Approximating Subadditive Hadamard Functions on Implicit Matrices

Vladimir Braverman[*1], **Alan Roytman**[†2], **and Gregory Vorsanger**[‡3]

1 **Department of Computer Science, Johns Hopkins University, Baltimore, USA**
vova@cs.jhu.edu
2 **School of Computer Science, Tel-Aviv University, Israel**
alan.roytman@cs.tau.ac.il
3 **Department of Computer Science, Johns Hopkins University, Baltimore, USA**
gregvorsanger@jhu.edu

Abstract

An important challenge in the streaming model is to maintain small-space approximations of entrywise functions performed on a matrix that is generated by the outer product of two vectors given as a stream. In other works, streams typically define matrices in a standard way via a sequence of updates, as in the work of Woodruff [22] and others. We describe the matrix formed by the outer product, and other matrices that do not fall into this category, as implicit matrices. As such, we consider the general problem of computing over such implicit matrices with Hadamard functions, which are functions applied entrywise on a matrix. In this paper, we apply this generalization to provide new techniques for identifying independence between two data streams. The previous state of the art algorithm of Braverman and Ostrovsky [9] gave a $(1 \pm \epsilon)$-approximation for the L_1 distance between the joint and product of the marginal distributions, using space $O(\log^{1024}(nm)\epsilon^{-1024})$, where m is the length of the stream and n denotes the size of the universe from which stream elements are drawn. Our general techniques include the L_1 distance as a special case, and we give an improved space bound of $O(\log^{12}(n)\log^2(\frac{nm}{\epsilon})\epsilon^{-7})$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Streaming Algorithms, Measuring Independence, Hadamard Functions, Implicit Matrices

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2016.25

1 Introduction

Measuring independence is a fundamental statistical problem that is well studied in computer science. Traditional non-parametric methods of testing independence over empirical data usually require space complexity that is polynomial in either the support size or input size. With large datasets, these space requirements may be impractical, and designing small-space algorithms becomes desirable.

Measuring independence is a classic problem in the field of statistics (see Lehmann [17]) as well as an important problem in databases. Further, the process of reading in a two-column

* This work was sponsored in part by the National Science Foundation under Grant No. 1447639 and the Google Faculty Award.
† The work of Alan Roytman was partially supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement number 337122, and in part by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).
‡ This work was supported in part by Raytheon BBN Technologies.

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016).
Editors: Klaus Jansen, Claire Matthieu, José D. P. Rolim, and Chris Umans; Article No. 25; pp. 25:1–25:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

database table can be viewed as a stream of pairs. Thus, the streaming model is a natural choice when approximating pairwise independence as memory is limited. Indeed, identifying correlations between database columns by measuring the level of independence between columns is of importance to the database and data warehouse community (see, e.g., [19] and [16], respectively).

In this paper we provide new techniques for measuring independence between two data streams and present new tools to expand existing techniques. The topic of independence was first studied in the streaming model by Indyk and McGregor [15] where the authors gave an optimal algorithm for approximating the L_2 distance between the joint and product of the marginal distributions of two random variables which generate a stream. In their work, they provided a sketch that is pairwise independent, but not 4-wise independent, so analysis similar to that of Alon, Matias, and Szegedy [3] cannot be applied directly. This work was continued by Braverman and Ostrovsky [9], where the authors considered comparing among a stream of k-tuples and provided the first $(1 \pm \epsilon)$-approximation for the L_1 distance between the joint and product of the marginal distributions. Their algorithm is currently the best known space bound, and uses $O(\frac{1}{\epsilon^{1024}} \log^{1024}(nm))$ space for $k = 2$, where m is the length of the stream and n denotes the size of the universe from which stream elements are drawn. We present new methods, in the form of a general tool, that enable us to improve this bound to $O(\frac{1}{\epsilon^7} \log^{12}(n) \log^2(\frac{nm}{\epsilon}))$. In previous works, a central challenge has been maintaining an approximation of the matrix that is generated by the outer product of the two streaming vectors. As such, we consider computing functions on such an implicit matrix. While matrices have been studied previously in the streaming model (e.g., [22]), note that we cannot use standard linear sketching techniques, as the entries of the matrix are given implicitly and thus these methods do not apply directly.

Generalizing this specific motivating example, we consider the problem of obtaining a $(1 \pm \epsilon)$-approximation of the L_1 norm of the matrix $g[A]$, where $g[A]$ is the matrix A with a function g applied to it entrywise. Such mappings g are called *Hadamard* functions (see [12, 13]). Note that we sometimes abuse notation and apply the function g to scalar values instead of matrices (e.g., $g(a_{ij})$ where a_{ij} is the $(i,j)^{th}$ entry in matrix A). We require the scalar form of the function g to be even, subadditive, non-negative, and zero at the origin. We show that, given a blackbox $r(n)$-approximation of $\|g[A]\|_1 = \sum_i \sum_j g(a_{ij})$ (where a_{ij} is the $(i,j)^{th}$ entry in matrix A) and a blackbox $(1 \pm \epsilon)$-approximation of the aggregate of g applied entrywise to a vector obtained by summing over all rows, we are able to improve the $r(n)$-approximation to a $(1 \pm \epsilon)$-approximation (where $r(n)$ is a sufficiently large monotonically increasing function of n). Hence, we give a reduction for any such function g. Our reduction can be applied as long as such blackbox algorithms exist.

An interesting special case of our result is when the matrix is defined by the L_1 distance between the joint and product of the marginal distributions, which corresponds to measuring independence in data streams. Since such blackbox algorithms are known for L_1, not only does our framework generalize the problem of measuring independence according to the L_1 distance, but our algorithmic techniques also yield improved space bounds over the previous state of the art result [9]. Moreover, our framework would immediately translate improved space bounds for the blackbox algorithms to improved space bounds for the application of measuring independence. Note that, for L_p where $0 < p < 1$, such blackbox algorithms are not known. If such algorithms for the L_p distance were to be designed, our reductions work and can be applied. While there are a variety of ways to compute distances between distributions, the L_p distance is of particular significance as evidenced in [14].

Motivating Problem

We begin by presenting our motivating problem, which concerns (approximately) measuring the distance between the joint and product of the marginal distributions of two random variables. That is, we attempt to quantify how close two random variables X and Y over a universe $[n] = \{1, \ldots, n\}$ are to being independent. There are many ways to measure the distance between distributions, but we focus on the L_1 distance. Recall that two random variables X and Y are independent if we have $\Pr[X = i \wedge Y = j] = \Pr[X = i] \Pr[Y = j]$ for every i and j. In our model, we have a data stream D which is presented as a sequence of m pairs $d_1 = (i_1, j_1), d_2 = (i_2, j_2), \ldots, d_m = (i_m, j_m)$. Each pair $d_k = (i_k, j_k)$ consists of two integers taken from the universe $[n]$.

Intuitively, we imagine that the two random variables X and Y over the universe $[n]$ generate these pairs, and in particular, the frequencies of each pair (i, j) define an empirical joint distribution, which is the fraction of pairs that equal (i, j). At the same time, the stream also defines the empirical marginal distributions $\Pr[X = i], \Pr[Y = j]$, namely the fraction of pairs of the form $(i, \cdot)$ and $(\cdot, j)$, respectively. We note that, even if the pairs are actually generated from two independent sources, it may not be the case that the empirical distributions reflect this fact, although for sufficiently long streams the joint distribution should approach the product of the marginal distributions for each i and j. This fundamental problem has received considerable attention within the streaming community, including the works of [15, 9]. We note that the main theoretical contribution of this paper is focused on a generalization of this problem. Nevertheless, this application serves as a very important motivation for our framework, and we explain how to apply our framework to it in Section 5. For the main problem we solve, please see Problem 2.

