings of the 26th International Computing and Combinatorics Conference, COCOON*, pp. 652–663, 2020.

   Full version submitted to *Journal*.

5. **Even the Easiest(?) Graph Coloring Problem is not Easy in Streaming!** with Anup Bhattacharya, Arijit Bishnu and Anannya Upasana.

   Proceedings of the 12th Innovations in Theoretical Computer Science Conference, Volume 185, pp. 15:1–15:19, 2021

   Full version submitted to a *Journal*.

Full list of publications by the author (inclusive of the above list)

Sub-linear Algorithms (Published)

1. **Sublinear-Time Algorithm for Matrix Distance using Projections on Hamming Cube**
   with Arijit Bishnu and Arijit Ghosh
   To appear in RANDOM 2021: International Conference on Randomization and Computation

   Full version to be submitted to a *Journal*.

2. **Interplay between Graph Isomorphism and Earth Mover's Distance in the Query and Communication Worlds**
   with Sourav Chakraborty, Arijit Ghosh and Sayantan Sen
   To appear in RANDOM 2021: International Conference on Randomization and Computation

   Full version to be submitted to a *Journal*.

3. **Query Complexity of Global Minimum Cut**
   with Arijit Bishnu, Arijit Ghosh and Manaswi Paraashar
   To appear in APPROX 2021: International Conference on Approximation Algorithms for Combinatorial Optimization Problems

   Full version to be submitted to a *Journal*.

4. **Even the Easiest(?) Graph Coloring Problem is not Easy in Streaming!**
   with Anup Bhattacharya, Arijit Bishnu and Anannya Upasana.

   Proceedings of the 12th Innovations in Theoretical Computer Science Conference, Volume 185, pp. 15:1–15:19, 2021

   Full version submitted to a *Journal*.

5. **Disjointness through the Lens of Vapnik–Chervonenkis Dimension: Sparsity and Beyond**,
   with Anup Bhattacharya, Sourav Chakraborty, Arijit Ghosh, and Manaswi Paraashar.

   Proceedings of the 24th International Conference on Randomization and Computation, RANDOM, Volume 176, pp. 23:1–23:12, 2020.

   Full version submitted to *Journal*.

6. **Fixed-Parameter Tractability of Graph Deletion Problems over Data Streams**,
   with Arijit Bishnu, Arijit Ghosh, Sudeshna Kolay, and Saket Saurabh

   In *Proceedings of the 26th International Computing and Combinatorics Conference, COCOON*, pp. 652–663, 2020.

   Full version submitted to *Journal*.

7. **Triangle Estimation using Tripartite Independent Set Queries**,
   with Anup Bhattacharya, Arijit Bishnu and Arijit Ghosh.

   Theory of Computing Systems, online, 2021

   Proceedings of the 30th International Symposium on Algorithms and Computation, ISAAC, Volume 149, pp. 19:1 – 19:17, 2019.

8. **Parameterized Query Complexity of Hitting Set using Stability of Sunflowers**,
   with Arijit Bishnu, Arijit Ghosh, Sudeshna Kolay and Saket Saurabh.

   Proceedings of the 29th International Symposium on Algorithms and Computation, ISAAC, Volume 123, pp. 25:1 – 25:12, 2018.

   Full version submitted to a *journal*.

Sublinear Algorithms (to be submitted)

9. **Hyperedge Estimation using Polylogarithmic Subset Queries**,
   with Anup Bhattacharya, Arijit Bishnu, and Arijit Ghosh.
   CoRR abs/1908.04196, 2019.

10. **Inner Product Oracle can Estimate and Sample**,
with Arijit Bishnu, Arijit Ghosh, and Manaswi Paraashar.
CoRR abs/1906.07398, 2019.

11. **On the Streaming Complexity of Fundamental Geometric Problems**,
with Arijit Bishnu, Arijit Ghosh and Sandeep Sen.
CoRR abs/1803.06875, 2018.

Others (Metric Embedding, Graph Theory and Combinatorial Geometry)

12. **Grid Obstacle Representation of Graphs**,
with Arijit Bishnu, Arijit Ghosh,Rogers Mathew, and Subhabrata Paul.

    Discrete Applied Mathematics, 296:39–51, 2021,

    Preliminary version appeared as a poster in *Proceedings of the 25th International Symposium on Graph Drawing & Network Visualization, GD*, pp. 603 – 605, 2017.

13. **Existence of Planar Support for Geometric Hypergraphs using Elementary Techniques**,
with Arijit Bishnu, Sameer Desai, Arijit Ghosh and Subhabrata Paul.

    Discrete Mathematics, Volume 343(6), pp 111853, 2020.

14. **FPT Algorithms for Embedding into Low Complexity Graph Metrics**,
with Arijit Ghosh and Sudeshna Kolay.

    ACM Transactions on Computation Theory, 12(1): 1 – 41, 2020.

    Preliminary version appeared in European Symposium on Algorithms (ESA), Volume 112, pp:35:1-35:13, 2018.

15. **Improved Algorithms for Evacuation Route Planning Problem**
with Subhra Mazumdar and Arindam Pal

Journal of Combinatorial Optimization, 36(1): 280 – 306, 2018.

Preliminary version appeared in Proceedings of the 9th Conference on Combinatorial Optimization and Applications (COCOA), Volume 9486, pp. 3-16, 2015.