Counting and Sampling from Substructures Using Linear Algebraic Queries

Arijit Bishnu
Indian Statistical Institute, Kolkata, India

Arijit Ghosh
Indian Statistical Institute, Kolkata, India

Gopinath Mishra[1]
University of Warwick, Coventry, UK

Manaswi Paraashar[1]
Aarhus University, Denmark

Abstract

For an unknown $n \times n$ matrix A having non-negative entries, the *inner product* (IP) oracle takes as inputs a specified row (or a column) of A and a vector $\mathbf{v} \in \mathbb{R}^n$ with non-negative entries, and returns their inner product. Given two input vectors $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^n$ with non-negative entries, and an unknown matrix A with non-negative entries with IP oracle access, we design almost optimal sublinear time algorithms for the following two fundamental matrix problems:

- Find an estimate $\mathcal{X}$ for the *bilinear form* $\mathbf{x}^T A\mathbf{y}$ such that $\mathcal{X} \approx \mathbf{x}^T A\mathbf{y}$.
- Designing a *sampler* $\mathcal{Z}$ for the entries of the matrix A such that $\mathbb{P}(\mathcal{Z} = (i,j)) \approx x_i A_{ij} y_j / \left(\mathbf{x}^T A\mathbf{y}\right)$, where x_i and y_j are i-th and j-th coordinate of $\mathbf{x}$ and $\mathbf{y}$ respectively.

As special cases of the above results, for any submatrix of an unknown matrix with non-negative entries and IP oracle access, we can efficiently estimate the sum of the entries of any submatrix, and also sample a random entry from the submatrix with probability proportional to its weight. We will show that the above results imply that if we are given IP oracle access to the adjacency matrix of a graph, with non-negative weights on the edges, then we can design sublinear time algorithms for the following two fundamental graph problems:

- Estimating the sum of the weights of the edges of an induced subgraph, and
- Sampling edges proportional to their weights from an induced subgraph.

We show that compared to the classical LOCAL queries (degree, adjacency, and neighbor queries) on graphs, we can get a quadratic speedup if we use IP oracle access for the above two problems.

Apart from the above, we study several matrix problems through the lens of IP oracle, like testing if the matrix is diagonal, symmetric, doubly stochastic, etc. Note that IP oracle is in the class of linear algebraic queries used lately in a series of works by Ben-Eliezer et al. [SODA'08], Nisan [SODA'21], Rashtchian et al. [RANDOM'20], Sun et al. [ICALP'19], and Shi and Woodruff [AAAI'19]. Recently, IP oracle was used by Bishnu et al. [RANDOM'21] to estimate dissimilarities between two matrices.

2012 ACM Subject Classification Mathematics of computing → Probabilistic algorithms

Keywords and phrases Query complexity, Bilinear form, Uniform sampling, Weighted graphs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.8

Related Version The missing proofs can be found in the full version of the paper.
Full Version: https://arxiv.org/abs/2201.04975 [13]

[1] The work was done when the author was at Indian Statistical Institute, Kolkata, India.

1 Introduction

Let A be an unknown $n \times n$ matrix with oracle access and with non-negative entries. Consider a submatrix B of A given by two subsets S_1 and S_2 of indices of the rows and columns A, that is, $B := (A_{ij})_{(i,j) \in S_1 \times S_2}$. Note that submatrix B is unknown as the matrix A itself is unknown. We are only given as inputs the sets S_1 and S_2, and oracle access to the unknown matrix A, and we have to *efficiently* solve the following two problems:

- Estimate the sum of the entries in the submatrix B, and
- sample a random entry (i, j) from B proportional to its weight A_{ij}, that is, we output $(i, j) \in S_1 \times S_2$ with probability

$$\frac{A_{ij}}{\sum_{(i,j) \in S_1 \times S_2} A_{ij}}.$$

We will show that using IP oracle access to A, which takes as inputs a specified row (or a column) of A and a vector $\mathbf{v} \in \mathbb{R}^n$ with non-negative entries and returns their inner product (see Section 1.1 for a formal definition), we can design efficient algorithms for the above defined fundamental problems. In fact, we will be giving efficient algorithms for even more general problems, see Section 1.2. Now, we discuss two consequences of our result on the above-mentioned matrix problem.

Counting and sampling edges from induced subgraphs

Let $G = (V(G), E(G))$ be an *unknown graph*[1] on n vertices, and S be any subset of vertices of G. Assume that the query access to graph G is INDUCED DEGREE query oracle which takes a vertex $u \in V(G)$ and a subset $X \subseteq V$ as input and reports the number of neighbors of u that are present in X. Given INDUCED DEGREE query access to G, two natural questions to consider on the subgraph G_S of G induced by the subset S are the following

- estimate the number of edges in G_S, and
- uniformly sample a random edge from G_S.

Observe that both of these graph problems with INDUCED DEGREE query access can be reduced to the matrix problems with IP oracle access to the adjacency matrix of the graph, where we only ask for inner product with the vectors in $\{0, 1\}^n$.

Weighted edge counting and sampling from graphs

For a graph with non-negative weights on its edges, we can similarly consider the problems of estimating the sum of the weights of the edges in the graph, and sampling edges of the graph proportional to its weight. Again, observe that these two fundamental problems on weighted graphs can also be reduced to the matrix problem defined at the beginning of this section by considering their adjacency matrix.

Notation

In this paper, we denote the set $\{1, \ldots, t\}$ by $[t]$ and $\{0, \ldots, t\}$ by $[[t]]$. For a (directed) graph G, $V(G)$ and $E(G)$ denote the vertex set and edge sets of G, we will use V and E when the graph is clear from the context. For a vertex u, let $d_G(u)$ denote the degree of u in G and $N_G(u)$ denote the set of neighbors of u in G. For a subset S of $V(G)$, the subgraph of G induced by S is denoted by $G_S = (S, E_S)$ such that $E_S := \{\{u, v\} \in E(G) \mid u \in S \text{ and } v \in S\}$. The LOCAL queries for a graph $G = (V(G), E(G))$ are:

[1] By unknown we mean that the vertex set $V(G)$ of G is known but the edge set $E(G)$ is not known.

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

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 $\perp$ [2].

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

For a non-empty set X and a given parameter $\varepsilon \in (0, 1)$, an *almost uniform* sample of X means each element of X is sampled with probability values that lie in the interval $[(1-\varepsilon)/|X|, (1+\varepsilon)/|X|]$. For a matrix A, A_{ij} denotes the element in the i-th row and j-th column of A. A_{i*} and A_{*j} denote the i-th row vector and j-th column vector of the matrix A, respectively. $A \in [[\rho]]^{n\times n}$ means $A_{ij} \in [[\rho]]$ for each $i, j \in [n]$, $\rho \in \mathbb{N}$. Throughout this paper, the number of rows or columns of a square matrix A is n, which will be clear from the context. Vectors are matrices of order $n \times 1$ and will be represented using boldface letters. Without loss of generality, we consider n to be a power of 2. The i-th coordinate of a vector $\mathbf{x}$ is denoted by x_i. We denote by $\mathbf{1}$ the vector with all coordinates 1. Let $\{0,1\}^n$ be the set of n-dimensional vectors with entries either 0 or 1. For $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{1_x}$ is a vector in $\{0,1\}^n$ whose i-th coordinate is 1 if $x_i \neq 0$ and 0 otherwise; $\text{nnz}(\mathbf{x}) = |i \in [n] : x_i \neq 0|$ denotes the number of non-zero components of the vector. By $\langle \mathbf{x}, \mathbf{y} \rangle$, we denote the standard inner product of $\mathbf{x}$ and $\mathbf{y}$, that is, $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{n} x_i y_i$. For a, $b \in \mathbb{R}$, we say $a = (1 \pm \varepsilon)b$ if $a \in [(1-\varepsilon)b, (1+\varepsilon)b]$. Similarly, P is a $(1 \pm \varepsilon)$-approximation to Q means $|P - Q| \leq \varepsilon \cdot Q$. *With high probability* means that the probability of success is at least $1 - \frac{1}{n^c}$, where c is a positive constant. $\widetilde{\Theta}(\cdot)$ and $\widetilde{\mathcal{O}}(\cdot)$ hides a $\text{poly}(\log n, 1/\varepsilon)$ term in the upper bound.

1.1 Definition and motivation of INNER PRODUCT oracle

Let $A \in [[\rho]]^{n\times n}$, $\rho \in \mathbb{N}$, be a matrix whose size is known but the entries are unknown. Now given a row index $i \in [n]$ (or, a column index $j \in [n]$) and a vector $\mathbf{v} \in \mathbb{R}^n$, with non-negative entries as input, the INNER PRODUCT query to A reports the value of $\langle A_{i*}, \mathbf{v} \rangle$ ($\langle A_{*j}, \mathbf{v} \rangle$). If the input index is for row (column), we refer to the corresponding query as row (column) IP query. Observe that INDUCED DEGREE query can be implemented for a graph G by IP oracle as a dot product with $\mathbf{1}_S$ (indicator vector for the set S) and the corresponding row of the matrix A that is the 0/1 adjacency matrix of G.

The IP query has both graph theoretic and linear algebraic flavors to it and we will highlight them shortly. It may be mentioned here that IP has been already used to estimate the Hamming distance between two matrices [12].

From a practical point of view, Rashtchian et al. [32] mention that vector-matrix-vector queries would most likely be useful in the context of specialized hardware or distributed environments. Needless to say, the same carries over to IP query. There are many computer architectures that allow us to compute inner products in one cycle of computation with more parallel processors. Inner product computation can be parallelized using *single instruction multiple data* (SIMD) architecture [26]. Modern day GPU processors use instruction-level parallelism. Nvidia GPUs precisely do that by providing a single API call to compute inner products [35, 2]. There are many such architectures where IP query has been given to users directly. Similarly, there are programming language constructs built on SIMD framework that can compute inner products [1].