▶ **Problem 1.** *Let X and Y be two random variables which generate a stream of m pairs $d_1 = (i_1, j_1), \ldots, d_m = (i_m, j_m)$, where each $i_k, j_k \in [n]$ for all k. Define the frequencies $p_i = |\{k : d_k = (i, \cdot)\}|$ and $q_j = |\{k : d_k = (\cdot, j)\}|$ (i.e., the frequency with which i appears in the first coordinate and j appears in the second coordinate, respectively). Moreover, let $f_{ij} = |\{k : d_k = (i, j)\}|$ be the frequency with which the pair (i, j) appears in the stream. This naturally defines the joint distribution $\Pr[X = i \wedge Y = j] = \frac{f_{ij}}{m}$ and the product of the marginal distributions $\Pr[X = i] \Pr[Y = j] = \frac{p_i q_j}{m^2}$. The L_1 distance between the joint and product of the marginal distributions is given by:*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \left| \frac{f_{ij}}{m} - \frac{p_i q_j}{m^2} \right|.$$

If X and Y are independent, we should expect this sum to be close to 0, assuming the stream is sufficiently long. As a generalization to this problem, we can view the n^2 values which appear in the summation as being implicitly represented via an $n \times n$ matrix, where the $(i, j)^{th}$ entry is given by $\left| \frac{f_{ij}}{m} - \frac{p_i q_j}{m^2} \right|$. For the motivating problem, this matrix is given implicitly as it is not given up front and changes over time according to the data stream (each new pair in the stream may change multiple entries in the matrix). However, one can imagine settings in which these entries are defined through other means. In practice, we may still be interested in computing approximate statistics over such implicitly defined matrices.

Contributions and Techniques

Our main contributions in this paper make progress on two important problems:

Table 1 Comparing Approximation Ratios and Space Complexity.

Previous Work	L_1 approximation	Memory
IM08 [15]	$\log(n)$	$O\left(\frac{1}{\epsilon^2}\log\left(\frac{nm}{\epsilon}\right)\log\left(\frac{m}{\epsilon}\right)\right)$
BO10[1] [9]	$(1\pm\epsilon)$	$O\left(\left(\frac{\log(nm)}{\epsilon}\right)^{1024}\right)$
Our Result	$(1\pm\epsilon)$	$O\left(\frac{1}{\epsilon^7}\log^{12}(n)\log^2\left(\frac{nm}{\epsilon}\right)\right)$

1. For any subadditive even Hadamard function g where g is non-negative and $g(0)=0$, given an implicitly defined $n\times n$ matrix A with entries a_{ij}, let $g[A]$ be the matrix where the $(i,j)^{th}$ entry is $g(a_{ij})$. We are the first to provide a general reduction framework for approximating $\|g[A]\|_1=\sum_{i=1}^n\sum_{j=1}^n g(a_{ij})$ to within a $(1\pm\epsilon)$-factor with constant success probability. More formally, suppose we have two blackbox algorithms with the following guarantees. One blackbox algorithm operates over the implicit matrix A and provides a very good ($\approx 1\pm\epsilon$) approximation to $\|g[JA]\|_1=\sum_{j=1}^n g(\sum_{i=1}^n a_{ij})$ except with inverse polylogarithmic probability, where $J=(1,\ldots,1)$ is the row vector of dimension n with every entry equal to 1. The second blackbox algorithm operates over the implicit matrix A and solves the problem we wish to solve (i.e., approximating $\|g[A]\|_1$) with constant success probability, although it does so with a multiplicative approximation ratio of $r(n)$ (which may be worse than $(1\pm\epsilon)$ in general). We show how to use these two blackbox algorithms to construct an algorithm that achieves a $(1\pm\epsilon)$-approximation of $\|g[A]\|_1$. If S_1,S_2 denote the space used by the first and second blackbox algorithms, respectively, then our algorithm uses space $O\left(\frac{r^4(n)\log^8(n)}{\epsilon^5}\cdot(\log^2(n)+S_1+\log(n)\cdot S_2)\right)$. We state this formally in Theorem 3.
2. Given the contribution above, it follows that setting $g(x)=|x|$ solves Problem 1, namely the problem of measuring how close two random variables are to being independent, as long as such blackbox algorithms exist. In particular, the work of Indyk [14] provides us with the first blackbox algorithm, and the work of [15] provides us with the second blackbox algorithm for this choice of g. Combining these results, we improve over the previous state of the art result of Braverman and Ostrovsky [9] and give improved bounds for measuring independence of random variables in the streaming model by reducing the space usage from $O\left(\left(\frac{\log(nm)}{\epsilon}\right)^{1024}\right)$ to $O\left(\frac{1}{\epsilon^7}\log^{12}(n)\log^2\left(\frac{nm}{\epsilon}\right)\right)$ (see Table 1).

Examples of such Hadamard functions which are subadditive, even, non-negative, and zero at the origin include $g(x)=|x|^p$, for any $0<p\leq 1$. Note that our reduction in the first item can only be applied to solve the problem of approximating $\|g[A]\|_1$ if such blackbox algorithms exist, but for some functions g this may not be the case. As a direct example of the tools we present, we give a reduction for computing the L_p distance for $0<p<1$ between the joint and product of the marginal distributions in the streaming model (as this function is even and subadditive). However, to the best of our knowledge, such blackbox algorithms do not exist for computing the L_p distance. Thus, as a corollary to our main result, the construction of such space efficient blackbox algorithms would immediately yield a space efficient algorithm that measures independence according to the L_p distance.

[1] The paper of [9] provides a general bound for the L_1 distance for k-tuples, but we provide analysis for pairs of elements, $k=2$, in this paper. The bound in the table is for $k=2$.

Our techniques leverage concepts provided in [9, 15] and manipulates them to allow them to be combined with the Recursive Sketches data structure [11] to gain a large improvement compared to existing bounds. Note that we cannot use standard linear sketching techniques because the entries of the matrix are given implicitly. Moreover, the sketch of Indyk and McGregor [15] is pairwise independent, but not 4-wise independent. Therefore, we cannot apply the sketches of [3, 15] directly. We first present an algorithm, independent of the streaming model, for finding heavy rows of a matrix norm given an arbitrary even subadditive Hadamard function g. In order to do this, we first prove a key theorem regarding such Hadamard functions g which states that the quantity $\|g[JA]\|_1 = \sum_{j=1}^{n} g(\sum_{i=1}^{n} a_{ij})$ is a $(1 \pm \epsilon)$-approximation to the heavy row of the matrix $g[A]$ (if it exists). With this in mind, we show how to use the blackbox algorithm that yields an $r(n)$-approximation to $\|g[A]\|_1$ in order to identify when heavy rows exist in the matrix, and then use the other blackbox algorithm to obtain a $(1 \pm \epsilon)$-approximation of $\|g[JA]\|_1$ (which is in turn a $(1 \pm \epsilon)$-approximation to the heavy row, as just mentioned). These ideas form the foundation of our algorithm for approximating heavy rows. We then apply the Recursive Sum algorithm from [11] on top of our heavy rows algorithm to obtain our main result.

