Asymptotically Optimal Bounds for Estimating H-Index in Sublinear Time with Applications to Subgraph Counting

Sepehr Assadi ✉ 🏠
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Hoai-An Nguyen ✉ 🏠
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Abstract

The *h-index* is a metric used to measure the impact of a user in a publication setting, such as a member of a social network with many highly liked posts or a researcher in an academic domain with many highly cited publications. Specifically, the h-index of a user is the largest integer h such that at least h publications of the user have at least h units of positive feedback.

We design an algorithm that, given query access to the n publications of a user and each publication's corresponding positive feedback number, outputs a $(1 \pm \varepsilon)$-approximation of the h-index of this user with probability at least $1 - \delta$ in time $O\left(\frac{n \cdot \ln(1/\delta)}{\varepsilon^2 \cdot h}\right)$, where h is the actual h-index which is unknown to the algorithm a-priori. We then design a novel lower bound technique that allows us to prove that this bound is in fact **asymptotically optimal** for this problem in **all parameters** n, h, ε, and δ.

Our work is one of the first in sublinear time algorithms that addresses obtaining asymptotically optimal bounds, especially in terms of the error and confidence parameters. As such, we focus on designing novel techniques for this task. In particular, our lower bound technique seems quite general – to showcase this, we also use our approach to prove an asymptotically optimal lower bound for the problem of estimating the number of triangles in a graph in sublinear time, which now is also optimal in the error and confidence parameters. This latter result improves upon prior lower bounds of Eden, Levi, Ron, and Seshadhri (FOCS'15) for this problem, as well as multiple follow-up works that extended this lower bound to other subgraph counting problems.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Sublinear time algorithms, h-index, asymptotically optimal bounds, lower bounds, subgraph counting

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2022.48

Category APPROX

Funding *Sepehr Assadi*: Department of Computer Science, Rutgers University. Research supported in part by a NSF CAREER Grant CCF-2047061, a gift from Google Research, and a Fulcrum award from Rutgers Research Council.
Hoai-An Nguyen: Research supported in part by a NSF CAREER Grant CCF-2047061.

Acknowledgements We thank Janani Sundaresan for helpful feedback on the presentation of our paper. We are also grateful to the anonymous reviewers of APPROX 2022 for their helpful feedback on previous work and the presentation of this paper.

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2022).
Editors: Amit Chakrabarti and Chaitanya Swamy; Article No. 48; pp. 48:1–48:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The *Hirsch* index, or h-index for short, is a metric used to measure the impact of a researcher's publications [20]. It is an integer that considers both the number of publications and citations a researcher has and is used in a number of contexts including consideration for grants and job opportunities. We can abstract out this problem by modeling each individual researcher as an array $A[1:n]$ where n is the number of papers they have published and $A[i]$ is the number of citations paper $i \in [n]$ has. The h-index of A is then defined as follows.

▶ **Definition 1.** *The* **h-index** *of an array* $A[1:n]$, *denoted by* $\mathsf{h}(A)$, *is the* *maximum* *integer* h *such that* $A[1:n]$ *has at least* h *indices,* i_j, *where for each* $j \in [h], A[i_j] \geqslant h$.

There are simple algorithms that can compute the value of $\mathsf{h}(A)$ for any given array A in $O(n)$ time. For instance, we can change each entry of $A[i]$ to $\min\{A[i], n\}$ without changing $\mathsf{h}(A)$ (since $\mathsf{h}(A) \leqslant n$) and then run counting sort on A in linear time to sort A in decreasing order. We can then make another pass over A and output the largest index $i \in [n]$ such that $A[i] \geqslant i$ which will be equal to $\mathsf{h}(A)$ now that A is sorted. This solves the h-index problem in $\Theta(n)$ time.

The question we focus on in this paper is whether we can solve this problem even faster than reading the entire input, namely, via a *sublinear time* algorithm, assuming we can read each single entry of A in $O(1)$ time. There are easy observations that show that the answer to this question is *No* without relaxing the problem: deterministic algorithms cannot solve this problem in sublinear time even approximately, and randomized algorithms cannot find an exact answer[1]. Such observations however are commonplace when it comes to sublinear time algorithms. Our goal in this paper is thus to solve this problem allowing both randomization and approximation.

▶ **Result 2.** *There is an algorithm that for any array* A *and any* $\varepsilon, \delta \in (0,1)$, *with probability at least* $1-\delta$, *outputs an estimate* $\tilde{h}$ *such that* $|\tilde{h} - \mathsf{h}(A)| \leqslant \varepsilon \cdot \mathsf{h}(A)$ *in* $O(\frac{n \cdot \ln(1/\delta)}{\varepsilon^2 \cdot \mathsf{h}(A)})$ *time. Moreover, we prove that this algorithm is asymptotically optimal in* *all* *parameters involved.*

Result 2 gives a randomized sublinear time algorithm for a $(1 \pm \varepsilon)$-approximation of the h-index problem, where the runtime improves depending on the value of the h-index itself. This is quite common in sublinear time algorithms; see, e.g. [9, 11, 1] for estimating the number of subgraphs, [4] for minimum cut, or [14, 15, 10, 31] for sampling small subgraphs, among others. In all the aforementioned examples, such dependences are necessary, which is also the case for ours by the lower bound we prove.

Our Result 2, however, is quite novel from a different perspective: the obtained bounds are asymptotically optimal in *all* the parameters of the problem, including ε and δ. We are not aware of any prior work with such strong guarantees as we will discuss in more detail in the next subsection. Moreover, as a corollary of our techniques in proving the lower bound

[1] A deterministic algorithm running in $o(n)$ time cannot distinguish between an array A which is all zeros and an array B obtained from A by making $n/2$ entries have value $n/2$ instead. This is because the first $n/2$ queries of the algorithm to indices of A or B can be 0 in both cases. Yet, we have $\mathsf{h}(A) = 0$ and $\mathsf{h}(B) = n/2$. Similarly, a randomized algorithm running in $o(n)$ time cannot distinguish between an array A with value n as every entry and an array B obtained from A by changing exactly one of the entries to $n-1$ instead. This can be proven for instance by using the $\Omega(n)$ lower bound on the query complexity of the OR problem [6]. In this case $\mathsf{h}(A) = n$ and $\mathsf{h}(B) = n-1$.

for Result 2 with dependence on both ε and δ, we also obtain an asymptotically optimal lower bound for the well-studied problem of counting triangles in sublinear time that now matches the dependence on ε and δ as well, improving upon the prior work in [9, 13, 1].

1.1 Key Motivations

There are two key, yet disjoint, motivations behind our work that we elaborate on below.

Measuring "impact" quickly

Consider any "publication setting" that allows for user feedback. This can range from social networks with users posting topics and others liking them all the way to the academic domain with researchers publishing papers and others citing them. A question studied frequently in social sciences is how to measure the "impact" of a single user in such a setting for many different contexts, including identifying impactful users for marketing or propagating information; see, e.g. [28] and the references therein.

One of the well-accepted measures of impact in these publication settings is the h-index measure we study in this paper [20, 28]. Given the ubiquity of massive publication settings and their evolving nature, say, social networks, we need algorithms that are able to compute the h-index of different users efficiently; see, e.g. [18] that design such algorithms in the closely related *streaming* model (which focuses on the space usage of algorithms instead of their time). Thus, a key motivation behind our Result 2 is to provide a time-efficient algorithm for this purpose. In general, it seems like a fascinating area of research to obtain efficient algorithms for measuring various notions of impact in these massive publication settings in parallel to the line of work, e.g., in [28], that searches for the "right" measure itself.

In particular, the h-index has numerous applications within network science. In [8], it is shown that when the h-index of a graph is large enough, the algorithm they design to approximate the degree distribution is sublinear. In [24], the focus is on computing coreness through iteratively using an operator that can calculate the h-index of any node to identify influential nodes: an important step in understanding a network's dynamics and structure. Both works do not specify how their algorithm computes the h-index, so the use of our algorithm could help prevent impractical runtimes. Building on [24], [29] generalizes using an iterative h-index operator for truss and nucleus decomposition to find dense subgraphs. They use the classical linear algorithm for calculating the h-index, which therefore leaves the opportunity to use our algorithm to achieve better efficiency.

Asymptotically optimal sublinear time algorithms

Traditionally, the work on sublinear time algorithms have been rather cavalier with the dependence on the error parameter ε, confidence parameter δ, and logarithmic factors. It is certainly important to focus on the "high order terms" in the complexity of problems, say, in numerous works on subgraph counting; see, e.g., [9, 11, 12] and references therein. However, as already observed in [17]: "the dependence of the complexity on the approximation parameter is a key issue". For instance, in any $(1 \pm \varepsilon)$-approximation algorithm, for a typical value of $\varepsilon \sim 1\%$, one extra factor of $1/\varepsilon$ in the runtime translates to roughly a $100x$ slower algorithm, which is almost always a deal breaker for the practical purposes of sublinear time algorithms! Similar considerations also apply, but perhaps to a lower extent, to having a large dependence on logarithmic factors instead of asymptotically optimal bounds. In terms of the confidence parameter, δ, the runtime dependence of sublinear time algorithms almost always includes the term $\ln(1/\delta)$. It is important for practical considerations to determine whether this dependence is necessary.