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

1.2 The problems, results and paper organization

The main matrix-related problems considered in this work involve estimating the bilinear form $\mathbf{x}^T A\mathbf{y}$ and sampling an element of a matrix almost uniformly using IP queries. Bilinear form estimation is inherently interesting as it is a generalization of the problem defined at the beginning of Section 1. Also, bilinear form estimation has huge importance in numerical linear algebra because of its use in calculating node centrality measures like resolvent subgraph centrality and resolvent subgraph communicability [10, 22], Katz score for adapting it to PageRank computing [14], etc.

BILINEAR FORM ESTIMATION$_A(\mathbf{x}, \mathbf{y})$, in short BFE$_A(\mathbf{x}, \mathbf{y})$
Input: Vectors $\mathbf{x} \in [[\gamma_1]]^n$, $\mathbf{y} \in [[\gamma_2]]^n$, IP access to matrix $A \in [[\rho]]^{n\times n}$, and $\varepsilon \in (0, 1)$.
Output: An $(1 \pm \varepsilon)$-approximation to $\mathbf{x}^T A\mathbf{y}$.

SAMPLE ALMOST UNIFORMLY$_A(\mathbf{x}, \mathbf{y})$, in short SAU$_A(\mathbf{x}, \mathbf{y})$
Input: Vectors $\mathbf{x} \in [[\gamma_1]]^n$, $\mathbf{y} \in [[\gamma_2]]^n$, IP access to matrix $A \in [[\rho]]^{n\times n}$, and $\varepsilon \in (0, 1)$.
Output: Report Z satisfying $(1-\varepsilon)\frac{x_i A_{ij} y_j}{\mathbf{x}^T A\mathbf{y}} \le \mathbb{P}(Z = (i, j)) \le (1+\varepsilon)\frac{x_i A_{ij} y_j}{\mathbf{x}^T A\mathbf{y}}$.

For the above problems, our results are stated in Theorem 1.1. We give the sketches of the proofs of the upper bound for a special case in Section 4. The general upper bounds can be derived from the special case and their proof is in the full version [13] of the paper. The lower bound parts of Theorem 1.1 are also proved in the full version [13].

▶ **Theorem 1.1.** BFE$_A(\mathbf{x}, \mathbf{y})$ *and* SAU$_A(\mathbf{x}, \mathbf{y})$ *can be solved by using*

$$\widetilde{\Theta}\left(\frac{\sqrt{\rho\gamma_1\gamma_2}\,(\mathrm{nnz}(\mathbf{x}) + \mathrm{nnz}(\mathbf{y}))}{\sqrt{\mathbf{x}^T A\mathbf{y}}}\right)$$

IP *queries, where* $nnz(\mathbf{x})$ *and* $nnz(\mathbf{y})$ *denotes the number of non-zero entries in* $\mathbf{x}$ *and* $\mathbf{y}$*, respectively.*

The above theorem has several important consequences. An immediate consequence of Theorem 1.1 is the following.

▶ **Corollary 1.2** (Estimating and sampling from submatrices of $0/1$ matrix)**.** [3] *Let* A *with* $n \times n$ *unknown* $0/1$ *matrix with* IP *oracle access. Let* $\varepsilon \in (0, 1)$*, and* S_1 *and* S_2 *be subsets of indices of the rows and columns of the matrix* A*, respectively. For the matrix* $B = (A_{ij})_{i\in S_1, j\in S_2}$ *using* $\widetilde{\Theta}\left(\frac{|S_1|+|S_2|}{\sqrt{\mathrm{nnz}(B)}}\right)$ IP *queries, we can find an estimate* $\mathcal{X}$ *of the number of ones in* B *such that* $\mathcal{X} = (1 \pm \varepsilon) \sum_{(i,j)\in S_1\times S_2} A_{ij}$*, and also design a sampler* $\mathcal{Z}$ *of* $S_1 \times S_2$ *satisfying, for all* $(i, j) \in S_1 \times S_2$*,*

$$\mathbb{P}\left(\mathcal{Z} = (i, j)\right) = (1 \pm \varepsilon)\,\frac{A_{ij}}{\mathrm{nnz}\,(B)}.$$

Furthermore, the IP *queries used by both algorithms only ask for inner products with vectors in* $\{0, 1\}^n$*.*

[3] Even though Corollary 1.3 is more general than Corollary 1.2, we state Corollary 1.2 to show its application in Remark 1.4 (i).

We now define a more general class of matrices. Let $\mathcal{F}(n,\rho)$ be a family of $n \times n$ matrices with non-negative entries such that for all $A \in \mathcal{F}(n,\rho)$ we have

$$\min_{(i,j):A_{ij}>0} A_{ij} \geq 1 \text{ and } \max_{(i,j):A_{ij}>0} A_{ij} \leq \rho.$$

The following result, which is a direct corollary from Theorem 1.1, shows that we can efficiently estimate and sample even from this family.

▶ **Corollary 1.3** (Estimating and sampling from a arbitrary matrix). *Let A with unknown matrix in $\mathcal{F}(n,\rho)$ with* IP *oracle access. Let $\varepsilon \in (0,1)$, and S_1 and S_2 be subsets of indices of the rows and columns of the matrix A, respectively. For the matrix $B = (A_{ij})_{i\in S_1, j\in S_2}$ using $\widetilde{O}\left(\frac{\sqrt{\rho}(|S_1|+|S_2|)}{\sqrt{\sum_{i,j} A_{ij}}}\right)$* IP *queries, we can find an estimate $\mathcal{X}$ of the number of ones in B, that is, $\mathcal{X} = (1\pm\varepsilon)\sum_{i,j} B_{ij}$, and we can also design a sampler $\mathcal{Z}$ of $[n]\times[n]$ satisfying, for all $(i,j) \in S_1 \times S_2$,*

$$\mathbb{P}(\mathcal{Z} = (i,j)) = (1\pm\varepsilon)\frac{A_{ij}}{\sum_{(i,j)\in S_1\times S_2} A_{ij}}.$$

We would like to point out that the above corollaries are also tight because the lower bound part of the proof of Theorem 1.1 holds even for their special cases.

▶ **Remark 1.4** (Consequences of Theorem 1.1).

(i) Induced subgraphs: Given INDUCED DEGREE query access to a unknown graph $G = (V(G), E(G))$ and a subset $S \subset V(G)$ and an $\varepsilon \in (0,1)$ as input, one can estimate the number edges in G_S up to $(1\pm\varepsilon)$ factor by using $\widetilde{\mathcal{O}}\left(\frac{|S|}{\sqrt{m_S}}\right)$ queries and sample a random edge almost uniformly from G_S, where G_S denotes the subgraph of G induced by S. We can also design a sampler for the edges in G_S where each edge $e \in E(G_S)$ gets sampled with probability $(1\pm\varepsilon)\frac{1}{|E(G_S)|}$ using the same number of queries. Note that this result follows from Corollary 1.2 as each INDUCED DEGREE query to graph G is analogous to an IP query to the adjacency matrix of G. It is because the algorithms corresponding to Corollary 1.2 only ask for inner product with vectors in $\{0,1\}^n$.

(ii) Weighted graphs: Given IP oracle access to the adjacency matrix $A \in [[\rho]]^{n\times n}$ [4] of weighted graph $G = (V(G), E(G))$, then we can estimate $\omega(G) := \sum_{(i,j)\in} A_{ij}$ up to $(1\pm\varepsilon)$-approximation using at most $\widetilde{O}\left(\frac{\sqrt{\rho}n}{\sqrt{\omega(G)}}\right)$ IP queries. We can also design a sampler for the edges of the graph G where each edge $(i,j) \in E(G)$ gets sampled with probability $(1\pm\varepsilon)\frac{A_{ij}}{\omega(G)}$ using the same number of queries. Note that this result follows from Corollary 1.3, and is a generalization of the edge estimation results using local queries by Feige [23], and Goldreich and Ron [25].

(iii) Stochastic matrices: Let A be an unknown *doubly stochastic*[5] $n \times n$ matrix with IP oracle access, and each non-zero entry of A is at least $\lambda > 0$. We can design a sampler $\mathcal{Z}$ of $[n]\times[n]$ such that, for all $(i,j) \in [n]\times[n]$, we have $\mathbb{P}(\mathcal{Z} = (i,j)) = (1\pm\varepsilon)A_{ij}$. Our sampler will use at most $\widetilde{O}\left(n/\sqrt{\lambda}\right)$ queries.

[4] Assume that G is a complete graph such that the weights on $\{i,j\} \notin E(G)$ is 0.

[5] A matrix A with non-negative entries is *doubly stochastic* if the sum of the entries of any row or column is one.

In Section 5, we discuss several other matrix problems using IP oracle query which were studied using stronger queries like *matrix vector* and *vector matrix vector* queries [39, 32].

In Section 3 we establish that LOCAL query access (to the entire unknown graph) cannot solve problems in induced subgraphs *efficiently*. Two crucial takeaways are that IP oracle and its derivative, the INDUCED DEGREE oracle act like a local query on any induced subgraphs, and there is a quadratic separation between the powers of LOCAL query and INDUCED DEGREE queries to solve EDGE ESTIMATION and EDGE SAMPLING in induced subgraphs.

2 Inner product oracle vis-a-vis other query oracles

Graph parameter estimation, where the graph can be accessed through query oracles only, has been an active area of research in sub-linear algorithms for a while [25, 19, 20, 34]. There are different granularities at which the graph can be accessed – the query oracle can answer properties about graphs that are local or global in nature. By now, the LOCAL queries have been used for edge [25], triangle [19], clique estimation [20] and has got a wide acceptance among researchers. Apart from the local queries, in the last few years, researchers have also used the RANDOM EDGE query [4, 6], 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 difficult to classify a random edge query as a local query. On the other hand, *global queries* come in different forms. Starting with the subset queries [37, 38, 33], there have been other queries like BIPARTITE INDEPENDENT SET query, INDEPENDENT SET query [8], GPIS query [11, 17], CUT query [34], etc. Linear measurements or queries [5, 3], based on dot product, have been used for different graph problems.