1.1 Related Work

In their seminal 1996 paper Alon, Matias, and Szegedy[3] provided an optimal space approximation for L_2. A key technical requirement of the sketch is the assumption of 4-wise independent random variables. This technique is the building block for measuring the independence of data streams using L_2 distances as well.

The problems of efficiently testing pairwise, or k-wise, independence were considered by Alon, Andoni, Kaufman, Matulef, Rubinfeld, and Xie [1]; Alon, Goldreich, and Mansour [2]; Batu, Fortnow, Fischer, Kumar, Rubinfeld, and White [4]; Batu, Kumar, and Rubinfeld [7]; Batu, Fortnow, Rubinfeld, Smith, and White [5, 6]. They addressed the problem of minimizing the number of samples needed to obtain a sufficient approximation, when the joint distribution is accessible through a sampling procedure.

In their 2008 work, Indyk and McGregor [15] provided exciting results for identifying the correlation of two streams, providing an optimal bound for determining the L_2 distance between the joint and product of the marginal distributions of two random variables.

In addition to the L_2 result, Indyk and McGregor presented a $\log(n)$-approximation for the L_1 distance. This bound was improved to a $(1 \pm \epsilon)$-approximation in the work of Braverman and Ostrovsky [9] in which they provided a bound of $O(\frac{1}{\epsilon^{1024}} \log^{1024}(nm))$ for pairs of elements. Further, they gave bounds for the comparison of multiple streaming vectors and determining k-wise relationships for L_1 distance. In addition, Braverman et al. [8] expanded the work of [15] to k dimensions for L_2. Recently, McGregor and Vu [18] studied a related problem regarding Bayesian networks in the streaming model.

Statistical distance, L_1, is one of the most fundamental metrics for measuring the similarity of two distributions. It has been the metric of choice in many of the above testing papers, as well as others such as Rubinfeld and Servedio [20]; Sahai and Vadhan [21]. As such, a main focus of this work is improving bounds for this measure in the streaming model.

2 Problem Definition and Notation

In this paper we focus on the problem of approximating even, subadditive, non-negative Hadamard functions which are zero at the origin on implicitly defined matrices (e.g., the

streaming model implicitly defines matrices for us in the context of measuring independence). The main problem we study in this paper is the following:

▶ **Problem 2.** *Let g be any even, subadditive, non-negative Hadamard function such that $g(0) = 0$. Given any implicit matrix A, for any $\epsilon > 0$, $\delta > 0$, output a $(1 \pm \epsilon)$-approximation of $\|g[A]\|_1$ except with probability δ.*

We now provide our main theorem, which solves Problem 2.

▶ **Theorem 3.** *Let g be any even, subadditive, non-negative Hadamard function g where $g(0) = 0$, and fix $\epsilon > 0$. Moreover, let A be an arbitrary matrix, and J be the all 1's row vector $J = (1, \ldots, 1)$ of dimension n. Suppose there are two blackbox algorithms with the following properties:*

1. *Blackbox Algorithm 1, for all $\epsilon' > 0$, returns a $(1 \pm \epsilon')$-approximation of $\|g[JA]\|_1$, except with probability δ_1.*
2. *Blackbox Algorithm 2 returns an $r(n)$-approximation of $\|g[A]\|_1$, except with probability δ_2 (where $r(n)$ is a sufficiently large monotonically increasing function of n).*

Then, there exists an algorithm that returns a $(1\pm\epsilon)$-approximation of $\|g[A]\|_1$, except with constant probability. If Blackbox Algorithm 1 uses space $SPACE1(n, \delta_1, \epsilon')$, and Blackbox Algorithm 2 uses space $SPACE2(n, \delta_2)$, the resulting algorithm has space complexity

$$O\left(\frac{r^4(n)}{\epsilon^5}(\log^{10}(n) + \log^8(n)SPACE1(n, \delta_1, \epsilon') + \log^9(n)SPACE2(n, \delta_2))\right),$$

where $\epsilon' = \frac{\epsilon}{2}$, δ_1 is a small constant, and δ_2 is inverse polylogarithmic.

Note that we can reduce the constant failure probability to inverse polynomial failure probability via standard techniques, at the cost of increasing our space bound by a logarithmic factor. Observe that Problem 2 is a general case of Problem 1 where $g(x) = |x|$ (i.e., L_1 distance). In the streaming model, we receive matrix A implicitly, but we conceptualize the problem as if the matrix were given explicitly and then resolve this issue by assuming we have blackbox algorithms that operate over the implicit matrix.

We define our stream such that each element in the stream d_k is a pair of values (i, j):

▶ **Definition 4** (Stream)**.** Let m, n be positive integers. A **stream** $D = D(m, n)$ is a sequence of length m, $d_1, d_2, \ldots, d_m$, where each entry is a pair of values in $\{1, \ldots, n\}$.

Let $g : \mathbb{R} \to \mathbb{R}$ be a non-negative, subadditive, and even function where $g(0) = 0$. Frequently, we will need to discuss a matrix where g has been applied to every entry. We use the notations from [12] which are in turn based on notations from [13].

▶ **Definition 5** (Hadamard Function)**.** Given a matrix A of dimensions $n \times n$, a **Hadamard function** g takes as input the matrix A and is applied entrywise to every entry of the matrix. The output is the matrix $g[A]$. Further, we note that the L_1 norm of $g[A]$ is equivalent to the value we aim to approximate, $\|g[A]\|_1 = \sum_{i=1}^{n}\sum_{j=1}^{n} g(a_{ij})$.

We frequently use hash functions in our analysis, we now specify some notation. We sometimes express a hash function H over a domain of size n as a vector of values $(h_1, h_2, ..., h_n)$. Multiplication of two hash functions H^a, H^b is given by the Hadamard product, denoted $H' = HAD(H^a, H^b) = H^a H^b$, where multiplication is performed entrywise so that $(h'_1 = h^a_1 h^b_1, ..., h'_n = h^a_n h^b_n)$.

We now define two additional matrices. All matrices in our definitions are of size $n \times n$, and all vectors are of size $1 \times n$. We denote by $[n]$ the set $\{1, \ldots, n\}$.

▶ **Definition 6** (Sampling Identity Matrix). Given a hash function $H : [n] \to \{0,1\}$, let $h_i = H(i)$. The **Sampling Identity Matrix** I_H with entries b_{ij} is defined as:

$$I_H = \begin{cases} b_{ii} = h_i \\ b_{ij} = 0 \text{ for } i \neq j. \end{cases}$$

That is, the diagonal of I_H corresponds to the values of H. When we multiply matrix I_H by A, each row of $I_H A$ is either the zero vector (corresponding to $h_i = 0$) or the original row i in A (corresponding to $h_i = 1$). We use the term "sampling" due to the fact that the hash functions we use throughout this paper are random, and hence which rows remain untouched is random. The same observations apply to columns when considering the matrix AI_H.

▶ **Definition 7** (Row Aggregation Vector). A **Row Aggregation Vector** J is a $1 \times n$ vector with all entries equal to 1.

Thus, JA yields a vector V where each value v_j is $\sum_{i=1}^{n} a_{ij}$.