Despite this, such considerations have not been studied in sublinear time algorithms. The only prior work we are aware of is the very recent work of [31] that improved the $O(\varepsilon^{-1/2})$-dependence of the algorithm of [14] for sampling edges ε-point-wise close to uniform to an $O(\log(1/\varepsilon))$-dependence. This is in stark contrast with the large body of work in related areas such as streaming [21, 23, 5], graph streaming [25, 2], compressed sensing [26, 27], sampling [22], and dynamic graph algorithms [30, 19, 3] which put emphasis on obtaining asymptotically optimal algorithms and lower bounds on all parameters.

In light of this discussion, another key motivation of our work has been to use the h-index problem as a *medium* for designing general techniques for obtaining asymptotic bounds for sublinear time algorithms in general. For instance, our algorithm involves careful subroutines that side-step typical "binary search" approaches in prior work that results in additional $O(\varepsilon^{-1} \cdot \log n)$ terms in the runtimes of algorithms and a more careful analysis of the error that bypasses a trivial union bound which leads to additional $O(\log n)$ factors. More importantly, we design a new lower bound technique, based on a new query complexity result that we establish, that allows us to prove lower bounds that depend on both parameters ε and δ. This approach can now be used to replace prior sublinear time lower bounds both based on ad-hoc arguments such as the ones in [9] or the ones based on communication complexity [14, 1]. As a result, we also obtain asymptotically optimal lower bounds for the problem of counting triangles in a graph that now matches the dependence on ε and δ as well, improving upon the prior work in [9, 13, 1].

1.2 Notation

For any integer $t \geqslant 1$, we define $[t] := \{1, 2, \ldots, t\}$. For any $p \in (0, 1)$, we use $\mathcal{B}(p)$ to denote the *Bernoulli* distribution with mean p. For a set S of integers, we write $i \in_R S$ to mean i is chosen uniformly at random from S.

1.3 Appendix

Due to space limitations, some details and proofs marked by a star are postponed to the full version of the paper which appears on arXiv. Appendix A includes the concentration results, other basic probabilistic tools, basic definitions and tools from query complexity, and measures of distance between distributions that we use in this paper.

2 The Algorithm

We describe our main algorithm for the h-index problem in this section.

▶ **Theorem 3.** *There exists a sublinear time algorithm that given query access to an integer array $A[1:n]$, approximation and confidence parameters $\varepsilon, \delta \in (0, 1)$, with probability at least $1 - \delta$ outputs an estimate $\tilde{h}$ of $\mathsf{h}(A)$ such that $|\tilde{h} - \mathsf{h}(A)| \leqslant \varepsilon \cdot \mathsf{h}(A)$ in $O(\frac{n \cdot \ln(1/\delta)}{\varepsilon^2 \cdot \mathsf{h}(A)})$ time.*

The algorithm in Theorem 3 is a combination of a "weak" and "strong" estimator that we design. The weak estimator only outputs whether $\mathsf{h}(A)$ is at least as large as a given threshold, but it is efficient and can be used to provide a lower bound on $\mathsf{h}(A)$. The strong estimator, which has a slower runtime, then uses the lower bound to output an estimate of $\mathsf{h}(A)$. In the next two subsections, we present these two estimators and then conclude the proof of Theorem 3 through a careful combination of them that preserves the asymptotic runtime of the overall algorithm.

2.1 A Weak Estimator

We present an algorithm that determines with high probability whether $\mathsf{h}(A)$ is at least as large as a given threshold.

▶ **Lemma 4.** *There exists a sublinear time algorithm that given query access to an integer array $A[1:n]$ and an integer $T \geqslant 1$ in $O(n/T)$ time outputs an answer satisfying the following:*

(i) *if $\mathsf{h}(A) \geqslant T$, the answer is* Large *with probability at least $1 - 1/16$;*

(ii) *if $\mathsf{h}(A) < T/4$, the answer is* Small *with probability at least $1 - \mathsf{h}(A)/(4T)$;*

(iii) *either* Small *or* Large *can be outputted in the remaining cases.*

Let us point out the asymmetric guarantee of the algorithm: it does not underestimate $\mathsf{h}(A)$ with a certain constant probability while it does not overestimate $\mathsf{h}(A)$ with probability proportional to the "rate" of overestimation. This guarantee will be crucial in our final algorithm. We also note that the guarantee on the runtime of the algorithm is deterministic.

2.1.1 The Algorithm

At a high level, our algorithm, `h-index-weak-estimator`, queries random indices from A and calculates the proportion of those indices that are above a threshold representing the mid-point between a h-index of $T/4$ and T. If the proportion is below the threshold, the algorithm outputs *Small*; otherwise, it outputs *Large*.

Algorithm 1 h-index-weak-estimator($A[1:n]$, T).

```
Sample $k := 64 \cdot n/T$ indices $S$ independently and uniformly with repetition from $[n]$.
Let $X$ denote the number of indices $i \in S$ such that $A[i] \geqslant T$.
If $X \geqslant kT/(2n)$, output Large. Otherwise, output Small.
```

The runtime of `h-index-weak-estimator` is simply $O(n/T)$ as we are sampling these many indices in S and then for each $i \in S$, we need to query $A[i]$; counting the value of X and outputting the answer can also be done in $O(n/T)$ time, which bounds the runtime as desired.

2.1.2 The Analysis

We now analyze the correctness of the algorithm. For any $j \in [k]$, define an indicator random variable X_j which is 1 iff the j-th sample in S, namely, $i_j \in [n]$, satisfies $A[i_j] \geqslant T$. This way, for the counter X in the algorithm, we have $X = \sum_{j=1}^{k} X_j$. Recall that the output of the algorithm depends on the value of X. In the following, we will separately consider the value of X in the case when the output is supposed to be *Large* versus when it is supposed to be *Small*.

Case I: the "Large" case

We first consider the case when the output should be *Large*, or when $\mathsf{h}(A) \geqslant T$. Thus,

$$\mathbb{E}\left[X\right] = \sum_{j=1}^{k} \mathbb{E}\left[X_j\right] = \sum_{j=1}^{k} \Pr_{i_j \in_R [n]}\left(A[i_j] \geqslant T\right) \geqslant k \cdot \frac{T}{n}, \tag{1}$$

since A consists of at least T indices with value $\geqslant T$ when $\mathsf{h}(A) \geqslant T$, and we are sampling indices $i_j \in [n]$ for $j \in [k]$ uniformly at random. We can similarly bound the variance of X using Fact 29 since variables X_j for $j \in [k]$ are independent, and thus,

$$\mathrm{Var}\left[X\right] = \mathrm{Var}\left[\sum_{j=1}^{k} X_j\right] = \sum_{j=1}^{k} \mathrm{Var}\left[X_j\right] \leqslant \sum_{j=1}^{k} \mathbb{E}\left[X_j^2\right] = \sum_{j=1}^{k} \mathbb{E}\left[X_j\right] = \mathbb{E}\left[X\right], \tag{2}$$

where the second to last equality is because for all $j \in [k]$, X_j is an indicator random variable.

We use Chebyshev's inequality (Proposition 30) to finalize the proof of this case.

▷ Claim 5 (⋆). When $\mathsf{h}(A) \geqslant T$, we have $\Pr\left(\text{algorithm outputs } \textit{Small}\right) \leqslant 1/16$.

This claim is now enough to establish property (*i*) in Lemma 4.

Case II: the "Small" case

We now consider the case when the output should be *Small*, namely, when $\mathsf{h}(A) < T/4$. In this case, we have,

$$\mathbb{E}\left[X\right] = \sum_{j=1}^{k} \mathbb{E}\left[X_j\right] = \sum_{j=1}^{k} \Pr_{i_j \in_R [n]}\left(A[i_j] \geqslant T\right) < k \cdot \frac{T}{4n}, \tag{3}$$

as there are less than $T/4$ indices in A with value $\geqslant T$ when $\mathsf{h}(A) < T/4$, and we are sampling indices $i_j \in [n]$ for $j \in [k]$ uniformly at random. We will also bound the variance of X similarly to Equation (2) but in a slightly more careful manner. By Fact 29, since variables X_j for $j \in [k]$ are independent, we have,

$$\mathrm{Var}\left[X\right] = \sum_{j=1}^{k} \mathrm{Var}\left[X_j\right] \leqslant \sum_{j=1}^{k} \mathbb{E}\left[X_j\right] = \sum_{j=1}^{k} \Pr_{i_j \in_R [n]}\left(A[i_j] \geqslant T\right) \leqslant k \cdot \frac{\mathsf{h}(A)}{n}, \tag{4}$$

where in the last inequality, we use the fact that the number of indices in A with value larger than T is at most $\mathsf{h}(A)$ (since we already know that $\mathsf{h}(A) < T$).