To this collection of query oracles, we introduce a new oracle called INNER PRODUCT (IP) oracle which is a natural oracle to consider for linear algebraic and graph problems. Using this oracle, we solve hitherto unsolved problems (by an unsolved problem, we mean that no non-trivial algorithm was known before) with graph theoretic and linear algebraic flavor, like (a) edge estimation in induced sub-graph; (b) bilinear form estimation; (c) sampling entries of matrices with non-negative entries. We also show weighted edge estimation and edge estimation in induced subgraph as applications of bilinear form estimation. Our lower bound result, for EDGE ESTIMATION in induced subgraph with only LOCAL query access, implies that there is a separation between the powers of LOCAL query and INDUCED DEGREE query. We will show that our newly introduced INNER PRODUCT query oracle can solve problems that can not be solved by the three local queries mentioned even coupled with the random edge query.

Our current survey of the literature (here we do not claim exhaustivity!) shows that a query related to a subgraph was first used in Ben-Eliezer et al. [9], and named as *group query*, where one asks if there is at least one edge between a vertex and a set of vertices. We found the latest query in this league to be the *demand query* (in bipartite graphs where the vertex set are partitioned into two parts left vertices and right vertices) introduced by Nissan [29] – a demand query accepts a left vertex and an order on the right vertices and returns the first vertex in that order that is a neighbor of the left vertex. One can observe that the group and demand queries are polylogarithmically equivalent. Staying on this line of study related to the relation of a vertex with a subset of vertices, we focus on the INDUCED DEGREE query which we feel handles many natural questions.

Query oracle based graph algorithms access the graph at different granularities – this gives rise to a whole gamut of queries with different capacities, ranging from local queries like degree, neighbor, adjacency queries [23, 25] to global queries like independent set based

queries [8, 17], random edge queries [4], and others like group [9] and demand queries [29]. This rich landscape of queries has unravelled many interesting algorithmic and complexity theoretic results [23, 25, 4, 9, 29, 8, 17, 33]. With this in mind, if we turn our focus to the landscape of linear algebraic queries, the most natural query is the *matrix entry* query where one gives an index of the matrix and asks for the value there. Lately, a series of works [32, 39, 36, 7] have used linear algebraic queries like *vector-matrix-vector query* and *matrix-vector* queries. The IP oracle is also motivated by these new query oracles. Notice the huge difference in power between *matrix entry* query and *vector-matrix-vector query* and *matrix-vector* queries. Note that IP query is strictly weaker than these matrix queries but stronger than the *matrix entry* query. We feel there is a need to study linear algebraic queries with intermediate power – the IP query fits in that slot.

3 A query model for induced subgraph problems

To the best of our knowledge, our work is a first attempt towards solving estimation problems in induced subgraphs. We start by showing a separation between LOCAL query and INDUCED DEGREE query using the problems of EDGE ESTIMATION and EDGE SAMPLING in induced subgraph. We now define INDUCED EDGE ESTIMATION and INDUCED EDGE SAMPLING.

INDUCED EDGE ESTIMATION
Input: A parameter $\varepsilon \in (0,1)$ and a subset S of the vertex set V of a graph G.
Output: A $(1 \pm \varepsilon)$-approximation to the number of edges E_S in the induced subgraph.

INDUCED EDGE SAMPLING
Input: A parameter $\varepsilon \in (0,1)$ and a subset S of the vertex set V of a graph G.
Output: Sample each edge $e \in E_S$ with probability between $\frac{1-\varepsilon}{|E_S|}$ and $\frac{1+\varepsilon}{|E_S|}$.

One of the main contributions of this paper is to show that LOCAL queries together with RANDOM EDGE query are *inefficient* for both INDUCED EDGE ESTIMATION and INDUCED EDGE SAMPLING. The lower bound results follow.

▶ **Theorem 3.1** (Lower bound for INDUCED EDGE ESTIMATION using LOCAL queries). *Let us assume that $s, m_s \in \mathbb{N}$ be such that $1 \leq m_s \leq \binom{s}{2}$ and the query algorithms have access to* DEGREE, NEIGHBOR, ADJACENCY *and* RANDOM EDGE *queries to an unknown graph $G = (V(G), E(G))$. Any query algorithm that can decide for all $S \subseteq V(G)$, with $|S| = \Theta(s)$, whether $|E_S| = m_s$ or $|E_S| = 2m_s$, with probability at least $2/3$, requires $\Omega\left(\frac{s^2}{m_s}\right)$ queries.*

▶ **Theorem 3.2** (Lower bound for INDUCED EDGE SAMPLING using LOCAL QUERIES). *Let us assume that $s, m_s \in \mathbb{N}$ be such that $1 \leq m_s \leq \binom{s}{2}$ and the query algorithms have access to* DEGREE, NEIGHBOR, ADJACENCY *and* RANDOM EDGE *queries to an unknown graph $G = (V(G), E(G))$. Any query algorithm that for any $S \subseteq V(G)$, with $|S| = \Theta(s)$, samples the edges in E_S ε-almost uniformly [6], with probability at least $99/100$, will require $\Omega\left(\frac{s^2}{m_s}\right)$ queries [7]. Note that $\varepsilon \in (0,1)$ is given as an input to the algorithm.*

[6] Each edge in E_S is sampled with probability between $(1-\varepsilon)\frac{1}{|E_S|}$ and $(1+\varepsilon)\frac{1}{|E_S|}$.

[7] Let U denote the uniform distribution on E_S. The lower bound even holds even if the goal is to get a distribution that is ε close to U with respect to ℓ_1 distance.

▶ **Remark 3.3.** When $S = V$, Induced Edge Estimation and Induced Edge Sampling are Edge Estimation and Edge Sampling problems, respectively. Both Edge Estimation and Edge Sampling can be solved with high probability by using $\widetilde{\Theta}\left(|V|^2/|E|\right)$ adjacency queries [24]. Notice that these bounds match the lower bounds. Contrast this with the fact that Edge Estimation and Edge Sampling can be solved with high probability by using $\widetilde{\Theta}\left(|V|/\sqrt{|E|}\right)$ local queries, where each local query is either a degree or a neighbor or an adjacency query [25, 21]. Thus, we observe that for Induced Edge Estimation and Induced Edge Sampling, the adjacency query is as good as the entire gamut of local queries and random edge query. On a different note, our results on Bilinear Form Estimation and Almost uniformly Sampling using IP query generalize the above mentioned results on Edge Estimation and Edge Sampling using local queries. Note that IP oracle is a natural query oracle for graphs where the unknown matrix is the adjacency matrix of a graph, and we will discuss that in Remark 3.8 that IP query on the adjacency matrix graphs is stronger than the local queries.

In Section 3.1, we prove Theorems 3.1 and 3.2 by reduction from a problem in communication complexity. In Section 3.2, we discuss the way in which induced degree query simulates local queries in any induced subgraph (see Remark 3.8). This will imply that the lower bound results in Theorems 3.1 and 3.2 can be overcome if we have an access to induced degree query to the whole graph (see Corollary 3.9). However, the implication is more general and will be discussed in Section 3.2.

3.1 Limitations of local and random edge queries

The proofs of the lower bounds use communication complexity. We will first provide a rudimentary introduction to communication complexity. For a detailed introduction to communication complexity refer to the following books [27, 31].

Brief introduction to communication complexity

In two-party communication complexity there are two parties, Alice and Bob, that wish to compute a function $\Pi : \{0,1\}^N \times \{0,1\}^N \to \{0,1\}$. Alice is given $\mathbf{x} \in \{0,1\}^N$ and Bob is given $\mathbf{y} \in \{0,1\}^N$. Let x_i (y_i) denotes the i-th bit of $\mathbf{x}$ ($\mathbf{y}$). While the parties know the function Π, Alice does not know $\mathbf{y}$, and similarly Bob does not know $\mathbf{x}$. Thus they communicate bits following a pre-decided protocol $\mathcal{P}$ in order to compute $\Pi(\mathbf{x}, \mathbf{y})$. We say a randomized protocol $\mathcal{P}$ computes Π if for all $(\mathbf{x}, \mathbf{y}) \in \{0,1\}^N \times \{0,1\}^N$ we have $\mathbb{P}[\mathcal{P}(\mathbf{x}, \mathbf{y}) = \Pi(\mathbf{x}, \mathbf{y})] \geq 2/3$. The model provides the parties access to a common random string of arbitrary length. The cost of the protocol $\mathcal{P}$ is the maximum number of bits communicated, where maximum is over all inputs $(\mathbf{x}, \mathbf{y}) \in \{0,1\}^N \times \{0,1\}^N$. The communication complexity of the function is the cost of the most efficient protocol computing Π.

Proofs of Theorems 3.1 and 3.2

We will use the following problem in our lower bound proofs.

▶ **Definition 3.4** (k-Intersection)**.** Let $k, N \in \mathbb{N}$ such that $k \leq N$. Let $S = \{(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in \{0,1\}^N, \sum_{i=1}^N x_i y_i = k \text{ or } 0\}$. The k-Intersection function over N bits is a partial function denoted by k-Intersection $: S \to \{0,1\}$, and is defined as follows: k-Intersection$(\mathbf{x}, \mathbf{y}) = 1$ if $\sum_{i=1}^N x_i y_i = k$ and 0, otherwise.

▶ **Lemma 3.5** ([27]). *Let $k, N \in \mathbb{N}$ such that $k \leq N$. The randomized communication complexity of k-*Intersection *function on N bits is $\Omega(N/k)$.*

Proof of Theorem 3.1. We give a reduction from m_s-Intersection problem over $N = s^2$ bits. Let $\mathbf{x} = (x_{ij}) \in \{0,1\}^N$ be such that $i, j \in [s]$. Similarly, let $\mathbf{y} \in \{0,1\}^N$. It is promised that Alice and Bob will be given $\mathbf{x}$ and $\mathbf{y}$ such that there are either 0 intersections or exactly m_s intersections, i.e., either $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ or m_s. Now we define a graph $G_{(\mathbf{x},\mathbf{y})}(V(G), E(G))$ as follows where $\sqcup$ denotes disjoint union.