▶ **Blackbox Algorithm 1** ($(1 \pm \epsilon')$-Approximation of g on an aggregated matrix).
Input: *Matrix A, and hash function H.*
Output: *$(1 \pm \epsilon')$-Approximation of $\|g[JI_H A]\|_1$ with probability $(1 - \delta_1)$.*

The space Blackbox Algorithm 1 (BA1) uses is referred to as $SPACE1(n, \delta_1, \epsilon')$ in our analysis.

▶ **Blackbox Algorithm 2** ($r(n)$-Approximation of $\|g[I_H A]\|_1$).
Input: *Matrix A, and hash function H.*
Output: *$r(n)$-Approximation of $\|g[I_H A]\|_1$ with probability $(1 - \delta_2)$.*

The space Blackbox Algorithm 2 ($BA2$) uses is referred to as $SPACE2(n, \delta_2)$ in our analysis.

▶ **Definition 8** (Complement Hash Function). For a hash function $H : [n] \to \{0,1\}$, define the **Complement Hash Function** $\bar{H} : [n] \to \{0,1\}$ as $\bar{H}(i) = 1$ if and only if $H(i) = 0$.

▶ **Definition 9** (Threshold Functions). We define two **Threshold Functions**, which we denote by $\rho(n, \epsilon) = O(\frac{r^4(n)}{\epsilon})$ and $\tau(n, \epsilon) = O(\frac{r^2(n)}{\epsilon})$.

▶ **Definition 10** (Weight of a Row). The **weight** of row i in matrix A is given by $u_{A,i} = \sum_{j=1}^{n} a_{ij}$.

▶ **Definition 11** (α-Heavy Rows). Row i is **α-heavy** for $0 < \alpha < 1$ if $u_{A,i} > \alpha \|A\|_1$.

▶ **Definition 12** (Key Row). We say row i is a **Key Row** if: $u_{A,i} > \rho(n, \epsilon)(\|A\|_1 - u_{A,i})$.

While Definition 11 and Definition 12 are similar, we define them for convenience, as our algorithm works by first finding key rows and then building on top of this to find α-heavy rows. We note that, as long as $\rho(n, \epsilon) \geq 1$, a matrix can have at most one key row (since any matrix can have at most $\frac{1}{\alpha}$ α-heavy rows, and a key row is α-heavy for $\alpha = \frac{\rho(n,\epsilon)}{1+\rho(n,\epsilon)}$).

3 Subadditive Approximations

In this section we show that a $(1 \pm \epsilon)$-approximation of even, subadditive, non-negative Hadamard functions which are zero at the origin are preserved under row or column aggregations in the presence of sufficiently heavy rows or columns.

▶ **Theorem 13.** *Let B be an $n \times n$ matrix and let $\epsilon \in (0,1)$ be a parameter. Recall that J is a row vector with all entries equal to 1. Let g be any even, subadditive, non-negative Hadamard function which satisfies $g(0) = 0$. Denote $u_i = \sum_{j=1}^{n} g(b_{ij})$, and thus $\|g[B]\|_1 = \sum_{i=1}^{n} u_i$. If there is a row h such that $u_h \geq (1 - \frac{\epsilon}{2})\|g[B]\|_1$, then $|u_h - \|g[JB]\|_1| \leq \epsilon \|g[JB]\|_1$.*

Proof. Denote $V = JB$. Without loss of generality assume u_1 is the row such that $u_1 \geq (1 - \frac{\epsilon}{2})\|g[B]\|_1$. By subadditivity of g: $\|g[V]\|_1 \leq \|g[B]\|_1 \leq \frac{1}{1-\frac{\epsilon}{2}} u_1 \leq (1+\epsilon)u_1$. Further, we have $b_{1j} = (\sum_{i=1}^{n} b_{ij} - \sum_{i=2}^{n} b_{ij})$. Since g is even and subadditive, and such functions are non-negative, we have $g(b_{1j}) \leq g\left(\sum_{i=1}^{n} b_{ij}\right) + \sum_{i=2}^{n} g(b_{ij})$. Rearranging and summing over j, we get: $\sum_{j=1}^{n} g\left(\sum_{i=1}^{n} b_{ij}\right) \geq \sum_{j=1}^{n} \left(g(b_{1,j}) - \sum_{i=2}^{n} g(b_{ij})\right)$.

Therefore:

$$\|g[V]\|_1 = \sum_{j=1}^{n} g\left(\sum_{i=1}^{n} b_{ij}\right) \geq \sum_{j=1}^{n} \left(g(b_{1,j}) - \left(\sum_{i=2}^{n} g(b_{ij})\right)\right) = u_1 - (\|g[B]\|_1 - u_1).$$

Finally:

$$\|g[V]\|_1 \geq u_1 - (\|g[B]\|_1 - u_1) = 2u_1 - \|g[B]\|_1 \geq u_1 \left(2 - \frac{1}{1-\frac{\epsilon}{2}}\right)$$
$$= u_1 \frac{1-\epsilon}{1-\frac{\epsilon}{2}} \geq u_1(1-\epsilon). \quad \blacktriangleleft$$

4 Algorithm for Finding Key Rows

▶ **Definition 14** (Algorithm for Finding Key Rows)**.**
Input: Matrix A and Sampling Identity Matrix I_H generated from hash function H.
Output: Pair (a, b), where the following holds for a, b, and the matrix $W = I_H A$:

1. The pair is either $(a, b) = (-1, 0)$ or $(a, b) = (i, \tilde{u}_{W,i})$. Here, $\tilde{u}_{W,i}$ is a $(1 \pm \epsilon)$-approximation to $u_{W,i}$ and the index i is the correct corresponding row.
2. If there is a key row i_0 for the matrix W, then $a = i_0$.

Before describing the algorithm and proving its correctness, we prove the following useful lemma in Appendix A.

▶ **Lemma 15.** *Let $U = (u_1, \ldots, u_n)$ be a vector with non-negative entries of dimension n and let H' be a pairwise independent hash function where $H' : [n] \to \{0, 1\}$ and $P[H'(i) = 1] = P[H'(i) = 0] = \frac{1}{2}$. Denote by $\bar{H}'$ the hash function defined by $\bar{H}'(i) = 1 - H'(i)$. Let $X = \sum_i H'(i)u_i$ and $Y = \sum_i \bar{H}'(i)u_i$. If there is no $\frac{1}{16}$-heavy element with respect to U, then:*

$$\Pr\left[\left(X \leq \frac{1}{4} \cdot \|U\|_1\right) \cup \left(Y \leq \frac{1}{4} \cdot \|U\|_1\right)\right] \leq \frac{1}{4}.$$

▶ **Theorem 16.** *If there exist two blackbox algorithms as specified in Blackbox Algorithms 1 and 2, then there exists an algorithm that satisfies the requirements in Definition 14 with high probability.*

Proof. We will prove that Algorithm 1 fits the description of Definition 14. Using standard methods such as in [10], we have a loop that runs in parallel $O(\log(n))$ times so that we can find the index of a heavy element and return it, if there is one. To prove this theorem, we consider the following three exhaustive and disjoint cases regarding the matrix $g[I_H A]$ (recall that $H : [n] \to \{0, 1\}$):