To conclude the proof, we again use Chebyshev's inequality but with a slightly different analysis.

▷ Claim 6 (⋆). When $\mathsf{h}(A) < T/4$, we have $\Pr\left(\text{algorithm outputs } \textit{Large}\right) \leqslant \mathsf{h}(A)/(4T)$.

Lemma 4 now follows from the previous two claims.

2.2 A Strong Estimator

We now present our second intermediate algorithm which outputs an estimate of $\mathsf{h}(A)$ when given the guarantee that $\mathsf{h}(A)$ is at least as large as a given threshold.

▶ **Lemma 7.** *There exists a sublinear time algorithm that given query access to an integer array $A[1:n]$, an integer $T \leqslant \mathsf{h}(A)$, and approximation parameter $\varepsilon \in (0,1)$, in $O(n/(\varepsilon^2 T))$ time outputs an estimate $\tilde{h}$ of $\mathsf{h}(A)$ such that $\Pr(|\tilde{h} - \mathsf{h}(A)| \leqslant \varepsilon \cdot \mathsf{h}(A)) \geqslant 2/3$.*

The guarantee on the runtime of the algorithm holds deterministically even when $T > \mathsf{h}(A)$.

We emphasize that while the guarantee on the runtime of the algorithm in Lemma 7 holds even when $T > \mathsf{h}(A)$, we clearly have no guarantee on the correctness in this case.

Algorithm 2 h-index-strong-estimator($A[1:n]$, T, ε).

```
Sample $k := 6n/(\varepsilon^2 T)$ indices $S$ independently and uniformly with repetition from $[n]$.
Let $B[1:k]$ be an array consisting of integers $A[i]$ for $i \in S$.
Return[2] the largest integer $q \in [n]$ such that $k \cdot q/n$ indices in $B$ are at least $q$.
```

2.2.1 The Algorithm

The algorithm, `h-index-strong-estimator`, queries a set of random indices from A and finds a scaled estimate of the h-index.

The first two lines of `h-index-strong-estimator` can be implemented in $O(k) = O(n/(\varepsilon^2 T))$ time in a straightforward way. We show that the last step can also be implemented in $O(k)$ time.

▶ **Lemma 8** (⋆). `h-index-strong-estimator` *runs in* $O(n/(\varepsilon^2 T))$ *time.*

2.2.2 The Analysis

We prove the correctness of `h-index-strong-estimator` in this subsection. We consider each case in which the algorithm may overestimate or underestimate $\mathsf{h}(A)$ separately.

Probability of overestimation

We first bound the probability that $\tilde{h} > (1+\varepsilon) \cdot \mathsf{h}(A)$. For this event to happen, we need B to have more than $(k/n) \cdot (1+\varepsilon) \cdot \mathsf{h}(A)$ indices with a value greater than $(1+\varepsilon) \cdot \mathsf{h}(A)$. We bound the probability of this happening in the following.

For any $j \in [k]$, define an indicator random variable X_j which is 1 iff the j-th sample $i_j \in S$ satisfies $A[i_j] > (1+\varepsilon) \cdot \mathsf{h}(A)$. Define $X = \sum_{j=1}^{k} X_j$. By the above discussion,

$$\Pr\left(\tilde{h} > (1+\varepsilon) \cdot \mathsf{h}(A)\right) = \Pr(X > (k/n) \cdot (1+\varepsilon) \cdot \mathsf{h}(A)). \tag{5}$$

We bound the probability of the RHS of this equation.

▷ Claim 9 (⋆). $\Pr\left(X > (k/n) \cdot (1+\varepsilon) \cdot \mathsf{h}(A)\right) < 1/6$.

Probability of underestimation

We now bound the probability that $\tilde{h} < (1-\varepsilon) \cdot \mathsf{h}(A)$. This case is essentially symmetric to the other one and is provided for completeness. For this event to happen, we need B to have less than $(k/n) \cdot (1-\varepsilon) \cdot \mathsf{h}(A)$ indices with a value of at least $(1-\varepsilon) \cdot \mathsf{h}(A)$. We bound the probability of this happening in the following.

For any $j \in [k]$, define an indicator random variable Y_j which is 1 iff the j-th sample $i_j \in S$ satisfies $A[i_j] \geqslant (1-\varepsilon) \cdot \mathsf{h}(A)$. Define $Y = \sum_{j=1}^{k} Y_j$. By the above discussion,

$$\Pr\left(\tilde{h} < (1-\varepsilon) \cdot \mathsf{h}(A)\right) = \Pr\left(Y < (k/n) \cdot (1-\varepsilon) \cdot \mathsf{h}(A)\right). \tag{6}$$

We bound the probability of the RHS of this equation.

▷ Claim 10 (⋆). $\Pr\left(Y < (k/n) \cdot (1-\varepsilon) \cdot \mathsf{h}(A)\right) < 1/6$.

Combining Claim 9 and Claim 10 concludes the proof of Lemma 7.

2.3 The Sublinear Time h-Index-Estimator Algorithm

We now combine our weak and strong estimators to obtain a sublinear time algorithm for estimating the h-index and prove Theorem 3. The algorithm runs `h-index-weak-estimator` on smaller and smaller thresholds to determine a threshold that tightly lower bounds $\mathsf{h}(A)$. Then, `h-index-strong-estimator` uses that threshold to output an estimate of $\mathsf{h}(A)$. Finally, to ensure a probability of success of at least $1 - \delta$, we combine the median/majority trick in a rather non-black-box way using the asymmetric guarantee of `h-index-weak-estimator` in part (ii) of Lemma 4.

Algorithm 3 h-index-estimator($A[1:n]$, ε, δ).

```
1 Let $r_1 := 7\ln(8/\delta)$ and $r_2 := 108\ln(8/\delta)$ and initialize $T$ to $n$.
2 While the majority answer of running h-index-weak-estimator($A$, $T$) $r_1$ times
    returns Small, update $T \leftarrow T/4$.
3 For the current value of $T$, run h-index-strong-estimator($A$, $T/16$, $\varepsilon$) $r_2$ times
    and return the median answer as the final estimate $\tilde{h}$.
```

We bound the runtime of the algorithm in the following lemma.

▶ **Lemma 11.** `h-index-estimator` *runs in* $O\Big(\dfrac{n \cdot \ln(1/\delta)}{\varepsilon^2 \cdot \mathsf{h}(A)}\Big)$ *time with probability* $1 - \delta/2$.

Proof. The runtime depends on both running `h-index-weak-estimator` on (potentially) multiple thresholds and running `h-index-strong-estimator`.

We define $\boldsymbol{T^*}$ as the "optimal" threshold: the *first* threshold given to `h-index-weak-estimator` that is not larger than $\mathsf{h}(A)$, namely, $T^* \leqslant \mathsf{h}(A) < 4 \cdot T^*$. The following claim bounds the probability that the while-loop in step two of `h-index-estimator` does not stop even after iteration T^*.

▷ Claim 12 (⋆). Pr (`h-index-estimator` continues its while-loop beyond T^*) $\leqslant \delta/2$.

In the following, we condition on the complement of the event in Claim 12 which happens with probability at least $1 - \delta/2$, which means we have only run the while-loop until at most iteration T^*. Let $T_0 = n, T_1 = n/4, \ldots, T_t = n/4^t = T^*$ denote the thresholds in these iterations. By Lemma 4 on the runtime of `h-index-weak-estimator` we have,

$$\text{runtime of while-loop} = \sum_{j=0}^{t} O(\frac{n}{T_j}) \cdot O(\ln{(1/\delta)}) = O\left(\frac{n}{T^*} \cdot \ln{(1/\delta)}\right) \cdot \sum_{j=0}^{t} \frac{1}{4^j}$$

$$= O\left(\frac{n}{\mathsf{h}(A)} \cdot \ln{(1/\delta)}\right),$$

since T^* is a 4-approximation to $\mathsf{h}(A)$ by definition and the given geometric series converges.

Moreover, by Lemma 7 on the runtime of `h-index-strong-estimator`, in this case, we have that the last line of the algorithm takes $O(\frac{n \cdot \ln{(1/\delta)}}{\varepsilon^2 \cdot T^*}) = O(\frac{n \cdot \ln{(1/\delta)}}{\varepsilon^2 \cdot \mathsf{h}(A)})$ time as well, again since T^* is a 4-approximation to $\mathsf{h}(A)$ (computing the medians can be done with the Median-of-Medians algorithm in $O(r_2)$ time which is negligible in the above bounds).

All in all, we have that with probability $1 - \delta/2$, the algorithm runs in $O(\frac{n \cdot \ln{(1/\delta)}}{\varepsilon^2 \cdot \mathsf{h}(A)})$ time. ◀

2.3.1 The Analysis

We prove the correctness of our algorithm in this subsection. Consider the parameter T^* defined earlier as the "optimal" threshold in the while-loop, meaning that $T^* \leqslant \mathsf{h}(A) < 4 \cdot T^*$. There are two potential sources for error:

1. **Event $\boldsymbol{\mathcal{E}_{weak}}$**: In the while-loop, `h-index-weak-estimator` outputs *Large* for an iteration $T > 16T^*$; assuming this happens, the threshold passed to `h-index-strong-estimator` is not necessarily valid, meaning that it may not be a lower bound on $\mathsf{h}(A)$.
2. **Event $\boldsymbol{\mathcal{E}_{strong}}$**: The threshold T obtained by the runs of `h-index-weak-estimator` in the while-loop satisfies $T \leqslant 16T^*$ and thus is valid, but `h-index-strong-estimator` nevertheless fails to output an accurate estimate of $\mathsf{h}(A)$.