- $|V(G)| = \Theta(s)$. $V(G) = S_A \sqcup S_B \sqcup T_A \sqcup T_B \sqcup C$ such that S_A, S_B, T_A, T_B are independent sets and $|S_A| = |S_B| = |T_A| = |T_B| = s$ and $|C| = \Theta(s)$. Note that $V(G)$ is independent of $\mathbf{x}$ and $\mathbf{y}$;
- The subgraph (of $G_{(\mathbf{x},\mathbf{y})}$) induced by C is a fixed graph, independent of $\mathbf{x}$ and $\mathbf{y}$, having exactly m_s edges. Also there are no edges in $G_{(\mathbf{x},\mathbf{y})}$ between the vertices of C and $V(G) \setminus C$.
- The edges in the subgraph (of $G_{(\mathbf{x},\mathbf{y})}$) induced by $V(G) \setminus C = S_A \sqcup T_A \sqcup S_B \sqcup T_B$ depend on $\mathbf{x}$ and $\mathbf{y}$ as follows. Let $S_A = \{s_i^A : i \in [s]\}$, $T_A = \{t_i^A : i \in [s]\}$, $S_B = \{s_i^B : i \in [s]\}$ and $T_B = \{t_i^B : i \in [s]\}$. For $i, j \in [s]$, if $x_{ij} = y_{ij} = 1$, then $(s_i^A, t_j^B) \in E(G)$ and $(s_i^B, t_j^A) \in E(G)$. For $i, j \in [s]$ if either $x_{ij} = 0$ or $y_{ij} = 0$, then $(s_i^A, t_j^A) \in E(G)$ and $(s_i^B, t_j^B) \in E(G)$;

The graph $G_{(\mathbf{x},\mathbf{y})}$ can be uniquely generated from $\mathbf{x}$ and $\mathbf{y}$. Moreover, Alice and Bob need to communicate to learn *useful* information about $G_{(\mathbf{x},\mathbf{y})}$. Observation 3.6 follows from the construction that shows the relation between the number of edges in the subgraph induced by $S_A \sqcup T_B \sqcup C$ with $\langle \mathbf{x}, \mathbf{y} \rangle$, where $\mathbf{x}, \mathbf{y} \in \{0,1\}^N$ are such that either $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ or $\langle \mathbf{x}, \mathbf{y} \rangle = m_s$.

▶ **Observation 3.6.** (i) $|S_A \sqcup T_B \sqcup C| = \Theta(s)$, (ii) irrespective of x and y: $|E_{S_A}| = |E_{S_B}| = |E_{T_A}| = |E_{T_B}| = 0$, also the degree of each vertex in $S_A \sqcup T_A \sqcup S_B \sqcup T_B$ is same (i.e., s), and (iii) if $\langle \mathbf{x}, \mathbf{y} \rangle = 0$, then $|E_{S_A \sqcup T_B \sqcup C}| = m_s$, (iv) if $\langle \mathbf{x}, \mathbf{y} \rangle = m_s$, then $|E_{S_A \sqcup T_B \sqcup C}| = 2m_s$.

The following observation completes the proof of the theorem.

▶ **Observation 3.7.** Alice and Bob can deterministically determine answer for each local query to graph $G_{(\mathbf{x},\mathbf{y})}$ by communicating $\mathcal{O}(1)$ bits.

We will give the proof of the above observation at the end of this subsection. ◀

Proof of Theorem 3.2. For clarity, we prove the theorem for $\varepsilon = 1/4$. However, the proof can be extended for any $\varepsilon \in (0, 1/2)$. We use the same set up and construction as in Theorem 3.1 with $S = S_A \sqcup S_B \sqcup C$. Let $\mathcal{A}$ be an algorithm that almost uniformly samples edges from the induced graph $G_S = (S, E_S)$ making T queries, with probability 99/100. Using $\mathcal{A}$ we give another algorithm $\mathcal{A}'$ that decides whether $|E_S| = m_s$ or $|E_S| = 2m_s$ by using $\mathcal{O}(T)$ queries, with probability at least 2/3. From the reduction presented in Theorem 3.1, Alice and Bob can use $\mathcal{A}'$ to solve m_s-Intersection over $N = s^2$ bits, and hence $T = \Omega(\frac{s^2}{m_s})$.

$\mathcal{A}'$ runs $\mathcal{A}$ 10 times independently to obtain edges $e_1, \ldots, e_{10}$. Note that each edge e_i is sampled almost uniformly. If at least one e_i satisfies $e_i \in E_{S_A \sqcup T_B}$, then $\mathcal{A}'$ reports that $|E_S| = 2m_s$. Otherwise, $\mathcal{A}'$ reports that $|E_S| = m_s$. The query cost of $\mathcal{A}'$ is $\Theta(T)$.

If $|E_S| = m_s$, then there is no edge in the subgraph induced by $S_A \sqcup T_B$. So, in this case, all the edges reported by $\mathcal{A}'$ are from the subgraph induced by C. Now consider when $|E_S| = 2m_s$. In this case, the subgraph induced by $S_A \sqcup T_B$ and C have exactly m_s edges each. So, by the assumption of the algorithm $\mathcal{A}$, the probability that any particular e_i is present in the subgraph induced by $S_A \sqcup T_B$ is at least $1/2 - \varepsilon = 1/4$ (since we are

analyzing for $\varepsilon = 1/4$). So, under the conditional space that all the ten runs of $\mathcal{A}$ succeed, the probability that none of the ten edges sampled by $\mathcal{A}$ is from the subgraph induced by $S_A \sqcup T_B$ is at most $(1 - 1/4)^{10} < 1/10$. As each run of algorithm $\mathcal{A}$ succeeds with probability at least 99/100, all the ten runs of the algorithm $\mathcal{A}$ succeeds with probability at least 9/10. So, the probability that algorithm $\mathcal{A}'$ succeeds is at least $9/10 \cdot (1 - 1/10) > 2/3$. ◀

We will finish this subsection with the proof of Observation 3.7 used in the proof of Theorem 3.1.

Proof of Observation 3.7.

DEGREE **query:** By Observation 3.6 (ii), the degree of every vertex in $V(G) \setminus C$ is s. Also, the subgraph induced by C is a fixed graph disconnected from the rest. That is Alice and Bob know the degree of every vertex in C. Therefore, any DEGREE query can be simulated without any communication.

NEIGHBOR **query:** Observe that Alice and Bob can get the answer to any neighbor query involving a vertex in C without any communication. Now, consider the set S_A. The labels of the j neighbors of any vertex in $s_i^A \in S_A$ are as follows: for $j \in [s]$, the j-th neighbor of s_i^A is either t_j^B or s_j^A depending on whether $x_{ij} = y_{ij} = 1$ or not, respectively. So, any NEIGHBOR query involving vertex in S_A can be answered by 2 bits of communication. Similar arguments also hold for the vertices in $S_B \sqcup T_A \sqcup T_B$.

ADJACENCY **query:** Observe that each adjacency query can be answered by at most 2 bits of communication, and it can be argued like the NEIGHBOR query.

RANDOM EDGE **query:** By Observation 3.6 (ii), the degree of every vertex in $V(G) \setminus C$ is s irrespective of the inputs of Alice and Bob. Also, they know the entire subgraph induced by the vertex set C. Also, C is disconnected from the rest. Alice and Bob use shared randomness to sample a vertex in V proportional to its degree. Let $r \in V$ be the sampled vertex. They again use shared randomness to sample an integer j in $[d(v)]$ uniformly at random. Then they determine the j-th neighbor of r using NEIGHBOR query. Observe that this procedure simulates a RANDOM EDGE query by using at most 2 bits of communication. ◀

3.2 A query model for induced subgraphs

Observe that INDUCED DEGREE query can simulate any LOCAL queries on subgraphs.

▶ **Remark 3.8.** Let us have an INDUCED DEGREE query oracle access to an unknown graph $G(V, E)$. Consider any $X \subseteq V(G)$ and G_X, the subgraph of G induced by X. Then

(i) Any query to G_X, which is either a DEGREE or ADJACENCY, can be answered by one INDUCED DEGREE query to G.

(ii) Moreover, any NEIGHBOR query to G_X can be answered by $\mathcal{O}(\log |X|)$ INDUCED DEGREE query to G by *binary search*.

The above remark together with the edge estimation result of Goldreich and Ron [25], and edge sampling result of Eden and Rosenbaum [21], will give us the following result.

▶ **Corollary 3.9** (Estimating and sampling edges in induced subgraphs)**.** *Let us assume that the query algorithms have access to* INDUCED DEGREE *query to an unknown graph* $G = (V(G), E(G))$*. There exists an algorithm that takes a subset* $S \subseteq V(G)$ *and* $\varepsilon \in (0, 1)$ *as inputs, and outputs a* $(1 \pm \varepsilon)$*-approximation to* $|E_S|$*, with high probability, using* $\widetilde{\mathcal{O}}\left(|S|/\sqrt{E_S}\right)$ INDUCED DEGREE *queries to* G*. Also, there exists an algorithm that* ε*-almost uniformly samples edges in* E_S*, with high probability, using* $\widetilde{\mathcal{O}}\left(|S|/\sqrt{E_S}\right)$ INDUCED DEGREE *queries.*

▶ **Remark 3.10.** More generally, Remark 3.8 implies that any problem $\mathcal{P}$ on a graph G that can be solved by using $f(|V(G)|, |E(G)|)$ local queries, can also be solved on any induced subgraph G_S, where $S \subseteq V(G)$, of G by using $f(|V(G_S)|, |E(G_S)|) \cdot \mathcal{O}(\log |V(G_S)|)$ INDUCED DEGREE queries.

4 Bilinear form estimating and sampling entries of a matrix

4.1 Algorithm for Bilinear Form Estimation

To give the main ideas behind the algorithm for BILINEAR FORM ESTIMATION, we will discuss, in this section, the algorithm for estimating $\mathbf{1}^T A \mathbf{1}$ using IP access to A, with A being symmetric. The algorithm for the special case is inspired by [25]. In the full version of the paper ([13]) we show how the algorithm for this special case can be extended for the general problem of estimating $\mathbf{x}^T A \mathbf{y}$, where $A \in [[\rho]]^{n \times n}$, $\mathbf{x} \in [[\gamma_1]]^n$ and $\mathbf{y} \in [[\gamma_2]]^n$. We will now give an outline of the proof of the following theorem.