Algorithm 1 Algorithm Find-Key-Row

The algorithm takes as input a matrix A and a hash function $H : [n] \to \{0, 1\}$

for $\ell = 1$ to $N = O(\log n)$ **do**
Generate a pairwise independent, uniform hash function $H_\ell : [n] \to \{0, 1\}$
Let $T_1 = HAD(H, H_\ell)$, $T_0 = HAD(H, \bar{H}_\ell)$
Let $y_1 = BA2(A, T_1)$, $y_0 = BA2(A, T_0)$ ($BA2$ is run with constant failure probability δ_2)
if $y_0 \geq \tau(n, \epsilon) \cdot y_1$ **then**
$b_\ell = 0$
else if $y_1 \geq \tau(n, \epsilon) \cdot y_0$ **then**
$b_\ell = 1$
else
$b_\ell = 2$
if $|\{\ell : b_\ell = 2\}| \geq \frac{2}{5} \cdot N$ **then**
Return $(-1, 0)$
else
if there is a row i such that i satisfies $|\{\ell : H_\ell(i) = b_\ell\}| \geq \frac{3}{4} \cdot N$ **then**
Return $(i, BA1(A, H))$ ($BA1$ is run with $\epsilon' = \frac{\epsilon}{2}$ and δ_1 is set to be inverse polylogarithmic)
else
Return $(-1, 0)$

1. The matrix has a key row (note that a matrix always has at most one key row).
2. The matrix has no α-heavy row for $\alpha = 1 - \frac{\epsilon}{8}$.
3. The matrix has an α-heavy row for $\alpha = 1 - \frac{\epsilon}{8}$, but there is no key row.

We prove that the algorithm is correct in each case in Lemmas 22, 23, and 24, respectively. These proofs can be found in Appendix B. ◀

With the proofs of these three cases, we are done proving that Algorithm 1 performs correctly. We now analyze the space bound for Algorithm 1.

▶ **Lemma 17.** *Algorithm 1 uses* $O\left(SPACE1(n, \delta_1, \frac{\epsilon}{2}) + \log(n)(\log^2(n) + SPACE2(n, \delta_2))\right)$ *bits of memory, where* δ_1 *is inverse polylogarithmic and* δ_2 *is a constant.*

Proof. Note that, in order for our algorithm to succeed, we run $BA1$ with an error parameter of $\epsilon' = \frac{\epsilon}{2}$ and a failure probability parameter δ_1 which is inverse polylogarithmic. Moreover, we run $BA2$ with a constant failure probability. We also require a number of random bits bounded by $O(\log^2(n))$ for generating each hash function H_ℓ, as well as the space required to run $BA2$ in each iteration of the loop. Since there are $O(\log n)$ parallel iterations, this gives the lemma. ◀

4.1 Algorithm for Finding All α-Heavy Rows

Algorithm 1 only guarantees that we return key rows. Given a matrix A, we now show that this algorithm can be used as a subroutine to find all α-heavy rows i with respect to the matrix $g[A]$ with high probability, along with a $(1 \pm \epsilon)$-approximation to the row weights $u_{g[A],i}$ for all i. In order to do this, we apply an additional hash function $H : [n] \to [\tau]$ which essentially maps rows of the matrix to some number of buckets τ (i.e., each bucket

Algorithm 2 Algorithm Find-Heavy-Rows

The algorithm takes as input a matrix A and a value $0 < \alpha < 1$

Generate a pairwise independent hash function $H : [n] \to [\tau]$, where $\tau = O\left(\frac{\rho(n,\epsilon)\log(n)}{\alpha^2}\right)$

for $k = 1$ to τ **do**

Let $H_k : [n] \to \{0, 1\}$ be the function defined by $H_k(i) = 1 \iff H(i) = k$

Let $C_k = \text{Find-Key-Row}(A, H_k)$

Return $\{C_k : C_k \neq (-1, 0)\}$

corresponds to a set of sampled rows based on H), and then run Algorithm 1 for each bucket. The intuition for why the algorithm works is that any α-heavy row i in the original matrix A is likely to be a key row for the matrix in the corresponding bucket to which row i is mapped. Note that, eventually, we find α-heavy rows for $\alpha = \frac{\epsilon^2}{\log^3 n}$. The algorithm sets $\tau = O\left(\frac{\rho(n,\epsilon)\log(n)}{\alpha^2}\right)$ and is given below.

▶ **Theorem 18.** *Algorithm 2 outputs a set of pairs $Q = \{(i_1, a_1), \ldots, (i_t, a_t)\}$ for $t \leq \tau$ which satisfies the following properties, except with probability $\frac{1}{\log n}$:*

1. *$\forall j \in [t]$: $(1 - \epsilon)u_{g[A],i_j} \leq a_j \leq (1 + \epsilon)u_{g[A],i_j}$.*
2. *$\forall i \in [n]$: If row i is α-heavy with respect to the matrix $g[A]$, then $\exists j \in [t]$ such that $i_j = i$ (for any $0 < \alpha < 1$).*

Proof. First, the number of pairs output by Algorithm 2 is at most the number of buckets, which equals τ. Now, the first property is true due to the fact that Algorithm 1 has a high success probability. In particular, as long as the failure probability is at most $\frac{1}{\tau \cdot \log^c(n)}$ for some constant c (which we ensure), then by the union bound the probability that there exists a pair $(i_j, a_j) \in Q$ such that a_j is not a $(1 \pm \epsilon)$-approximation to $u_{g[A],i_j}$ is at most inverse polylogarithmic.

Now, to ensure the second item, we need to argue that every α-heavy row gets mapped to its own bucket with high probability, since if there is a collision the algorithm cannot find all α-heavy rows. Moreover, we must argue that for each α-heavy row i with respect to the matrix $g[A]$, if i is mapped to bucket k by H, then row i is actually a key row in the corresponding sampled matrix $g[A_k]$ (for ease of notation, we write A_k to denote the matrix $H_k A_k$). More formally, suppose row i is α-heavy. Then the algorithm must guarantee with high probability that, if $H(i) = k$, then row i is a key row in the matrix $g[A_k]$. If we prove these two properties, then the theorem holds (since Algorithm 1 outputs a key row with high probability, if there is one).

Observe that there must be at most $\frac{1}{\alpha}$ rows which are α-heavy. In particular, let R be the set of α-heavy rows, and assume towards a contradiction that $|R| > \frac{1}{\alpha}$. Then we have:

$$\|g[A]\|_1 \geq \sum_{i \in R} u_{g[A],i} \geq \sum_{i \in R} \alpha \|g[A]\|_1 = \alpha \cdot \|g[A]\|_1 \cdot |R| > \|g[A]\|_1,$$

which is a contradiction. Hence, we seek to upper bound the probability of a collision when throwing $\frac{1}{\alpha}$ balls into τ bins. By a Birthday paradox argument, this happens with probability at most $\frac{1}{2 \cdot \tau \cdot \alpha^2}$, which can be upper bounded as follows:

$$\frac{1}{2\tau\alpha^2} \leq \frac{\alpha^2}{2\alpha^2 \rho(n,\epsilon)\log(n)} = \frac{1}{2\rho(n,\epsilon)\log(n)} \leq \frac{\epsilon}{2r^4(n)\log(n)},$$

which is inverse polylogarithmically small.