Among these, the probability of the second event is quite easy to bound using Lemma 7. Thus, in the following, we focus primarily on proving the first part.

▷ Claim 13 (⋆). In `h-index-estimator`, for any $T = 4^{\ell} \cdot T^*$ for an integer $\ell \geqslant 2$, $\Pr\left(\text{the while-loop terminates at iteration } T\right) \leqslant (\delta/8)^{\ell-1}$.

We can now bound the error probability due to event $\mathcal{E}_{weak}$. We have,

$$\begin{aligned}\Pr\left(\mathcal{E}_{weak}\right) &\leqslant \sum_{\ell \geqslant 2} \Pr\left(\text{the while-loop terminates at } T = 4^{\ell} \cdot T^*\right) \\ &\leqslant \sum_{\ell \geqslant 2} \left(\frac{\delta}{8}\right)^{\ell-1} && \text{(by Claim 13)} \\ &= \frac{(\delta/8)}{1-(\delta/8)} < \frac{\delta}{4}. && \text{(as } \textstyle\sum_{j=1}^{\infty} x^j = \frac{x}{1-x} \text{ for } x \in (0,1))\end{aligned}$$

We now bound the other source of error. Assuming $\mathcal{E}_{weak}$ does not happen, for the parameter T that the while-loop terminates on, we have $T \leqslant 16T^* \leqslant 16\mathsf{h}(A)$ by the definition of T^*. This implies that the parameter $T/16$ passed to `h-index-strong-estimator` is a lower bound on $\mathsf{h}(A)$. Thus, by Lemma 7, each of the r_2 runs of `h-index-strong-estimator` outputs a $(1 \pm \varepsilon)$-approximation to $\mathsf{h}(A)$ with probability at least $2/3$.

▷ Claim 14 (⋆). $\Pr\left(\mathcal{E}_{strong} \mid \overline{\mathcal{E}_{weak}}\right) \leqslant \delta/4$.

Therefore, by the union bound, the total probability of error is at most $\delta/4 + \delta/4 = \delta/2$. This concludes the analysis of `h-index-estimator`.

3 The Lower Bound

We now prove the asymptotic optimality of the bounds obtained by our algorithm in Theorem 3.

▶ **Theorem 15.** *Any algorithm that, given query access to an array $A[1:n]$, approximation parameter $\varepsilon \in (0, 1/4)$, and confidence parameter $\delta \in (0, 1/100)$, with probability $1-\delta$ uses at most q queries and outputs an estimate $\tilde{h}$ such that $|\tilde{h} - \mathsf{h}(A)| \leqslant \varepsilon \cdot \mathsf{h}(A)$ needs to satisfy $q = \Omega(\min(n, \frac{n \cdot \ln(1/\delta)}{\varepsilon^2 \cdot \mathsf{h}(A)}))$.*

To prove Theorem 15, we define a new problem which we call the *Popcount Thresholding Problem (PTP)* and prove a lower bound on its randomized query complexity. We will then perform a reduction from this problem to establish our theorem.

▶ Remark 16. Let us suppose that $100 < \mathsf{h}(A) < \ln(1/\delta) \cdot 12/\varepsilon^2$. There exists some $\varepsilon' > \varepsilon$ and $\delta' > \delta$ such that $\mathsf{h}(A) = \ln(1/\delta') \cdot 12/\varepsilon'^2$, and therefore, $(n \cdot \ln(1/\delta'))/(\varepsilon'^2 \cdot \mathsf{h}(A)) = \Omega(n)$. So, the lower bound in Theorem 15 of $\Omega(n)$ given the above promises on the value of $\mathsf{h}(A)$ is arbitrarily proven. In the following, we focus on proving that when $\mathsf{h}(A) \geqslant \ln(1/\delta) \cdot 12/\varepsilon^2$, the randomized query complexity is $\Omega((n \cdot \ln(1/\delta))/(\varepsilon^2 \cdot \mathsf{h}(A)))$.

In passing, we note that *PTP* seems quite a natural and general problem of its own independent interest; we will also use this problem in the subsequent section to prove asymptotically optimal lower bounds for the well-studied problem of estimating the number of triangles in a graph in sublinear time.

3.1 Popcount Thresholding Problem (PTP)

We define the Popcount Thresholding Problem as follows.

▶ **Problem 17.** *In $PTP_{m,k,\gamma}$, for integers $m, k, \geqslant 1$ and parameter $\gamma \in (0, 1)$, we are given a string $x \in \{0, 1\}^m$ sampled with equal probability from either D_0 where for each index $i \in [m]$, x_i is independently set to 1 with probability $p_0 := (1 - 2\gamma) \cdot k/m$ or D_1 where for each index $i \in [m]$, x_i is independently set to 1 with probability $p_1 := (1 + 2\gamma) \cdot k/m$. The answer is* Yes *if x was drawn from D_1, and it is* No *if x was drawn from D_0.*

We prove the following lemma on the query complexity of *PTP*.

▶ **Lemma 18.** *For any $\gamma \in (0, 1/4)$, $\delta \in (0, 1/100)$, and integers $m \geqslant 1$, $\ln(1/\delta) \cdot 12/\gamma^2 \leqslant k \leqslant m/6$, $R_\delta(PTP_{m,k,\gamma}) \geqslant \frac{m \cdot \ln(1/(4\delta))}{24\,\gamma^2 \cdot k}$ where $R_\delta(\cdot)$ denotes the randomized query complexity with error probability δ.*

To prove Lemma 18, we use the easy direction of Yao's minimax principle (Proposition 28) which allows us to focus on *deterministic* algorithms for *PTP* on the input distribution. As per Problem 17, the input distribution is $D = (1/2) \cdot D_0 + (1/2) \cdot D_1$.

▶ **Lemma 19** ($\star$)**.** *In the distribution D,*

$$\Pr(|x|_1 > (1 - \gamma) \cdot k \mid D_0) \leqslant \delta \quad \textit{and} \quad \Pr(|x|_1 < (1 + \gamma) \cdot k \mid D_1) \leqslant \delta.$$

Lemma 19 implies that any algorithm that can differentiate whether $|x|_1 \geqslant (1 + \gamma) \cdot k$ or $|x|_1 \leqslant (1 - \gamma) \cdot k$ with probability $1 - \delta$ can also solve *PTP* with probability $1 - 2\delta$. This is simply because when $x \sim D_\theta$ for $\theta \in \{0, 1\}$, with probability at most δ, $|x|_1$ is not within the "right" range for such an algorithm to detect, and with another probability δ, the algorithm may fail to output the correct answer. A union bound then implies the bound of $1 - 2\delta$ on the probability of correctly solving *PTP*. We will use this later to prove Theorem 15 and in our extension to triangle counting.

For the rest of the proof, let $\mathcal{A}$ be any deterministic query algorithm on D with the worst-case number of queries $q(\mathcal{A}) := q < \frac{m \cdot \ln(1/(4\delta))}{24\,\gamma^2 \cdot k}$. Without loss of generality, we assume that $\mathcal{A}$ always makes q queries on any input (by potentially making "dummy" queries to reach q if needed). For an input $x \sim D$, we use $Q_{\mathcal{A}}(x) \in \{0, 1\}^q$ to denote the string of answers returned to the query algorithm based on x.

Distribution of $Q_{\mathcal{A}}(x)$

A key observation is that given only $Q_{\mathcal{A}}(x) = (b_1, \ldots, b_q)$, since $\mathcal{A}$ is a deterministic algorithm, we will learn the value of exactly q specific entries in x: b_1 is the value of the index of x queried first by $\mathcal{A}$, then, b_2 is the value of the second index queried by $\mathcal{A}$ where the query is uniquely determined after seeing the answer b_1 to the first query, and so on and so forth. Thus, for any choice of $\theta \in \{0,1\}$, *conditioned on* x being sampled from D_θ, for any $i \in [m]$, *independent* of the value of $(b_1, \ldots, b_{i-1})$, the value of b_i is sampled from a Bernoulli distribution with mean p_θ. This means that:

$$\text{distribution } (Q_{\mathcal{A}}(x) \mid D_0) \text{ is } \mathcal{B}(p_0)^q \qquad \text{and} \qquad \text{distribution } (Q_{\mathcal{A}}(x) \mid D_1) \text{ is } \mathcal{B}(p_1)^q.$$

The following claim bounds the KL-divergence (Equation (8)) between these two distributions.