▶ **Theorem 4.1.** *There exists a query algorithm for* BFE *that takes $\varepsilon \in (0, 1/2)$ as input and determines a $(1 \pm \varepsilon)$-approximation to $\mathbf{1}^T A \mathbf{1}$ with high probability by using $\widetilde{\mathcal{O}}\left(\frac{\sqrt{\rho} n}{\sqrt{\mathbf{1}^T A \mathbf{1}}}\right)$* IP *queries to a symmetric matrix $A \in [[\rho]]^{n \times n}$. Moreover, the algorithm only uses* IP *query of the form $\langle A_{k*}, \mathbf{u} \rangle$ for some $k \in [n]$ and $\mathbf{u} \in \{0,1\}^n$.*

The algorithms for BILINEAR FORM ESTIMATION and SAMPLE ALMOST UNIFORMLY (Section 4.2) will use a subroutine, which takes as input a given row $i \in [n]$ of A and a non-empty set $S \subseteq [n]$, and outputs A_{ij}, where $j \in S$, with probability $A_{ij} / \left(\sum_{j \in S} A_{ij}\right)$.

Algorithm 1 REGR$(\mathbf{x}, i)$.

Input: A vector $\mathbf{x} \in \{0,1\}^n$ such that the 1's in $\mathbf{x}$ are consecutive and the number of 1's is a power of 2, an integer $i \in [n]$ and IP access to a matrix A.
Output: Ordered pair (i, j) with probability $\frac{A_{ij} \cdot x_j}{\langle A_{i*}, \mathbf{x} \rangle}$.
begin
  if *(the number of 1's in* $\mathbf{x}$ *is 1)* **then**
    Report the ordered pair (ij^*) where $x_{j^*} = 1$.
  end
  else
    Form a vector $\mathbf{y}$ ($\mathbf{z}$) in $\{0,1\}^n$ by setting second (first) half of the nonzero elements in $\mathbf{x}$ to 0 and keeping the remaining elements unchanged.
    Determine $\langle A_{i*}, \mathbf{y} \rangle$ and $\langle A_{i*}, \mathbf{z} \rangle$.
    With probability $\frac{\langle A_{i*}, \mathbf{y} \rangle}{\langle A_{i*}, \mathbf{x} \rangle}$ report REGR$(\mathbf{y}, \mathbf{i})$ and with probability $\frac{\langle A_{i*}, \mathbf{z} \rangle}{\langle A_{i*}, \mathbf{x} \rangle}$ report REGR$(\mathbf{z}, \mathbf{i})$.
  end
end

▶ **Observation 4.2.** There exists an algorithm REGR (See Algorithm 1) that takes $i \in [n]$ and $\mathbf{x} \in \{0,1\}^n$ as inputs, outputs A_{ij} with probability $A_{ij} x_j / \left(\sum_{j \in [n]} A_{ij} x_j\right)$ by using $\mathcal{O}(\log n)$ IP queries to matrix A.

We will now discuss in the following paragraphs the details of the algorithm (Algorithm 2) for estimating $\mathbf{1}^T A \mathbf{1}$. The ingredients, to prove the correctness of Algorithm 2, are formally stated in Lemma 4.6. The approximation guarantee of Algorithm 2, which matches the guarantee mentioned in Theorem 4.1, is given in Claim 4.7.

Partition of rows of A induced by ε

Given ε as input, we argue that the rows of the symmetric matrix A can be partitioned into "buckets" such that the total number of buckets is small and every row in a particular bucket has approximately the same total weight. Consider a partition of $[n]$, that corresponds to the set of the indices of the rows of the symmetric matrix A, into buckets with the property that all js present in a particular bucket B_i have approximately the same value of $\langle A_{j*}, \mathbf{1}\rangle$. Let $t = \lceil \log_{1+\beta}(\rho n)\rceil + 1$, where $\beta \leq \varepsilon/8$. For $i \in [t]$, we define the set $B_i := \left\{j \in [n] \ : \ (1+\beta)^{i-1} \leq \langle A_{j*}, \mathbf{1}\rangle < (1+\beta)^i\right\}$. Since $A_{ij} \leq \rho$, the maximum number of such buckets B_i required are at most $t = \lceil \log_{1+\beta}(\rho n)\rceil + 1$. Now consider the following fact that will be used in our analysis.

▶ **Fact 4.3.** For every $i \in [t]$, $(1+\beta)^{i-1}\,|B_i| \leq \sum_{j\in B_i}\langle A_{j*}, \mathbf{1}\rangle < (1+\beta)^i\,|B_i|$.

Based on the number of rows in a bucket, we classify the buckets to be either *large* or *small*. To define the large and small buckets, we require a lower bound ℓ on the value of $m = \mathbf{1}^T A\mathbf{1}$. Moreover, let us assume that, $m/6 \leq \ell \leq m$. However, this restriction can be removed by using standard techniques from property testing. For details, see [13, Section 4].

▶ **Definition 4.4.** We fix a threshold $\theta = \frac{1}{t}\cdot\frac{1}{n}\sqrt{\frac{\varepsilon}{8}\cdot\frac{\ell}{\rho}}$. For $i \in [t]$, we define the set B_i to be a *large bucket* if $|B_i| \geq \theta n$, otherwise B_i is defined to be a *small bucket*. Thus, the set of large buckets L is defined as $L = \{i \in [t] : \ |B_i| \geq \theta n\}$, and $[t] \setminus L$ is the set of small buckets.

Let $V, U \subseteq [n]$ be the sets of indices of rows that lie in large and small buckets, respectively. For $I \subseteq [n]$, let $\mathbf{x}_I$ denote the sub-vector of $\mathbf{x}$ induced by the indices present in I. Similarly, for $I, J \subseteq [n]$, let A_{IJ} denote the sub-matrix of A where the rows and columns are induced by the indices present in I and J, respectively. Observe that,

$$\mathbf{1}^T A\mathbf{1} = {\mathbf{1}_V}^T A_{VV}\mathbf{1}_V + {\mathbf{1}_V}^T A_{VU}\mathbf{1}_U + {\mathbf{1}_U}^T A_{UV}\mathbf{1}_V + {\mathbf{1}_U}^T A_{UU}\mathbf{1}_U.$$

2-Approximation of $\mathbf{1^T A 1}$

Note that at this point we know β and, upon querying $\langle A_{j*}, \mathbf{1}\rangle$, we can determine the bucket to which j belongs, for $j \in [n]$. The algorithm begins by sampling a subset S of rows of A, such that $|S| = K$, independently and uniformly at random with replacement, and for each sampled row j, the algorithm determines $\langle A_{j*}, \mathbf{1}\rangle$ by using IP oracle. This determines the bucket in which each sampled row belongs. Depending on the number of sampled rows present in different buckets, our algorithm classifies each bucket as either large or small. Let $\widetilde{V}$ and $\widetilde{U}$ be the indices of the rows present in large and small buckets, respectively. Note that the algorithm does not find $\widetilde{V}$ and $\widetilde{U}$ explicitly – these are used only for analysis purposes.

Observe that,

$$\mathbf{1}^T A\mathbf{1} = {\mathbf{1}_{\widetilde{V}}}^T A_{\widetilde{V}\widetilde{V}}\mathbf{1}_{\widetilde{V}} + {\mathbf{1}_{\widetilde{V}}}^T A_{\widetilde{V}\widetilde{U}}\mathbf{1}_{\widetilde{U}} + {\mathbf{1}_{\widetilde{U}}}^T A_{\widetilde{U}\widetilde{V}}\mathbf{1}_{\widetilde{V}} + {\mathbf{1}_{\widetilde{U}}}^T A_{\widetilde{U}\widetilde{U}}\mathbf{1}_{\widetilde{U}}.$$

We can show that ${\mathbf{1}_{\widetilde{U}}}^T A_{\widetilde{U}\widetilde{U}}\mathbf{1}_{\widetilde{U}}$ is at most $\frac{\varepsilon}{4}\ell$, where ℓ is a lower bound on $\mathbf{1}^T A\mathbf{1}$. Thus,

$$\mathbf{1}^T A\mathbf{1} \approx {\mathbf{1}_{\widetilde{V}}}^T A_{\widetilde{V}\widetilde{V}}\mathbf{1}_{\widetilde{V}} + {\mathbf{1}_{\widetilde{V}}}^T A_{\widetilde{V}\widetilde{U}}\mathbf{1}_{\widetilde{U}} + {\mathbf{1}_{\widetilde{U}}}^T A_{\widetilde{U}\widetilde{V}}\mathbf{1}_{\widetilde{V}}.$$

Lemma 4.6 shows that for a sufficiently large K, with high probability, the fraction of rows in any large bucket is approximately preserved in the sampled set of rows. Also observe that we know tight (upper and lower) bounds on $\langle A_{j*}, \mathbf{1}\rangle$ for every row j, where $j \in \widetilde{V}$. Thus, the random sample of S rows, such that $|S| = K$, approximately preserves ${\mathbf{1}_{\widetilde{V}}}^T A_{\widetilde{V}\widetilde{V}}\mathbf{1}_{\widetilde{V}} + {\mathbf{1}_{\widetilde{V}}}^T A_{\widetilde{V}\widetilde{U}}\mathbf{1}_{\widetilde{U}}$. Observe that this is already 2-approximation of $\mathbf{1^T A 1}$.

Using REGR for tight approximation

In order to get a $(1 \pm \varepsilon)$-approximation to $\mathbf{1}^T A \mathbf{1}$, we need to estimate $\mathbf{1}_{\widetilde{U}}^T A_{\widetilde{U}\widetilde{V}} \mathbf{1}_{\widetilde{V}}$, which is same as estimating $\mathbf{1}_{\widetilde{V}}^T A_{\widetilde{V}\widetilde{U}} \mathbf{1}_{\widetilde{U}}$ since A is a symmetric matrix. We estimate $\mathbf{1}_{\widetilde{V}}^T A_{\widetilde{V}\widetilde{U}} \mathbf{1}_{\widetilde{U}}$, that is, the sum of A_{ij}s such that $i \in \widetilde{V}$ and $j \in \widetilde{U}$, as follows. For each bucket B_i that is declared as large by the algorithm, we select *enough* number of rows randomly with replacement from $S_i = S \cap B_i$, invoke REGR for each selected row in S_i and increase the count by 1 if the element A_{ij} reported by REGR be such that $j \in \widetilde{U}$. A formal description of our algorithm is given in Algorithm 2. Now, we focus on the correctness proof of our algorithm for BFE.