Now, we argue that every α-heavy row i for the matrix $g[A]$ is mapped to a sampled matrix such that i is a key row in the sampled matrix with high probability. In particular, suppose $H(i) = k$, implying that row i is mapped to bucket k. For $\ell \neq i$, let X_ℓ be the indicator random variable which is 1 if and only if row ℓ is mapped to the same bucket as i, namely $H(\ell) = k$ (i.e., $X_\ell = 1$ means the sampled matrix $g[A_k]$ contains row i and row ℓ). If row i is not a key row for the matrix $g[A_k]$, this means that $u_{g[A_k],i} \leq \rho(n,\epsilon)(\|g[A_k]\|_1 - u_{g[A_k],i})$. Observe that, if row i is mapped to bucket k, then we have $u_{g[A_k],i} = u_{g[A],i}$. Hence, the the probability that row i is not a key row for the sampled matrix $g[A_k]$ (assuming row i is mapped to bucket k) can be expressed as $\Pr[u_{g[A],i} \leq \rho(n,\epsilon)(\|g[A_k]\|_1 - u_{g[A],i}) | H(i) = k]$. By pairwise independence of H, and by Markov's inequality, we can write:

$$\begin{aligned}
&\Pr\left[u_{g[A],i} \leq \rho(n,\epsilon)(\|g[A_k]\|_1 - u_{g[A],i}) \;\middle|\; H(i) = k\right] \\
&\quad = \Pr\left[u_{g[A],i} \leq \rho(n,\epsilon)\sum_{\ell \neq i} u_{g[A],\ell} X_\ell \;\middle|\; H(i) = k\right] \\
&\quad = \Pr\left[u_{g[A],i} \leq \rho(n,\epsilon)\sum_{\ell \neq i} u_{g[A],\ell} X_\ell\right] \\
&\quad = \Pr\left[\sum_{\ell \neq i} u_{g[A],\ell} X_\ell \geq \frac{u_{g[A],i}}{\rho(n,\epsilon)}\right] \leq \frac{\rho(n,\epsilon)\mathbb{E}\left[\sum_{\ell \neq i} u_{g[A],\ell} X_\ell\right]}{u_{g[A],i}} \\
&\quad = \frac{\rho(n,\epsilon)\sum_{\ell \neq i} u_{g[A],\ell}}{\tau \cdot u_{g[A],i}} \leq \frac{\rho(n,\epsilon)\|g[A]\|_1}{\alpha\tau\|g[A]\|_1} = \frac{\alpha^2\rho(n,\epsilon)}{4\alpha \cdot \rho(n,\epsilon)\log(n)} \leq \frac{\alpha}{4\log(n)}.
\end{aligned}$$

Here, we choose $\tau = \frac{4\rho(n,\epsilon)\log(n)}{\alpha^2}$, and get that the probability that a particular α-heavy row i is not a key row in its corresponding sampled matrix is at most $\frac{\alpha}{4\log(n)}$. Since there are at most $\frac{1}{\alpha}$ rows which are α-heavy, by the union bound the probability that there exists an α-heavy row that is not a key row in its sampled matrix is at most $\frac{1}{4\log(n)}$.

Thus, in all, the probability that at least one bad event happens (i.e., there exists a pair (i_j, a_j) such that a_j is not a good approximation to $u_{g[A],i_j}$, there is a collision between α-heavy rows, or an α-heavy row is not a key row in its corresponding sampled matrix) is at most $\frac{1}{\log(n)}$. This gives the theorem. ◀

4.2 Sum from α-Heavy Rows

We now have an algorithm that is able to find all α-heavy rows for $\alpha = \frac{\epsilon^2}{\log^3 n}$, except with probability $\frac{1}{\log n}$. In the language of [11], by Theorem 18, our α-heavy rows algorithm outputs an (α, ϵ)-cover with respect to the vector $(u_{g[A],1}, u_{g[A],2}, \ldots, u_{g[A],n})$ except with probability $\frac{1}{\log n}$, where $\epsilon > 0$ and $\alpha > 0$. Hence, we can apply the Recursive Sum algorithm from [11] (see Appendix C for the formal definition of an (α, ϵ)-cover, along with the Recursive Sum algorithm) to get a $(1 \pm \epsilon)$-approximation of $\|g[A]\|_1$. Note that the Recursive Sum algorithm needs $\alpha = \frac{\epsilon^2}{\log^3 n}$ and a failure probability of at most $\frac{1}{\log n}$, which we provide. Hence, we get the following theorem.

▶ **Theorem 19.** *The Recursive Sum Algorithm, using Algorithm 2 as a subroutine, returns a $(1 \pm \epsilon)$-approximation of $\|g[A]\|_1$.*

4.3 Space Bounds

▶ **Lemma 20.** *Recursive Sum, using Algorithm 2 as a subroutine as described in Section 4.2, uses the following amount of memory, where $\epsilon' = \frac{\epsilon}{2}$, δ_1 is inverse polylogarithmic, and δ_2 is a small constant:*

$$O\left(\frac{r^4(n)}{\epsilon^5}(\log^{10}(n) + \log^8(n)SPACE1(n, \delta_1, \epsilon') + \log^9(n)SPACE2(n, \delta_2))\right).$$

Proof. The final algorithm uses the space bound from Lemma 17, multiplied by $\tau = O\left(\frac{\rho(n,\epsilon)\log(n)}{\alpha^2}\right)$, where $\alpha = \frac{\epsilon^2}{\phi^3}$, $\phi = O(\log n)$, and $\rho(n, \epsilon) = O(\frac{r^4(n)}{\epsilon})$. This gives $\tau = \frac{1}{\epsilon^5} r^4(n) \log^7(n)$ to account for the splitting required to find α-heavy rows in Section 4.1. Finally, a multiplicative cost of $\log(n)$ is needed for the Recursive Sum algorithm, giving the final bound. ◀

5 Applications

We now apply our algorithm to the problem of determining the L_1 distance between the joint and product of the marginal distributions as described in Problem 1.

Space Bounds for Determining L_1 Independence

Given an $n \times n$ matrix A with entries $a_{ij} = \frac{f_{ij}}{m} - \frac{p_i q_j}{m}$, we have provided a method to approximate the value $\|g[A]\|_1$:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} g\left(\frac{f_{ij}}{m} - \frac{p_i q_j}{m}\right).$$

Let g be the L_1 distance, namely $g(x) = |x|$ (hence, the $(i,j)^{th}$ entry in $g[A]$ is given by $|\frac{f_{ij}}{m} - \frac{p_i q_j}{m}|$). we now state explicitly which blackbox algorithms we use:

- Let Blackbox Algorithm 1 ($BA1$) be the $(1 \pm \epsilon')$-approximation of L_1 for vectors from [14]. The space of this algorithm is upper bounded by the number of random bits required and uses $O(\log(\frac{nm}{\delta_1 \epsilon'})\log(\frac{m}{\delta_1 \epsilon'})\log(\frac{1}{\delta_1})\epsilon'^{-2})$ bits of memory.
- Let Blackbox Algorithm 2 ($BA2$) be the $r(n)$-approximation, using the L_1 sketch of the distance between the joint and product of the marginal distributions from [15]. This algorithm does not have a precise polylogarithmic bound provided, but we compute that it is upper bounded by the random bits required to generate the Cauchy random variables similarly to $BA1$ (which are generated in parallel $O(\log \frac{1}{\delta_2})$ times). This algorithm requires $O(\log(\frac{nm}{\delta_1 \epsilon'})\log(\frac{m}{\delta_1 \epsilon'})\log(\frac{1}{\delta_1})\log(\frac{1}{\delta_2})\epsilon'^{-2})$ bits of memory. Note that $BA2$ does not depend on ϵ', δ_1, but we are stating a loose upper bound.