▷ Claim 20. For any $q \geqslant 1$ and $0 < p_0, p_1 < 1/3$, we have, $\mathbb{D}(\mathcal{B}(p_0)^q \mid\mid \mathcal{B}(p_1)^q) < \ln(1/(4\delta))$.

Proof. By the chain rule of KL-divergence and using the fact that both arguments are product distributions (Fact 32), we have

$$\mathbb{D}(\mathcal{B}(p_0)^q \mid\mid \mathcal{B}(p_1)^q) = q \cdot \mathbb{D}(\mathcal{B}(p_0) \mid\mid \mathcal{B}(p_1)).$$

Moreover, for each term, using Proposition 33, we have

$$\mathbb{D}(\mathcal{B}(p_0) \mid\mid \mathcal{B}(p_1)) \leqslant \frac{(p_0 - p_1)^2}{p_1 \cdot (1 - p_1)} \leqslant \frac{(4\gamma \cdot k/m)^2}{(1 + 2\gamma) \cdot k/m \cdot 2/3} \leqslant 24\gamma^2 \cdot \frac{k}{m},$$

concluding the proof. ◁

Let us now use Claim 20 to conclude the proof. As argued earlier, all the information that is revealed to the algorithm $\mathcal{A}$ is the string $Q_{\mathcal{A}}(x)$ on an input $x \sim D$, and its task is to distinguish whether x is sampled from D_0 or D_1. By Fact 31, the best probability of success of $\mathcal{A}$ is then:

$$\begin{aligned}
\frac{1}{2} + \frac{1}{2} \cdot \|(Q_{\mathcal{A}}(x) \mid D_0) - (Q_{\mathcal{A}}(x) \mid D_1)\|_{\text{tvd}} &\leqslant 1 - \frac{1}{4} \cdot \exp\left(-\mathbb{D}(Q_{\mathcal{A}}(x) \mid D_0 \mid\mid Q_{\mathcal{A}}(x) \mid D_1)\right) \\
&\qquad \text{(by the extension of Pinsker's inequality in Proposition 34)} \\
&= 1 - \frac{1}{4} \cdot \exp\left(-\mathbb{D}(\mathcal{B}(p_0)^q \mid\mid \mathcal{B}(p_1)^q)\right) \\
&\qquad \text{(by the distribution of } Q_{\mathcal{A}}(x) \text{ argued earlier)} \\
&< 1 - \frac{1}{4} \cdot \exp\left(\ln(4\delta)\right) = 1 - \delta. \\
&\qquad \text{(by Claim 20 as } k \leqslant m/6,\ \gamma < 1/4, \text{ and thus } p_0, p_1 < 1/3)
\end{aligned}$$

This means that $\mathcal{A}$ can succeed with probability $< 1 - \delta$ in distinguishing between D_0 and D_1. Combined with the easy direction of Yao's minimax principle (namely, an averaging principle, Proposition 28), this concludes the proof of Lemma 18.

3.2 Reducing PTP to the H-Index Problem

We now prove Theorem 15 via a reduction from *PTP* and our lower bound for the latter problem in Lemma 18. Suppose towards a contradiction that there is an algorithm $\mathcal{A}_h$ for h-index that with probability $1 - \delta/2$ uses $o(n \ln(1/\delta)/(\varepsilon^2 \mathsf{h}(A)))$ queries on input array A and estimates $\mathsf{h}(A)$ to within a $(1 \pm \varepsilon)$-factor. Given an instance of $PTP_{m,k,\gamma}$, we use $\mathcal{A}_h$ to solve *PTP* with probability $1 - \delta$ in `PTP-estimator`.

It is clear that the worst-case query complexity of `PTP-estimator` is $< \tau(n, k, \varepsilon, \delta)$ by the condition on the second line of the algorithm. In terms of parameters for $PTP_{m,k,\gamma}$, this translates to the bound of $\frac{m \cdot \ln(1/(4\delta))}{24\,\gamma^2 \cdot k}$ on the worst-case query complexity of `PTP-estimator`. In the following, we will prove that *if* $\mathcal{A}_h$ truly exists, then `PTP-estimator` solves $PTP_{m,k,\gamma}$ with probability of success at least $1 - \delta$. But, then `PTP-estimator` contradicts the lower bound of Lemma 18 – this implies that $\mathcal{A}_h$ cannot exist, and we get our desired lower bound in Theorem 15.

Algorithm 4 PTP-estimator(x, k, γ, δ).

```
Run $\mathcal{A}_h$ with parameters $n = m$, $\varepsilon = \gamma$ and error $\delta/2$ on an array $A$ defined as
  follows: for any query of $\mathcal{A}_h$ to $A[i]$ for $i \in [n]$, return $A[i] = (1+\varepsilon) \cdot k$ if $x_i = 1$ and
  return 0 otherwise.
If at any point, the number of queries of $\mathcal{A}_h$ reaches
  $\tau(n, k, \varepsilon, \delta) = \frac{n \cdot \ln(1/(4\delta))}{24\varepsilon^2 \cdot k}$,
  stop $\mathcal{A}_h$ and return No as the answer.
If we never stopped $\mathcal{A}_h$, return Yes if $\mathcal{A}_h$ returns $\tilde{h} \geqslant k - \varepsilon^2 \cdot k$; otherwise return No.
```

▶ **Lemma 21.** *`PTP-estimator` outputs the correct answer to any instance of $PTP_{m,k,\gamma}$ with probability at least $1-\delta$.*

Proof. Lemma 19 implies that any algorithm that can differentiate whether $|x|_1 \geqslant (1+\gamma) \cdot k$ or $|x|_1 \leqslant (1-\gamma) \cdot k$ with probability $1-\delta/2$ can also solve *PTP* with probability $1-\delta$. Therefore, it is sufficient to prove that `PTP-estimator` outputs *Yes* when $|x|_1 \geqslant (1+\gamma) \cdot k$ and *No* when $|x|_1 \leqslant (1-\gamma) \cdot k$ with probability at least $1-\delta/2$. We consider each case of the right answer to *PTP* separately.

Case I. Suppose first that the input x to *PTP* is a *Yes*-instance, meaning that $|x|_1 \geqslant (1+\gamma) \cdot k$. Consider the array A *implicitly* constructed by `PTP-estimator`. Given that $\varepsilon = \gamma$, A contains at least $(1+\varepsilon) \cdot k$ entries each with a value of at least $(1+\varepsilon) \cdot k$. Moreover, it does not contain any entry with a value larger than $(1+\varepsilon) \cdot k$. Thus, we have $\mathsf{h}(A) = (1+\varepsilon) \cdot k$. By the guarantee of $\mathcal{A}_h$ on its correctness and since $\mathsf{h}(A) > k$, the probability that $\mathcal{A}_h$ outputs a value

$$\tilde{h} < \mathsf{h}(A) - \varepsilon \cdot \mathsf{h}(A) = (1+\varepsilon) \cdot k - \varepsilon \cdot k - \varepsilon^2 \cdot k = k - \varepsilon^2 \cdot k$$

or makes more than $\tau(n, k, \varepsilon, \delta)$ queries on A and thus we stop it is at most $\delta/2$.

Case II. Suppose now that the input x to *PTP* is a *No*-instance, meaning that $|x|_1 \leqslant (1-\gamma) \cdot k$. Consider the array A *implicitly* constructed by `PTP-estimator`. Given that $\varepsilon = \gamma$, A contains at most $(1-\varepsilon) \cdot k$ non-zero entries, so $\mathsf{h}(A) \leqslant (1-\varepsilon) \cdot k$. Thus, by the guarantee of $\mathcal{A}_h$ on its correctness, the probability that $\mathcal{A}_h$ outputs a value

$$\tilde{h} \geqslant k - \varepsilon^2 \cdot k = (1-\varepsilon) \cdot k + \varepsilon \cdot k - \varepsilon^2 \cdot k \geqslant \mathsf{h}(A) + \varepsilon \cdot \mathsf{h}(A)$$

is at most $\delta/2$. This means that *if* we do not stop $\mathcal{A}_h$ (because it has made too many queries), the output will only be wrong with probability at most $\delta/2$. But now note that we do not have any particular guarantee on the probability that we stop $\mathcal{A}_h$ as it is possible that $\mathsf{h}(A)$ is much less than k and thus the bound of $o(n \ln(1/\delta)/(\varepsilon^2 \mathsf{h}(A)))$ on the queries of $\mathcal{A}_h$ will still be way less than $\tau(n, k, \varepsilon, \delta)$. Nevertheless, even if we stop the algorithm, we output *No* as the answer and thus make no error here. Thus, in this case also, the probability of outputting a wrong answer is $\delta/2$ at most as desired.

This concludes the proof of Lemma 21. ◀

Theorem 15 now follows immediately from Lemma 18 and Lemma 21 as argued earlier.

4 Triangle Counting Problem

In this section, we switch from the main theme of our paper which was on the h-index problem and instead show an application of our lower bound techniques to the well-studied problem of subgraph counting using local queries, in particular, the triangle counting problem.