Algorithm 2 BFE (ℓ, ε).

Input: An estimate ℓ for $\mathbf{1}^T A \mathbf{1}$ and $\varepsilon \in (0, 1/2)$.
Output: $\widehat{m}$, which is a $(1 \pm \varepsilon)$-approximation of $\mathbf{1}^T A \mathbf{1}$.
begin
Independently select $K = \Theta\left(\frac{\sqrt{\rho n}}{\sqrt{\ell}} \cdot \varepsilon^{-4.5} \cdot \log^2(\rho n) \cdot \log(1/\varepsilon)\right)$ rows of A uniformly at random and let S denote the multiset of the selected indices (of rows) sampled. For $i \in [t]$, let $S_i = B_i \cap S$.
Let $\widetilde{L} = \left\{i : \frac{|S_i|}{|S|} \geq \frac{1}{t} \cdot \frac{1}{n}\sqrt{\frac{\varepsilon}{6} \cdot \frac{\ell}{\rho}}\right\}$. Note that $\widetilde{L}$ is the set of buckets that the algorithm declares to be large. Similarly, $[t] \setminus \widetilde{L}$ is the set of buckets declared to be small by the algorithm.
For every $i \in \widetilde{L}$, select $|S_i|$ samples uniformly at random from S_i, with replacement, and let Z_i be the set of samples obtained. For each $z \in Z_i$, make a REGR$(\mathbf{1}, z)$ query and let $A_{zk_z} =$ REGR$(\mathbf{1}, z)$. Let Y_z be a random variable that takes value 1 if $k_z \in \widetilde{U}$ and 0, otherwise.
Determine $\widetilde{\alpha}_i = \frac{\sum_{z \in Z_i} Y_z}{|S_i|}$.
Output $\widehat{m} = \frac{n}{K} \sum_{i \in \widetilde{L}} (1 + \widetilde{\alpha}_i) \cdot |S_i| \cdot (1 + \beta)^i$.
end

To prove that $\widehat{m}$ is a $(1 \pm \varepsilon)$-approximation of $m = \mathbf{1}^T A \mathbf{1}$, we need the following definition and the technical Lemma 4.6.

▶ **Definition 4.5.** For $i \in L$, α_i is defined as $\frac{\sum_{u \in B_i} \langle A_{u*}, \mathbf{1}_{\widetilde{U}} \rangle}{\sum_{u \in B_i} \langle A_{u*}, \mathbf{1} \rangle}$.

▶ **Lemma 4.6.** *For a suitable choice of constant in $\Theta(\cdot)$ for selecting K samples in Algorithm 2, the followings hold with high probability:*

(i) *For each $i \in L$, we have $\frac{|S_i|}{K} = \left(1 \pm \frac{\varepsilon}{4}\right) \frac{|B_i|}{n}$.*

(ii) *For each $i \in [t] \setminus L$, $\frac{|S_i|}{K} < \frac{1}{t} \cdot \frac{1}{n}\sqrt{\frac{\varepsilon}{6} \cdot \frac{\ell}{\rho}}$.*

(iii) *We have $|\widetilde{U}| < \sqrt{\frac{\varepsilon}{4} \cdot \frac{\ell}{\rho}}$, where $\widetilde{U} = \{j \in B_i : i \in [t] \setminus \widetilde{L}\}$.*

(iv) *For every $i \in \widetilde{L}$,* **(a)** *if $\alpha_i \geq \frac{\varepsilon}{8}$, then $\widetilde{\alpha}_i = \left(1 \pm \frac{\varepsilon}{4}\right) \alpha_i$, and* **(b)** *if $\alpha_i < \varepsilon/8$, then $\widetilde{\alpha}_i < \varepsilon/4$.*

The above Lemma can be proved by using Chernoff bound (see Appendix A). Now, we have all the ingredients to show the following claim, which shows that $\widehat{m}$ is a $(1 \pm \varepsilon)$-approximation of $m = \mathbf{1}^T A \mathbf{1}$. The following claim is proved in the full version of the paper [13].

▷ Claim 4.7. With high probability, we have,
(i) $\widehat{m} \geq \left(1 - \frac{\varepsilon}{2}\right)\left(m - \frac{\varepsilon}{4}\ell\right)$, and
(ii) $\widehat{m} \leq \left(1 + \frac{3\varepsilon}{4}\right) m$, where $m = \mathbf{1}^T A \mathbf{1}$.

Recall that we have assumed $m/6 \leq \ell \leq m$. Under this assumption, the above claim says that $\widehat{m}$ is in fact a $(1 \pm \varepsilon)$-approximation to m. From the description of Algorithm 2, the number of IP queries made by the algorithm is $\widetilde{O}\left(\frac{\sqrt{\rho}n}{\sqrt{\ell}}\right) = \widetilde{O}\left(\frac{\sqrt{\rho}n}{\sqrt{\mathbf{1}^T A \mathbf{1}}}\right)$ as $m/10 \leq \ell \leq m$. We will discuss how to remove the assumption, that $m/6 \leq \ell \leq m$, by using a standard technique in property testing (see the full version of the paper [13]). So, we are done with the proof of Theorem 4.1.

4.2 Algorithm for Sampling Almost Uniformly

In this section, we will be proving the following theorem on almost uniformly sampling the entries of a symmetric matrix $A \in [[\rho]]^{n \times n}$. The algorithm for the special case is inspired by [21]. In the full version of the paper ([13]), we show how this algorithm can be extended to solve the more general $\textsc{Sau}_A(\mathbf{x}, \mathbf{y})$ problem.

▶ **Theorem 4.8.** *Let $A \in [[\rho]]^{n \times n}$ be an unknown symmetric matrix with* IP *query access. There exists an algorithm that takes $\varepsilon \in (0, 1)$ as input and with high probability outputs a sample from a distribution on $[n] \times [n]$, such that each $(i, j) \in [n] \times [n]$ is sampled with probability p_{ij} satisfying:*

$$p_{ij} = (1 \pm \varepsilon) \frac{A_{ij}}{\left(\sum_{1 \leq i,j \leq n} A_{ij}\right)}.$$

Moreover, the algorithm makes $\widetilde{\mathcal{O}}\left(\frac{\sqrt{\rho}n}{\sqrt{\mathbf{1}^T A \mathbf{1}}}\right)$ IP *queries to the matrix A of the form $\langle A_{k*}, \mathbf{u}\rangle$ for some $k \in [n]$ and $\mathbf{u} \in \{0, 1\}^n$.*

Our algorithm for Sample Almost Uniformly is a generalization of Eden and Rosenbaum's algorithm for *sampling edges of an unweighted graph* [21]. First, consider the following strategy by which we sample each ordered pair $(i, j) \in [n] \times [n]$ proportional to A_{ij} when the matrix A is such that $\langle A_{i,*}, \mathbf{1}\rangle$ is the same for each $i \in [n]$.
Strategy-1: Sample $r \in [n]$ uniformly at random and then sample an ordered pair of the form (r, j) from the r-th row using Regr query. Observe that this strategy fails when $\langle A_{i*}, \mathbf{1}\rangle$'s are not the same for every $i \in [n]$. So, the modified strategy is as follows.
Strategy-2: Sample $r \in [n]$ with probability $\frac{\langle A_{r*}, \mathbf{1}\rangle}{\mathbf{1}^T A \mathbf{1}}$ and then sample an ordered pair of the form (r, j) from the r-th row by using Regr query.

Note that Strategy-2 samples each ordered pair (i, j) proportional to A_{ij}. However, there are two challenges in executing Strategy-2:
(i) We do not know the value of $\mathbf{1}^T A \mathbf{1}$.
(ii) We need $\Omega(n)$ queries to determine $\langle A_{r*}, \mathbf{1}\rangle$ for each $r \in [n]$.
The first challenge can be taken care of by finding an estimate $\widehat{m}$ for $\mathbf{1}^T A \mathbf{1}$, with high probability, by using Theorem 4.1 such that $\widehat{m} = \Theta(\mathbf{1}^T A \mathbf{1})$. To cope up with the second challenge, we partition the elements as well as rows into two classes as defined in Definition 4.9. In what follows, we consider a parameter τ in terms of which we base our discussion as well as algorithm. τ is a function of $\widehat{m}$ that will evolve over the calculation and will be $\tau = \sqrt{\frac{\rho\widehat{m}}{\varepsilon}}$.

▶ **Definition 4.9.** The i-th row of the matrix is *light* if $\langle A_{i,*}, \mathbf{1}\rangle \leq \tau$, otherwise i-th row is *heavy*. Any order pair (i,j), for a fixed i, is light (heavy) if the i-th row is light (heavy).

We denote the set of all light (heavy) ordered pairs by $\mathcal{L}$ ($\mathcal{H}$). Also, let $I(L)$ ($I(H)$) denote the set of light (heavy) rows of the matrix A. Let $w(\mathcal{L}) = \sum_{A_{ij}\in\mathcal{L}} A_{ij}$ and $w(\mathcal{H}) = \sum_{A_{ij}\in\mathcal{H}} A_{ij}$.

Our algorithm consists of repeated invocation of two subroutines, that is, Sample-Light and Sample-Heavy. Both Sample-Light and Sample-Heavy succeed with *good* probability and sample elements from $\mathcal{L}$ and $\mathcal{H}$ almost uniformly, respectively. The threshold τ is set in such a way that there are *large* [8] number of light rows and small number of heavy rows. In Sample-Light, we select a row uniformly at random, and if the selected row is light, then we sample an ordered pair from the selected row randomly using Regr. This gives us an element from $\mathcal{L}$ uniformly. However, the same technique will not work for Sample-Heavy as we have few heavy rows. To cope up with this problem, we take a row uniformly at random and if the selected row is light, we sample an ordered pair from the selected row randomly using Regr. Let (i,j) be the output of the Regr query. Then we go to the j-th row, if it is heavy, and then select an ordered pair from the j-th row randomly using Regr query.