These two algorithms match the definitions given in Section 2, and thus we are able to give a bound of $O(\frac{1}{\epsilon^7}\log^{14}(n)\log^2(\frac{nm}{\epsilon}))$ on the space our algorithm requires (recall that we set $\epsilon' = \theta(\epsilon)$, δ_2 to be some small constant, and δ_1 to be inverse polylogarithmically small). We can improve this slightly as follows.

▶ **Corollary 21.** *Due to the nature of the truncated Cauchy distribution (see [15]), we can further improve our space bound to $O\left(\frac{1}{\epsilon^7}\log^{12}(n)\log^2(\frac{nm}{\epsilon})\right)$.*

Proof. Due to the constant lower bound on the approximation of L_1, instead of $\frac{1}{r^2(n)} \leq \|g[W]\|_1 \leq r^2(n)$, we get $C \leq \|g[W]\|_1 \leq \log^2(n)$ for some constant C. As the space cost from dividing the matrix into submatrices as shown in Section 4.1 directly depends on these bounds, we only pay an $O(r^2(n))$ multiplicative factor instead of an $O(r^4(n))$ multiplicative factor and achieve a bound of $O\left(\frac{1}{\epsilon^7}\log^{12}(n)\log^2(\frac{nm}{\epsilon})\right)$. ◀

References

1. Noga Alon, Alexandr Andoni, Tali Kaufman, Kevin Matulef, Ronitt Rubinfeld, and Ning Xie. Testing k-wise and almost k-wise independence. In *Proceedings of the 39th annual ACM Symposium on Theory of Computing*, 2007.
2. Noga Alon, Oded Goldreich, and Yishay Mansour. Almost k-wise independence versus k-wise independence. *Information Processing Letters*, 88(3):107–110, 2003.
3. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the 28th annual ACM Symposium on Theory of Computing*, 1996.
4. Tuğkan Batu, Eldar Fischer, Lance Fortnow, Ravi Kumar, Ronitt Rubinfeld, and Patrick White. Testing random variables for independence and identity. In *Proceedings of the 42nd annual IEEE Symposium on Foundations of Computer Science*, 2001.
5. Tuğkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren D. Smith, and Patrick White. Testing that distributions are close. In *Proceedings of the 41st annual IEEE Symposium on Foundations of Computer Science*, 2000.
6. Tuğkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren D. Smith, and Patrick White. Testing closeness of discrete distributions. *Journal of the ACM*, 60(1):4, 2013.
7. Tuğkan Batu, Ravi Kumar, and Ronitt Rubinfeld. Sublinear algorithms for testing monotone and unimodal distributions. In *Proceedings of the 36th annual ACM Symposium on Theory of Computing*, 2004.
8. Vladimir Braverman, Kai-Min Chung, Zhenming Liu, Michael Mitzenmacher, and Rafail Ostrovsky. AMS Without 4-Wise Independence on Product Domains. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science*, 2010.
9. Vladimir Braverman and Rafail Ostrovsky. Measuring independence of datasets. In *Proceedings of the 42nd annual ACM Symposium on Theory of Computing*, 2010.
10. Vladimir Braverman and Rafail Ostrovsky. Zero-one frequency laws. In *Proceedings of the 42nd annual ACM Symposium on Theory of Computing*, 2010.
11. Vladimir Braverman and Rafail Ostrovsky. Generalizing the layering method of Indyk and Woodruff: Recursive sketches for frequency-based vectors on streams. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*. Springer, 2013.
12. Dominique Guillot, Apoorva Khare, and Bala Rajaratnam. Complete characterization of hadamard powers preserving loewner positivity, monotonicity, and convexity. *Journal of Mathematical Analysis and Applications*, 425(1):489–507, 2015.
13. Roger A. Horn and Charles R. Johnson. Topics in matrix analysis. *Cambridge University Press, Cambridge*, 1991.
14. Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM*, 53(3):307–323, 2006.
15. Piotr Indyk and Andrew McGregor. Declaring independence via the sketching of sketches. In *Proceedings of the 19th annual ACM-SIAM Symposium on Discrete Algorithms*, 2008.
16. Ralph Kimball and Joe Caserta. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. John Wiley & Sons, 2004.
17. Erich L. Lehmann and Joseph P. Romano. *Testing statistical hypotheses*. Springer Science & Business Media, 2006.
18. Andrew McGregor and Hoa T. Vu. Evaluating bayesian networks via data streams. In *Proceedings of the 21st International Computing and Combinatorics Conference*, 2015.
19. Viswanath Poosala and Yannis E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, 1997.

20 Ronitt Rubinfeld and Rocco A. Servedio. Testing monotone high-dimensional distributions. In *Proceedings of the 37th annual ACM Symposium on Theory of Computing*, 2005.

21 Amit Sahai and Salil Vadhan. A complete problem for statistical zero knowledge. *Journal of the ACM*, 50(2):196–249, 2003.

22 David P. Woodruff. Sketching as a tool for numerical linear algebra. *CoRR*, abs/1411.4357, 2014. URL: `http://arxiv.org/abs/1411.4357`.

A Proof of Lemma 15

Proof. Note that we always have the equality $X+Y = \sum_i H'(i)u_i + \bar{H}'(i)u_i = \sum_i H'(i)u_i + (1-H'(i))u_i = \|U\|_1$, and moreover $\mathbb{E}[X] = \sum_i u_i \mathbb{E}[H'(i)] = \frac{1}{2}\cdot\|U\|_1$. Also, observe that

$$\begin{aligned} Var[X] &= \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \\ &= \sum_i \mathbb{E}[(H'(i))^2]u_i^2 + \sum_{i\neq j}\mathbb{E}[H'(i)H'(j)]u_iu_j - \frac{1}{4}\cdot\|U\|_1^2 \\ &= \frac{1}{2}\sum_i u_i^2 + \frac{1}{4}\sum_{i\neq j} u_iu_j - \frac{1}{4}\left(\sum_i u_i^2 + \sum_{i\neq j} u_iu_j\right) = \frac{1}{4}\sum_i u_i^2. \end{aligned}$$

Using the fact that there is no $\frac{1}{16}$-heavy element with respect to U, which implies that $u_i \leq \frac{1}{16}\cdot\|U\|_1$ for all i, we have:

$$Var[X] = \frac{1}{4}\sum_i u_i^2 \leq \frac{\|U\|_1}{64}\sum_i u_i = \frac{\|U\|_1^2}{64}.$$

Now we can apply Chebyshev's inequality to obtain:

$$\begin{aligned} \Pr\left[\left(X \leq \frac{1}{4}\cdot\|U\|_1\right) \cup \left(Y \leq \frac{1}{4}\cdot\|U\|_1\right)\right] &= \Pr\left[|X-\mathbb{E}[X]| \geq \frac{\|U\|_1}{4}\right] \\ &\leq \frac{16\cdot Var[X]}{\|U\|_1^2} \leq \frac{16\cdot\|U\|_1^2}{64\cdot\|U\|_1^2} = \frac{1}{4}. \end{aligned}$$

◀

B Proof of Correctness of Algorithm 1

Throughout the lemmas, we imagine that the hash function $H : [n] \to \{0,1\}$ is fixed, and hence the matrix $g[I_H A]$ is fixed. All randomness is taken over the pairwise independent hash functions H_ℓ that are generated in parallel, along with both blackbox algorithms.