▶ **Problem 22.** *In $TCP_{n,m,\varepsilon}$, for integers $n, m \geqslant 1$ and parameter $\varepsilon \in (0,1)$, we are given an undirected graph $G = (V, E)$ with n vertices and m edges, and the goal is to estimate the number of triangles, namely, cliques on three vertices, in G to within a $(1 \pm \varepsilon)$-factor. In order to do this, we can make the following queries to the graph:*

1. *Degree queries: Given a vertex $v \in V$, return the degree of v $(\deg(v))$.*
2. *Neighbor queries: Given a vertex $v \in V$ and $i \in [n]$, return the i^{th} neighbor of v if $i \leqslant \deg(v)$ and "None" otherwise.*
3. *Pair queries: Given two vertices $u, v \in V$, return 1 if $(u, v) \in E$ and 0 otherwise.*
4. *Edge-sample queries: Return an edge $e \in E$ independently and uniformly at random.*

We refer the reader to [9, 11, 13, 1] and references therein for more on the background of this problem. Here, we only note that [9] designed an algorithm for this problem with time complexity $O^*(\frac{n}{t^{1/3}} + \frac{m^{3/2}}{t})$, where t is the number of triangles and O^* hides the dependence on ε, error probability δ, and logarithmic factors in n. The algorithm of [9] only requires the first three types of queries mentioned above, which is generally considered the baseline for sublinear time algorithms and is referred to as the general query model. Later, by using the fourth type of query also, [1] obtained an algorithm for this problem with time complexity $O(\frac{m^{3/2} \cdot \ln(1/\delta)}{\varepsilon^2 \cdot t})$ (the algorithm of [1] extends to counting *all* subgraphs, not just triangles, with a runtime depending on the fractional edge cover of the subgraph we are counting; see [1]).

On the lower bound front, [13], building on [9], proved a lower bound of $\Omega(\frac{m^{3/2}}{t})$ for the triangle counting problem under the four queries mentioned. This lower bound, however, only holds for some constant ε and δ and does not incorporate the dependence on them.

In this section, using our lower bound for the PTP problem in Lemma 18, we will improve the lower bound of [13] and obtain a lower bound that matches the algorithmic bounds of [1], settling the asymptotic complexity of the triangle counting problem in all parameters involved.

▶ **Theorem 23.** *Any algorithm that given access to an undirected graph $G = (V, E)$ through degree, neighbor, pair, and edge-sample queries, approximation parameter $\varepsilon \in (0, 1/4)$, and confidence parameter $\delta \in (0, 1/100)$, outputs an estimate $\tilde{t}$ of the number of triangles, t, in G such that $\Pr(|\tilde{t} - t| \leqslant \varepsilon \cdot t) \geqslant 1 - \delta$ requires $\Omega(\min(m, \dfrac{m^{3/2} \cdot \ln(1/\delta)}{\varepsilon^2 \cdot t}))$ queries to the graph provided that $t = o(\varepsilon \cdot m)$.*

Similarly to the h-index problem, we prove Theorem 23 via a reduction from PTP and our lower bound for that problem in Lemma 18.

▶ Remark 24. For concreteness, we focused on proving a lower bound only for the triangle counting problem as a representative of the wider family of subgraph counting problems. However, by using our PTP in place of the lower bound arguments in [11] and [1], one can also extend their lower bounds to asymptotically optimal bounds (matching the algorithm of [1]) for larger cliques as well as odd-cycles.

▶ Remark 25. To avoid confusion, in the rest of this proof, we use m to denote the number of edges in the triangle counting problem and instead use M (in place of the original m) for the dimension of the PTP problem.

Suppose towards a contradiction that there is an algorithm $\mathcal{A}_t$ for triangle counting that queries input undirected graph, G, and estimates t to within a $(1 \pm \varepsilon)$-factor with probability at least $1 - \delta/2$ using $o(m^{3/2} \ln(1/\delta)/(\varepsilon^2 t))$ queries. Given an instance of $PTP_{M,k,\gamma}$, we use $\mathcal{A}_t$ to solve PTP with probability $1 - \delta$.

Define $M = (\sqrt{m}/2)^2 = m/4$. We define a mapping from inputs of PTP, $x \in \{0,1\}^M$, to $G_x(V, E)$ on $n = 2\sqrt{m}$ vertices and m edges.

- Let the vertices of G_x consist of two sets, $U \cup V$, such that $U = \{u_1, ..., u_{\sqrt{m}}\}$ and $V = \{v_1, ..., v_{\sqrt{m}}\}$. There is no overlap between the two sets, so $U \cap V = \emptyset$. Let U consist of two sets, $U_1 \cup U_2$, such that $U_1 = \{u_1, ..., u_{\sqrt{m}/2}\}$ and $U_2 = \{u_{\sqrt{m}/2+1}, ..., u_{\sqrt{m}}\}$. Similarly, let V consist of two sets, $V_1 \cup V_2$, such that $V_1 = \{v_1, ..., v_{\sqrt{m}/2}\}$ and $V_2 = \{v_{\sqrt{m}/2+1}, ..., v_{\sqrt{m}}\}$.
- We view x as being indexed by pairs $i \in [\sqrt{m}/2], j \in [\sqrt{m}/2+1, \sqrt{m}]$ such that $i < j$. Now, we add edges in the following way. If $x_{ij} = 1$, G_x contains edges $(u_i, u_j) \in U_1 \times U_2$ and $(v_i, v_j) \in V_1 \times V_2$. If $x_{ij} = 0$, G_x contains edges $(u_i, v_j) \in U_1 \times V_2$ and $(v_i, u_j) \in V_1 \times U_2$. Additionally, for each vertex $u_1 \in U_1$ and $v_1 \in V_1$, G_x contains edge (u_1, v_1). For each vertex $u_2 \in U_2$ and $v_2 \in V_2$, G_x contains edge (u_2, v_2). There are no other edges that are added.

See Figure 1 for an illustration.

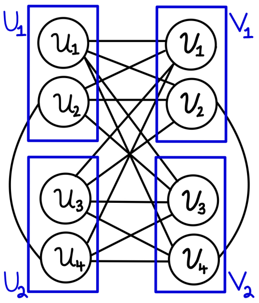

Figure 1 The graph G_x for $x = 0001$. The bits are indexed by the vertex pairs $(13, 14, 23, 24)$.

We call the reduction algorithm `PTP-estimator-two`.

It is clear that the worst-case query complexity of `PTP-estimator-two` is $< \tau(m, k, \varepsilon, \delta)$. In terms of parameters for $PTP_{M,k,\gamma}$, this translates to the bound of $\dfrac{M \cdot \ln(1/(4\delta))}{24\gamma^2 \cdot k}$ on the worst-case query complexity of `PTP-estimator-two`. In the following, we will prove that *if* $\mathcal{A}_t$ exists, then `PTP-estimator-two` solves $PTP_{M,k,\gamma}$ with probability of success at least $1-\delta$. But then, `PTP-estimator-two` contradicts the lower bound of Lemma 18 which implies that $\mathcal{A}_t$ cannot exist, and we get our desired lower bound in Theorem 23.

We note that in the following lemma, the lower bound on k and upper bound on ε is benign as otherwise the $\Omega(m)$ part of our lower bound in Theorem 23 should instead kick in.

▶ **Lemma 26.** *`PTP-estimator-two` outputs the correct answer to any instance of $PTP_{M,k,\gamma}$ with probability at least $1 - \delta$ as long as $k = \omega(\ln(1/\delta)/\varepsilon^2)$, $k = o(\varepsilon \cdot m)$, and $\varepsilon = \omega(1/\sqrt{m})$.*

Proof. Lemma 19 implies that any algorithm that can differentiate whether $|x|_1 \geqslant (1+\gamma) \cdot k$ or $|x|_1 \leqslant (1-\gamma) \cdot k$ with probability $1 - \delta/2$ can also solve PTP with probability $1 - \delta$. Therefore, it is sufficient to prove that `PTP-estimator-two` outputs *Yes* when $|x|_1 \geqslant (1+\gamma) \cdot k$ and *No* when $|x|_1 \leqslant (1-\gamma) \cdot k$ with probability at least $1 - \delta/2$.