The formal algorithms for Sample-Light and Sample-Heavy are given in Algorithm 3 and Algorithm 4, respectively. The formal correctness of Sample-Light and Sample-Heavy are given in Lemmas 4.10 and 4.11, respectively. We give the final algorithm along with its proof of correctness in Theorem 4.8.

Algorithm 3 Sample-Light.

Input: An estimate $\widehat{m}$ for $\mathbf{1}^T A\mathbf{1}$ and a threshold τ.
Output: $(i,j) \in \mathcal{L}$ with probability $\frac{A_{ij}}{n\tau}$.
begin
  Select a row $r \in [n]$ uniformly at random.
  if *($r \in I(\mathcal{L})$, that is, $\langle A_{r*}, \mathbf{1}\rangle$ is at most τ)* **then**
    Return Fail with probability $p = \frac{\tau - \langle A_{r*}, \mathbf{1}\rangle}{\tau}$, and Return Regr$(r, \mathbf{1})$ with probability $1-p$ as the output.
  end
  Return Fail
end

▶ **Lemma 4.10.** Sample-Light *succeeds with probability* $\frac{w(\mathcal{L})}{n\tau}$. *Let* Z_ℓ *be the output in case it succeeds. Then* $\mathbb{P}(Z_\ell = (i,j)) = \frac{A_{ij}}{n\tau}$ *if* $(i,j) \in \mathcal{L}$, *and* $\mathbb{P}(Z_\ell = (i,j)) = 0$, *otherwise. Moreover,* Sample-Light *makes* $\mathcal{O}(\log n)$ *queries.*

▶ **Lemma 4.11.** Sample-Heavy *succeeds with probability at most* $\frac{w(\mathcal{H})}{n\tau}$ *and at least* $\left(1 - \frac{\rho\widehat{m}}{\tau^2}\right)\frac{w(\mathcal{H})}{n\tau}$. *Let* Z_h *be the output in case it succeeds. Then,* $\left(1 - \frac{\rho\widehat{m}}{\tau^2}\right)\frac{A_{ij}}{n\tau} \leq \mathbb{P}(Z_h = (i,j)) \leq \frac{A_{ij}}{n\tau}$ *for each* $(i,j) \in \mathcal{H}$, *and* $\mathbb{P}(Z_h = (i,j)) = 0$, *otherwise. Moreover,* Sample-Heavy *makes* $\mathcal{O}(\log n)$ *queries.*

Now we will us the above lemmas to prove Theorem 4.8.

[8] Large is parameterized by τ.

Algorithm 4 SAMPLE-HEAVY ($\widehat{m}$).

Input: An estimate $\widehat{m}$ for $\mathbf{1}^T A\mathbf{1}$ and a threshold τ.
Output: $A_{ij} \in \mathcal{H}$ with probability at most $\frac{A_{ij}}{n\tau}$ and at least $\left(1 - \frac{\rho\widehat{m}}{\tau^2}\right)\frac{A_{ij}}{n\tau}$.
begin
  Select a row $r \in [n]$ uniformly at random;
  if *($r \in I(\mathcal{L})$, that is, $\langle A_{r*}, \mathbf{1}\rangle$ is at most τ)* **then**
    Return FAIL with probability $p = \frac{\tau - \langle A_{r*}, \mathbf{1}\rangle}{\tau}$, and with probability $1-p$ do the following;
    $A_{rs} =$REGR$(r, \mathbf{1})$
    If $s \in I(\mathcal{H})$, that is, $\langle A_{s*}, \mathbf{1}\rangle > \tau$, then Return REGR$(s, \mathbf{1})$ as the output. Otherwise, Return FAIL;
  end
  Return FAIL;
end

Proof of Theorem 4.8. Our algorithm first finds a rough estimate $\widehat{m}$ for $\mathbf{1}^T A\mathbf{1}$, with high probability, by using Theorem 4.1 such that $\widehat{m} = \Theta(\mathbf{1}^T A\mathbf{1})$. For the rest of the proof, we work on the conditional probability space that $\widehat{m} = \Theta(\mathbf{1}^T A\mathbf{1})$. We set $\tau = \sqrt{\frac{\rho\widehat{m}}{\varepsilon}}$ and do the following for Γ times, where Γ is a parameter to be set later. With probability $1/2$, we invoke SAMPLE-LIGHT and with probability $1/2$, we invoke SAMPLE-HEAVY. If the ordered pair (i, j) is reported as the output by either SAMPLE-LIGHT or SAMPLE-HEAVY, we report that. If we get FAIL in all the trials, we report FAIL.

Now, let us consider a particular trial and compute the probability of success $\mathbb{P}(S)$, which is $\mathbb{P}(S) = \frac{1}{2}\left(\mathbb{P}(\text{SAMPLE-LIGHT succeeds}) + \mathbb{P}(\text{SAMPLE-HEAVY succeeds})\right)$. Observe that from Lemmas 4.10 and 4.11, we have,

$$\frac{1}{2}\left(\frac{w(\mathcal{L})}{n\tau} + \left(1 - \frac{\rho\widehat{m}}{\tau^2}\right)\frac{w(\mathcal{H})}{n\tau}\right) \leq \mathbb{P}(S) \leq \frac{1}{2}\left(\frac{w(\mathcal{L})}{n\tau} + \frac{w(\mathcal{H})}{n\tau}\right).$$

This implies $(1-\varepsilon)\frac{\mathbf{1}^T A\mathbf{1}}{2n\tau} \leq \mathbb{P}(S) \leq \frac{\mathbf{1}^T A\mathbf{1}}{2n\tau}$ as $\tau = \sqrt{\frac{\rho\widehat{m}}{\varepsilon}}$ and using $w(\mathcal{L}) + w(\mathcal{H}) = \mathbf{1}^T A\mathbf{1}$.

Now, let us compute the probability of the event $\mathcal{E}_{ij}$, that is, the algorithm succeeds and it returns A_{ij}. If $A_{ij} \in \mathcal{L}$, by Lemma 4.10, we have $\mathbb{P}(Z = (i,j)) = \frac{1}{2} \cdot \frac{A_{ij}}{n\tau}$. Also, if $A_{ij} \in \mathcal{H}$, by Lemma 4.11, we have, $\left(1 - \frac{\rho\widehat{m}}{\tau^2}\right)\frac{A_{ij}}{2n\tau} \leq \mathbb{P}(Z = (i,j)) \leq \frac{A_{ij}}{2n\tau}$. So, for any (i,j), we get $(1-\varepsilon)\frac{A_{ij}}{2n\tau} \leq \mathbb{P}(\mathcal{E}_{ij}) \leq \frac{A_{ij}}{2n\tau}$. Let us compute the probability of $\mathcal{E}_{ij}$ on the conditional probability space that the algorithm succeeds, that is, $\mathbb{P}(Z = (i,j) \mid S) = \frac{\mathbb{P}(\mathcal{E}_{ij})}{\mathbb{P}(S)}$, which lies in the interval $\left[(1-\varepsilon)\frac{A_{ij}}{\mathbf{1}^T A\mathbf{1}}, (1+\varepsilon)\frac{A_{ij}}{\mathbf{1}^T A\mathbf{1}}\right]$ as $\varepsilon \in \left(0, \frac{1}{2}\right)$.

To boost the probability of success, we set $\Gamma = \mathcal{O}\left(\frac{n\sqrt{\rho}}{(1-\varepsilon)\sqrt{\varepsilon\widehat{m}}}\log n\right)$ for a suitable *large* constant in $\mathcal{O}(\cdot)$ notation. The query complexity of each call to SAMPLE-LIGHT and SAMPLE-HEAVY is $\mathcal{O}(\log n)$. Also note that our algorithm for SAMPLE ALMOST UNIFORMLY makes at most $\mathcal{O}\left(\frac{n\sqrt{\rho}}{(1-\varepsilon)\sqrt{\varepsilon\widehat{m}}}\log n\right)$ invocations to SAMPLE-LIGHT and SAMPLE-HEAVY. Hence, the total query complexity of our algorithm is $\widetilde{\mathcal{O}}\left(\frac{\sqrt{\rho}n}{\sqrt{\mathbf{1}^T A\mathbf{1}}}\right)$. ◀

Table 1 Comparing IP with matrix-vector and vector-matrix-vector queries. Recall that MV anf VMV stand for matrix-vector and vector-matrix-vector queries.

Problem	IP **Query**	VMV **Query** [32]	MV **Query** [40]
Symmetric Matrix	$\widetilde{\Theta}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Diagonal Matrix	$\Theta(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Trace	$\Theta(n)$	$\mathcal{O}(n)$ $\Omega\left(\frac{n}{\log n}\right)$	$\mathcal{O}(n)$ $\Omega\left(\frac{n}{\log n}\right)$
Permutation Matrix	$\Theta(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Doubly Stochastic matrix	$\Theta(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Identical columns[9]	$\widetilde{\Theta}(n)$	$\widetilde{\Theta}(n)$	$\widetilde{\Theta}(n)$
All Ones Columns	$\Theta(n)$	$\mathcal{O}(n)$ $\Omega\left(\frac{n}{\log n}\right)$	$\mathcal{O}(n)$ $\Omega\left(\frac{n}{\log n}\right)$

5 Conclusion and discussions

Other matrix problems

Recently, vector-matrix query [40] and vector-matrix-vector query [32] were introduced to study a bunch of matrix, graph and statistics problems. As noted earlier, IP query oracle is in the same linear algebraic framework of vector-matrix (VM) query and vector-matrix-vector (VMV) query, but these queries are stronger than IP query. Study of the various matrix, graph and statistics problems, introduced in [40, 32], using IP query will be of independent interest. As a first step in that direction, in the full version of this paper [13], we study the query complexity of the following problems using IP queries.