Algorithm 5 PTP-estimator-two($x \in \{0,1\}^M$, k, γ, δ).

```
Run $\mathcal{A}_t$ with parameters $n = 2\sqrt{m}$, $m = 4M$, $\varepsilon = \gamma$, and error $\delta/2$ on an undirected
  graph $G$ defined as follows:
Degree queries. For any degree query of $\mathcal{A}_t$, return $\sqrt{m}$.
Neighbor queries. For any neighbor query of $\mathcal{A}_t$, do the following. Assume w.l.o.g.
  that we get a vertex $u_i \in U_1$ and want to find the $k^{th}$ neighbor. If $k \leqslant \sqrt{m}/2$,
  return $v_i$. Otherwise, set $j \leftarrow k$. Then, if $x_{ij}$ is 1, return $u_j$; else, $v_j$.
Pair queries. For any pair query of $\mathcal{A}_t$, if an edge between a vertex $u \in U_1$ and a
  vertex $v \in V_1$ or between $u \in U_2$ and $v \in V_2$ is queried, return 1. If an edge between
  any two vertices in $U_1$, $U_2$, $V_1$, or $V_2$ is queried, return 0. Else, for some query
  $(u_i, v_j)$ such that $i < j$, return $\neg x_{ij}$. For some query $(u_i, u_j)$ such that $i < j$, return
  $x_{ij}$.
Edge-sample queries. For any random edge-sample query made by $\mathcal{A}_t$, uniformly at
  random pick a vertex $v \in V$ and then uniformly at random pick one of its neighbors
  $u$. Return the edge $(u, v)$.
If at any point, the number of queries of $\mathcal{A}_t$ reaches

    $\tau(m, k, \varepsilon, \delta) = \frac{m \cdot \ln(1/(4\delta))}{9600\varepsilon^2 \cdot k}$,

  stop $\mathcal{A}_t$ and return No as the answer.
If we never stopped $\mathcal{A}_t$, return Yes if $\mathcal{A}_t$ returns $\tilde{t} \geqslant 2k(\sqrt{m} - 2)(1 - \varepsilon^2)$; otherwise,
  return No.
```

Within G_x, we will define **red edges**. Let the red edges include any edges between any two vertices $\in U_1$. The set of red edges will also include any edges between any two vertices $\in V_1$. For every vertex v, we define **reddeg(v)** as the number of red edges incident on v.

We consider each case of the right answer to *PTP* separately.

Case I. Suppose first that the input x to *PTP* is a *Yes*-instance, meaning that for each index $i \in [M]$, x_i was set to 1 independently with probability $(1 + 2\gamma) \cdot k/M$. Consider the graph G *implicitly* constructed by `PTP-estimator-two`. For every bit set to 1 in x, there are two red edges in G_x. Each red edge (u, v) creates $(\sqrt{m} - 2) - \text{reddeg}(u) - \text{reddeg}(v)$ triangles.

We want to ensure that in the *Yes*-instance, there are enough triangles. We first lower bound the total number of red edges. Since the number of red edges corresponds to $|x|_1$, we can use Lemma 19. By the choice of $k = \omega(\ln(1/\delta)/\varepsilon^2)$, we can see that the probability that $|x|_1 < (1+\gamma) \cdot k$ is bounded by $\delta/2$. Now, we bound for each edge, (u, v), $\text{reddeg}(u) + \text{reddeg}(v)$. Let us first bound the number of red edges incident on each vertex.

▷ Claim 27. When x is a *Yes*-instance, for each vertex v, $\Pr(\mathit{reddeg}(v) > \varepsilon/3 \cdot \sqrt{m}) \leqslant \delta/\sqrt{m}$.

Proof. For each vertex v, the probability of an edge incident on it being red is $(1+2\varepsilon) \cdot k/(m/4)$ and there are potentially $\sqrt{m}/2$ red edges. Therefore, $\mathbb{E}[\text{reddeg}(v)] = (1+2\varepsilon) \cdot k/(m/4) \cdot \sqrt{m}/2$. By the lower bound on k, $\mathbb{E}[\text{reddeg}(v)] \leqslant \varepsilon/4 \cdot \sqrt{m}$. We now use the Chernoff bound (Proposition 30) to bound the probability that $\text{reddeg}(v)$ is too large and have

$$\Pr(\text{reddeg}(v) > \varepsilon/3 \cdot \sqrt{m}) \leqslant \exp(-\frac{(1/3)^2 \cdot \mathbb{E}[\text{reddeg}(v)]}{3}) \leqslant \delta/\sqrt{m}$$

where the last inequality is because of the lower bound on ε. ◁

Claim 27 implies that any edge (u, v), $\text{reddeg}(u) + \text{reddeg}(v)$ is at most $2/3 \cdot \varepsilon/\sqrt{m}$. Thus, by the guarantee of $\mathcal{A}_t$ on its correctness, the probability that $\mathcal{A}_t$ outputs a value

$$\tilde{t} < t - \varepsilon \cdot t \leqslant 2(\sqrt{m} - 2)(1 + \varepsilon) \cdot k - \varepsilon \cdot 2(\sqrt{m} - 2)(1 + \varepsilon) \cdot k = 2k(\sqrt{m} - 2)(1 - \varepsilon^2)$$

is at most $\delta/2$. This means that *if* we do not stop $\mathcal{A}_t$ (because it has made too many queries), the output will only be wrong with probability at most $\delta/2$. Additionally, since $t/(2(\sqrt{m} - 2)) > k$ and the number of queries made by $\mathcal{A}_t$ is supposed to be $o(m^{3/2} \ln(1/\delta)/(\varepsilon^2 t))$, $\mathcal{A}_t$ will never make more than $\tau(m, k, \varepsilon, \delta)$ queries on G. Therefore, in this case, the probability of outputting a wrong answer is at most $\delta/2$ as desired.

Case II. Suppose instead that the input x to *PTP* is a *No*-instance, meaning that for each index $i \in [M]$, x_i was set to 1 independently with probability $(1 - 2\gamma) \cdot k/M$. Consider the graph G *implicitly* constructed by `PTP-estimator-two`. Every red edge can create at most $(\sqrt{m} - 2)$ triangles with vertices on the other side of the bipartition.

We first bound the total number of red edges. Since the number of red edges corresponds to $|x|_1$, we can use Lemma 19. By the choice of $k = \omega(\ln(1/\delta)/\varepsilon^2)$, we can see that the probability that $|x|_1 > (1 - \gamma) \cdot k$ is bounded by $\delta/2$. Therefore, by the guarantee of $\mathcal{A}_t$ on its correctness, the probability that $\mathcal{A}_t$ outputs a value

$$\tilde{t} \geqslant 2k(\sqrt{m} - 2)(1 - \varepsilon^2) = 2(\sqrt{m} - 2)(1 - \varepsilon) \cdot k + \varepsilon \cdot 2(\sqrt{m} - 2)(1 - \varepsilon) \cdot k \geqslant t + \varepsilon \cdot t$$

is at most $\delta/2$. This means that *if* we do not stop $\mathcal{A}_t$ (because it has made too many queries), the output will only be wrong with probability at most $\delta/2$. But now note that we do not have any particular guarantee on the probability that we stop $\mathcal{A}_t$ since it is possible that $t/(2(\sqrt{m}-2))$ is much less than k and thus the bound of $o(m^{3/2} \ln(1/\delta)/(\varepsilon^2 t))$ on the queries of $\mathcal{A}_t$ will still be much less than $\tau(m, k, \varepsilon, \delta)$. Nevertheless, even if we stop the algorithm, we output *No* as the answer and thus make no error here. Thus, in this case also, the probability of outputting a wrong answer is $\delta/2$ at most as desired.

This concludes the proof of Lemma 26. ◀

References

1 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.

2 Sepehr Assadi and Vihan Shah. An asymptotically optimal algorithm for maximum matching in dynamic streams. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 9:1–9:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

3 Sayan Bhattacharya, Fabrizio Grandoni, Janardhan Kulkarni, Quanquan C. Liu, and Shay Solomon. Fully dynamic (Δ +1)-coloring in $O(1)$ update time. *ACM Trans. Algorithms*, 18(2):10:1–10:25, 2022.

4 Arijit Bishnu, Arijit Ghosh, Gopinath Mishra, and Manaswi Paraashar. Query complexity of global minimum cut. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 6:1–6:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

5 Vladimir Braverman, Jonathan Katzman, Charles Seidell, and Gregory Vorsanger. An optimal algorithm for large frequency moments using o(n^(1-2/k)) bits. In Klaus Jansen, José D. P. Rolim, Nikhil R. Devanur, and Cristopher Moore, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, volume 28 of *LIPIcs*, pages 531–544. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.

6 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.*, 288(1):21–43, 2002.

7 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms.* Cambridge University Press, 2009.

8 Talya Eden, Shweta Jain, Ali Pinar, Dana Ron, and C. Seshadhri. Provable and practical approximations for the degree distribution using sublinear graph samples. *CoRR*, abs/1710.08607, 2017. `arXiv:1710.08607`.

9 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.

10 Talya Eden, Saleet Mossel, and Ronitt Rubinfeld. Sampling multiple edges efficiently. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 51:1–51:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

11 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.

12 Talya Eden, Dana Ron, and C. Seshadhri. Faster sublinear approximation of the number of k-cliques in low-arboricity graphs. 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 1467–1478. SIAM, 2020.

13 Talya Eden and Will Rosenbaum. Lower bounds for approximating graph parameters via communication complexity. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, volume 116 of *LIPIcs*, pages 11:1–11:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

14 Talya Eden and Will Rosenbaum. On sampling edges almost uniformly. In Raimund Seidel, editor, *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, volume 61 of *OASIcs*, pages 7:1–7:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

15 Hendrik Fichtenberger, Mingze Gao, and Pan Peng. Sampling arbitrary subgraphs exactly uniformly in sublinear time. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 45:1–45:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

16 Alison L Gibbs and Francis Edward Su. On choosing and bounding probability metrics. *International statistical review*, 70(3):419–435, 2002.