- Symmetric Matrix: Is $A \in \{0,1\}^{n\times n}$ a symmetric matrix?
- Diagonal Matrix: Is A a diagonal matrix?
- Trace: Compute the trace of the matrix A.
- Permutation Matrix: Is $A \in \{0,1\}^{n\times n}$ a permutation matrix?
- Doubly stochastic matrix: Is $A \in \{0,1\}^{n\times n}$ a doubly stochastic matrix?
- Identical columns: Does there exist two columns in $A \in \{0,1\}^{n\times n}$ that are identical?
- All ones column: Does there exist a column in A all of whose entries are 1?

Table 1 compares the query complexities of IP oracle with matrix-vector (MV) and vector-matrix-vector (VMV) queries.

Data structure complexity and open problems

Besides property testing, there have been extensive work concerning vector-matrix-vector product in data structure complexity and other models of computation like the cell probe model [15, 16, 17, 28, 30]. For the purposes of this paper, it is an interesting question to find a pre-processing scheme for the matrix such the IP queries on the matrix can be answered efficiently.

[9] Upper and lower bounds results for Identical columns problem in [32, 40] uses vectors from $\{0,1\}^n$ in their respective queries.

One of the open problems that is left from our work is to design algorithm and/or prove lower bound for Bilinear Form Estimation and Sampling when the entries of the matrices are not necessarily positive. Here we would like to state that our technique does not work for a matrix with both positive and negative entries. Some other natural open questions are:

- Are there some special kind of matrices where we can solve Bilinear Form Estimation and Sampling using fewer queries?
- Can we solve some other linear algebraic problems using IP queries?
- Are there other graph problems, where induced degree outperforms local queries?

References

1 Intel C++ Compiler Classic Developer Guide and Reference: Floating Point Dot Product Intrinsics. `https://software.intel.com/content/www/us/en/develop/documentation/cpp-compiler-developer-guide-and-reference/top/compiler-reference/intrinsics/intrinsics-for-intel-streaming-simd-extensions-4-intel-sse4/vectorizing-compiler-and-media-accelerators/floating-point-dot-product-intrinsics.html`.

2 NVIDIA Developer: Cg 3.1 Toolkit Documentation. `https://developer.download.nvidia.com/cg/dot.html`.

3 K. J. Ahn, S. Guha, and A. McGregor. Analyzing Graph Structure via Linear Measurements. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 459–467, 2012.

4 M. Aliakbarpour, A. S. Biswas, T. Gouleakis, J. Peebles, R. Rubinfeld, and A. Yodpinyanee. Sublinear-Time Algorithms for Counting Star Subgraphs via Edge Sampling. *Algorithmica*, 80(2):668–697, 2018.

5 S. Assadi, D. Chakrabarty, and S. Khanna. Graph connectivity and single element recovery via linear and OR queries. In *European Symposium on Algorithms, ESA*, volume 204, pages 7:1–7:19, 2021.

6 S. Assadi, M. Kapralov, and S. Khanna. A Simple Sublinear-Time Algorithm for Counting Arbitrary Subgraphs via Edge Sampling. In *Innovations in Theoretical Computer Science Conference, ITCS*, pages 6:1–6:20, 2019.

7 M.-F. Balcan, Y. Li, D. P. Woodruff, and H. Zhang. Testing Matrix Rank, Optimally. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 727–746, 2019.

8 P. Beame, S. Har-Peled, S. N. Ramamoorthy, C. Rashtchian, and M. Sinha. Edge Estimation with Independent Set Oracles. In *Innovations in Theoretical Computer Science Conference, ITCS*, pages 38:1–38:21, 2018.

9 I. Ben-Eliezer, T. Kaufman, M. Krivelevich, and D. Ron. Comparing the strength of query types in property testing: the case of testing k-colorability. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1213–1222, 2008.

10 M. Benzi and C. Klymko. Total Communicability as a Centrality Measure. *Journal of Complex Networks*, 1:124–149, 2013.

11 A. Bishnu, A. Ghosh, S. Kolay, G. Mishra, and S. Saurabh. Parameterized Query Complexity of Hitting Set Using Stability of Sunflowers. In *International Symposium on Algorithms and Computation, ISAAC*, pages 25:1–25:12, 2018.

12 A. Bishnu, A. Ghosh, and G. Mishra. Distance Estimation Between Unknown Matrices Using Sublinear Projections on Hamming Cube. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, volume 207, pages 44:1–44:22, 2021.

13 A. Bishnu, A. Ghosh, G. Mishra, and M. Paraashar. Efficiently sampling and estimating from substructures using linear algebraic queries. *CoRR*, abs/1906.07398, 2019. Version 2. `arXiv:1906.07398v2`.

14 F. Bonchi, P. Esfandiar, D. F. Gleich, C. Greif, and L. V.S. Lakshmanan. Fast Matrix Computations for Pairwise and Columnwise Commute Times and Katz Scores. *Internet Mathematics*, 1-2(8):73–112, 2012.

15 D. Chakraborty, L. Kamma, and K. G. Larsen. Tight Cell Probe Bounds for Succinct Boolean Matrix-Vector Multiplication. In *ACM SIGACT Symposium on Theory of Computing, STOC*, pages 1297–1306, 2018.

16 A. Chattopadhyay, M. Koucký, B. Loff, and S. Mukhopadhyay. Simulation Beats Richness: New Data-Structure Lower Bounds. In *ACM SIGACT Symposium on Theory of Computing, STOC*, pages 1013–1020, 2018.

17 H. Dell, J. Lapinskas, and K. Meeks. Approximately Counting and Sampling Small Witnesses using a Colourful Decision Oracle. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2201–2211, 2020.

18 D. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 1st edition, 2009.

19 T. Eden, A. Levi, D. Ron, and C. Seshadhri. Approximately Counting Triangles in Sublinear Time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017.

20 T. Eden, D. Ron, and C. Seshadhri. On Approximating the Number of k-Cliques in Sublinear Time. In *ACM SIGACT Symposium on Theory of Computing, STOC*, pages 722–734, 2018.

21 T. Eden and W. Rosenbaum. On Sampling Edges Almost Uniformly. In *Symposium on Simplicity in Algorithms, SOSA*, pages 7:1–7:9, 2018.

22 E. Estrada and D. J. Higham. Network Properties Revealed through Matrix Functions. *SIAM Review*, 52:696–714, 2010.

23 U. Feige. On Sums of Independent Random Variables with Unbounded Variance and Estimating the Average Degree in a Graph. *SIAM Journal on Computing*, 35(4):964–984, 2006.

24 O. Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.

25 O. Goldreich and D. Ron. Approximating Average Parameters of Graphs. *Random Structures & Algorithms*, 32(4):473–493, 2008.

26 J. L. Hennessy and D. A. Patterson. *Computer Architecture – A Quantitative Approach (5. ed.)*. Morgan Kaufmann, 2012.

27 E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

28 K. G. Larsen and R. R. Williams. Faster Online Matrix-Vector Multiplication. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2182–2189, 2017.

29 N. Nisan. The demand query model for bipartite matching. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 592–599, 2021.

30 S. N. Ramamoorthy and C. Rashtchian. Equivalence of Systematic Linear Data Structures and Matrix Rigidity. In *Innovations in Theoretical Computer Science Conference, ITCS*, volume 151, pages 35:1–35:20, 2020.

31 A. Rao and A. Yehudayoff. *Communication Complexity: and Applications*. Cambridge University Press, 2020.

32 C. Rashtchian, D. P. Woodruff, and H. Zhu. Vector-Matrix-Vector Queries for Solving Linear Algebra, Statistics, and Graph Problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, volume 176, pages 26:1–26:20, 2020.

33 D. Ron and G. 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.

34 A. Rubinstein, T. Schramm, and S. M. Weinberg. Computing Exact Minimum Cuts Without Knowing the Graph. In *Innovations in Theoretical Computer Science Conference, ITCS*, pages 39:1–39:16, 2018.

35 J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley, Upper Saddle River, NJ, 2010.

36 X. Shi and D. P. Woodruff. Sublinear Time Numerical Linear Algebra for Structured Matrices. In *AAAI Conference on Artificial Intelligence, AAAI*, pages 727–746, 2019.

37 L. J. Stockmeyer. The Complexity of Approximate Counting (Preliminary Version). In *ACM SIGACT Symposium on Theory of Computing, STOC*, pages 118–126, 1983.

38 L. J. Stockmeyer. On Approximation Algorithms for #P. *SIAM Journal on Computing*, 14(4):849–861, 1985.

39 X. Sun, D. P. Woodruff, G. Yang, and J. Zhang. Querying a Matrix Through Matrix-Vector Products. In *International Colloquium on Automata, Languages, and Programming, ICALP*, pages 94:1–94:16, 2019.

40 X. Sun, D. P. Woodruff, G. Yang, and J. Zhang. Querying a Matrix Through Matrix-Vector Products. In *International Colloquium on Automata, Languages, and Programming, ICALP*, volume 132, pages 94:1–94:16, 2019.

A Useful probability bounds

▶ **Lemma A.1** (See [18]). *Let* $X = \sum_{i=1}^{n} X_i$ *where* X_i, $i \in [n]$, *are independent random variables,* $X_i \in [0,1]$ *and* $\mathbb{E}[X]$ *is the expected value of* X.

(i) *For* $\varepsilon > 0$, *we have* $\Pr\left[|X - \mathbb{E}[X]| > \varepsilon \mathbb{E}\left[X\right]\right] \leq \exp\left(-\varepsilon^2 \mathbb{E}[X]/3\right)$.

(ii) *Suppose* $\mu_L \leq \mathbb{E}[X] \leq \mu_H$. *Then, for all* $0 < \varepsilon < 1$, *we have*

(a) $\Pr[X > (1+\varepsilon)\mu_H] \leq \exp\left(-\varepsilon^2 \mu_H/3\right)$.

(b) $\Pr[X < (1-\varepsilon)\mu_L] \leq \exp\left(-\varepsilon^2 \mu_L/2\right)$.