17 Oded Goldreich. *Introduction to Property Testing.* Cambridge University Press, 2017.

18 Priya Govindan, Morteza Monemizadeh, and S. Muthukrishnan. Streaming algorithms for measuring h-impact. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '17, pages 337–346, New York, NY, USA, 2017. Association for Computing Machinery. `doi:10.1145/3034786.3056118`.

19 Monika Henzinger and Pan Peng. Constant-time dynamic (Δ+1)-coloring. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 53:1–53:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

20 Jorge E. Hirsch. An index to quantify an individual's scientific research output. *Proc. Natl. Acad. Sci. USA*, 102(46):16569–16572, 2005.

21 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In Jan Paredaens and Dirk Van Gucht, editors, *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 41–52. ACM, 2010.

22 Michael Kapralov, Jelani Nelson, Jakub Pachocki, Zhengyu Wang, David P. Woodruff, and Mobin Yahyazadeh. Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 475–486. IEEE Computer Society, 2017.

23 Yi Li and David P. Woodruff. A tight lower bound for high frequency moment estimation with small error. In Prasad Raghavendra, Sofya Raskhodnikova, Klaus Jansen, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, volume 8096 of *Lecture Notes in Computer Science*, pages 623–638. Springer, 2013.

24 Linyuan Lü, Tao Zhou, Qian-Ming Zhang, and H. Eugene Stanley. The H-index of a network node and its relation to degree and coreness. *Nature Communications*, 7(1):1–7, April 2016. doi:10.1038/ncomms10168.

25 Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1844–1860, 2019.

26 Eric Price and David P. Woodruff. (1 + eps)-approximate sparse recovery. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 295–304. IEEE Computer Society, 2011.

27 Eric Price and David P. Woodruff. Lower bounds for adaptive sparse recovery. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 652–663. SIAM, 2013.

28 Fabián Riquelme and Pablo Gonzalez Cantergiani. Measuring user influence on twitter: A survey. *Inf. Process. Manag.*, 52(5):949–975, 2016.

29 Ahmet Erdem Sariyüce, C. Seshadhri, and Ali Pinar. Local algorithms for hierarchical dense subgraph discovery. *Proc. VLDB Endow.*, 12(1):43–56, 2018. doi:10.14778/3275536.3275540.

30 Shay Solomon. Fully dynamic maximal matching in constant update time. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 325–334. IEEE Computer Society, 2016.

31 Jakub Tětek and Mikkel Thorup. Sampling and counting edges via vertex accesses. *arXiv preprint arXiv:2107.03821. To appear in STOC 2022*, 2021.

32 Alexandre B. Tsybakov. *Introduction to Nonparametric Estimation.* Springer series in statistics. Springer, 2009. doi:10.1007/b13794.

33 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 222–227. IEEE Computer Society, 1977.

A Detailed Preliminaries

A.1 Basics of Query Complexity

We use the basics of query complexity to establish our lower bounds on the runtime of sublinear algorithms (as the number of queries made to the input is always a lower bound on the runtime).

Let $f : \{0,1\}^n \mapsto \{0,1\}$ be any Boolean function. A query algorithm for f on any input x can *query* the values of x_i for $i \in [n]$ and determine the value of $f(x)$ with a minimal number of queries. We will work with the following definitions:

- **Randomized query complexity**: For any $\delta \in (0,1)$, $R_\delta(f)$ denotes the worst-case number of queries made by the best randomized algorithm that computes f on any input with probability of success at least $1-\delta$.
- **Distributional query complexity**: For any $\delta \in (0,1)$ and any distribution μ on $\{0,1\}^n$, $D_{\mu,\delta}(f)$ denotes the worst-case number of queries made by the best deterministic algorithm that computes f on inputs sampled from μ with probability of success at least $1-\delta$.

Yao's minimax principle [33] relates these two measures.

▶ **Proposition 28** (Yao's minimax principle [33]). *For any $f : \{0,1\}^n \mapsto \{0,1\}$ and $\delta \in (0,1)$:*

(i) Easy direction (averaging argument): *For any distribution μ on $\{0,1\}^n$, $D_{\mu,\delta}(f) \leqslant R_\delta(f)$.*

(ii) Hard direction (duality): *There is some distribution μ^* on $\{0,1\}^n$ such that $D_{\mu^*,\delta}(f) = R_\delta(f)$.*

A.2 Basic Probabilistic Tools

We use the linearity of variance of independent random variables.

▶ **Fact 29.** *For any two* independent *random variables X and Y,* $\mathrm{Var}[X+Y] = \mathrm{Var}[X] + \mathrm{Var}[Y]$.

The following proposition lists the standard concentration inequalities we use in this paper.

▶ **Proposition 30** (Concentration Inequalities; cf. [7]).

(i) Chebyshev's inequality: *For any random variable X and $t > 0$,*

$$\Pr\left(|X - \mathbb{E}[X]| \geqslant t\right) \leqslant \frac{\mathrm{Var}[X]}{t^2}.$$

(ii) Chernoff bound: *Suppose $X_1, \ldots, X_n$ are n independent random variables in $[0,1]$ and define $X := \sum_{i=1}^n X_i$. Then, for any $\varepsilon \in (0,1)$ and $\mu \geqslant \mathbb{E}[X]$,*

$$\Pr\left(X > (1+\varepsilon)\cdot\mu\right) \leqslant \exp\left(-\frac{\varepsilon^2\cdot\mu}{3}\right) \text{ and } \Pr\left(X < (1-\varepsilon)\cdot\mu\right) \leqslant \exp\left(-\frac{\varepsilon^2\cdot\mu}{3}\right).$$

Moreover, for any $t \geqslant 1$ and $\mu \geqslant \mathbb{E}[X]$, $\Pr\left(|X - \mathbb{E}[X]| \geqslant t\cdot\mu\right) \leqslant 2\cdot\exp\left(-\frac{t\cdot\mu}{3}\right)$.

A.3 Measures of Distance Between Distributions

We use two main measures of distance (or divergence) between distributions, namely the *total variation distance* and the *Kullback-Leibler divergence* (KL-divergence).

Total variation distance

We denote the total variation distance between two distributions μ and ν on the same support Ω by $\|\mu - \nu\|_{\mathrm{tvd}}$, defined as:

$$\|\mu - \nu\|_{\mathrm{tvd}} := \max_{\Omega' \subseteq \Omega} \left(\mu(\Omega') - \nu(\Omega')\right) = \frac{1}{2} \cdot \sum_{x \in \Omega} \left|\mu(x) - \nu(x)\right|. \tag{7}$$

We use the following basic property of total variation distance.

▶ **Fact 31.** *Suppose μ and ν are two distributions with same support Ω; then, given a single sample from either μ or ν, the best probability of successfully deciding whether s came from μ or ν is $\frac{1}{2} + \frac{1}{2} \cdot \|\mu - \nu\|_{\mathrm{tvd}}$.*

KL-divergence

For two distributions μ and ν over the same probability space, the *Kullback-Leibler divergence* between μ and ν is denoted by $\mathbb{D}(\mu \mid\mid \nu)$ and defined as:

$$\mathbb{D}(\mu \mid\mid \nu) := \mathop{\mathbb{E}}_{a \sim \mu} \left[\log \frac{\Pr_{\mu}(a)}{\Pr_{\nu}(a)}\right]. \tag{8}$$

A key property of KL-divergence is that it satisfies a chain rule.

▶ **Fact 32** (Chain rule for KL-divergence)**.** *Given two distributions $p(x_1, \ldots, x_t)$ and $q(x_1, \ldots, x_t)$ on t-tuples, we have,*

$$\mathbb{D}(p \mid\mid q) = \sum_{i=1}^{t} \mathop{\mathbb{E}}_{p(x_{<i})} \mathbb{D}(p(x_i \mid x_{<i}) \mid\mid q(x_i \mid x_{<i})).$$

In particular, if p and q are product distributions, then,

$$\mathbb{D}(p \mid\mid q) = \sum_{i=1}^{t} \mathbb{D}(p(x_i) \mid\mid q(x_i)).$$

The following result gives a simple upper bound for the KL-divergence of two Bernoulli distributions that we shall use in our proofs.

▶ **Proposition 33** (KL-divergence on Bernoulli distributions; c.f. [16, Theorem 5])**.** *For any $0 < p, q < 1$, the following is true:*

$$\mathbb{D}(\mathcal{B}(p) \mid\mid \mathcal{B}(q)) \leqslant \frac{(p-q)^2}{q \cdot (1-q)}.$$

We shall also use the following extension of Pinsker's inequality to relate total variation distance and Kullback-Leibler divergence.

▶ **Proposition 34** (c.f. [32], p. 88-89)**.** *Given two distributions μ and ν over the same discrete support, $\|\mu - \nu\|_{\mathrm{tvd}} \leqslant 1 - \dfrac{1}{2} \exp\left(-\mathbb{D}(\mu \mid\mid \nu)\right)$.*