To ease the notation, we define

$$W = I_H A,\ W_1 = I_{T_1} A, \text{ and } W_0 = I_{T_2} A$$

(recall the notation from Algorithm 1 that $T_1 = HAD(H, H_\ell)$ and $T_0 = HAD(H, \bar{H}_\ell)$). Finally, for each row i in the matrix $g[W]$, we define the shorthand notation $u_i = u_{g[W],i}$.

▶ **Lemma 22.** *If the matrix $g[I_H A]$ has a key row, Algorithm 1 correctly returns the index of the row and a $(1\pm\epsilon)$-approximation of the weight of the key row except with inverse polylogarithmic probability.*

Proof. Suppose the matrix $g[I_H A]$ has a key row, and let i_0 be the index of this row. We prove that we return a good approximation of $u_{g[W],i_0}$ with high probability. In particular, we first argue that, for a fixed iteration ℓ of the loop, we have the property that b_ℓ equals

$H_\ell(i_0)$, and moreover this holds with high probability. We assume without loss of generality that $H_\ell(i_0) = 1$ (the case when $H_\ell(i_0) = 0$ is symmetric). In particular, this implies that the key row i_0 appears in the matrix $g[W_1]$.

By definition of $BA2$, the following holds for $y_1 = BA2(A, T_1)$ and $y_0 = BA2(A, T_0)$, except with probability $2\delta_2$ (where δ_2 is the failure probability of $BA2$):

$$y_1 \geq \frac{\|g[W_1]\|_1}{r(n)} \text{ and } y_0 \leq \|g[W_0]\|_1 r(n).$$

We have the following set of inequalities:

$$\|g[W_1]\|_1 \geq u_{i_0} > \rho(n,\epsilon)(\|g[W]\|_1 - u_{i_0}) \geq \rho(n,\epsilon)\|g[W_0]\|_1,$$

where the first inequality follows since g is non-negative and the key row i_0 appears in the matrix $g[W_1]$ (and hence the L_1 norm of $g[W_1]$ is at least u_{i_0} since it includes the row i_0), the second inequality follows by definition of i_0 being a key row for the matrix W, and the last inequality follows since the entries in row i_0 of the matrix W_0 are all zero (as $H_\ell(i_0) = 1$) and the remaining rows of W_0 are sampled from W, along with the facts that g is non-negative and $g(0) = 0$.

Substituting for $\rho(n,\epsilon)$, and using the fact that y_1 and y_0 are good approximations for $\|g[W_1]\|_1$ and $\|g[W_0]\|_1$ (respectively), except with probability $2\delta_2$, we get:

$$y_1 \geq \frac{\|g[W_1]\|_1}{r(n)} > \frac{\rho(n,\epsilon)}{r(n)} \cdot \|g[W_0]\|_1 \geq \frac{\rho(n,\epsilon)}{r^2(n)} \cdot y_0 \geq \tau(n,\epsilon) \cdot y_0.$$

Thus, in this iteration of the loop we have $b_\ell = 1$ except with probability $2\delta_2$ (in the case that $H_\ell(i_0) = 0$, it is easy to verify by a similar argument that $y_0 \geq \tau(n,\epsilon) \cdot y_1$, and hence we have $b_\ell = 0$). Hence, for the row i_0, we have the property that $b_\ell = H_\ell(i_0)$ for a fixed ℓ, except with probability $2\delta_2$. By the Chernoff bound, as long as δ_2 is a sufficiently small constant, we have $b_\ell = H_\ell(i_0)$ for at least a $\frac{3}{4}$-fraction of iterations ℓ, except with inverse polynomial probability. The only issue to consider is the case that there exists another row $i \neq i_0$ with the same property, namely $b_\ell = H_\ell(i)$ for a large fraction of iterations ℓ. However, if $b_\ell = H_\ell(i)$, it must be that at least one of y_1, y_0 is a bad approximation or $H_\ell(i) = H_\ell(i_0)$, which happens with probability at most $2\delta_2 + \frac{1}{2}$. Therefore, by the Chernoff bound, the probability that this happens for at least a $\frac{3}{4}$-fraction of iterations ℓ is at most $\frac{1}{2^{O(\log n)}}$, which is inverse polynomially small. By applying the union bound, the probability that there exists such a row is at most $\frac{n-1}{2^{O(\log n)}}$, which is at most an inverse polynomial. Hence, in this case, the algorithm returns $(i_0, BA1(A, H))$ except with inverse polynomial probability.

We now argue that $\tilde{u}_{g[W],i_o} = BA1(A, H)$ is a $(1 \pm \epsilon)$-approximation of $u_{g[W],i_0}$, except with inverse polylogarithmic probability. By definition of $BA1$, which we run with an error parameter of $\epsilon' = \frac{\epsilon}{2}$, it returns a $\left(1 \pm \frac{\epsilon}{2}\right)$-approximation of $\|g[JW]\|_1$ except with inverse polylogarithmic probability, where $W = I_H A$. Moreover, since i_0 is a key row, we have:

$$u_{i_0} > \rho(n,\epsilon)(\|g[W]\|_1 - u_{i_0}) \Rightarrow u_{i_0} > \frac{\rho(n,\epsilon)\|g[W]\|_1}{1 + \rho(n,\epsilon)} \geq \left(1 - \frac{\epsilon}{8}\right)\|g[W]\|_1,$$

where the last inequality follows as long as $r^4(n) \geq 8 - \epsilon$. This implies that i_0 is $\left(1 - \frac{\epsilon}{8}\right)$-heavy with respect to the matrix $g[W]$, and hence we can apply Theorem 13 to get that:

$$\begin{aligned}(1+\epsilon)u_{i_0} \geq \frac{\left(1+\frac{\epsilon}{2}\right)}{\left(1-\frac{\epsilon}{4}\right)}u_{i_0} &\geq \left(1+\frac{\epsilon}{2}\right)\|g[JW]\|_1 \geq \tilde{u}_{g[W],i_0} \\ &\geq \left(1-\frac{\epsilon}{2}\right)\|g[JW]\|_1 \geq \frac{\left(1-\frac{\epsilon}{2}\right)}{\left(1+\frac{\epsilon}{4}\right)}u_{i_0} \geq (1-\epsilon)u_{i_0}.\end{aligned}$$

The first inequality holds for any $0 < \epsilon \leq 1$, the second inequality holds by Theorem 13, the third inequality holds since $\tilde{u}_{g[W],i_0}$ is a $\left(1 \pm \frac{\epsilon}{2}\right)$-approximation of $\|g[JW]\|_1$, and the rest hold for similar reasons. Hence, our algorithm returns a good approximation as long as $BA1$ succeeds. Noting that this happens except with inverse polylogarithmic probability gives the lemma. ◀

▶ **Lemma 23.** *If the input matrix has no α-heavy row, where $\alpha = 1 - \frac{\epsilon}{8}$, then with high probability Algorithm 1 correctly returns $(-1, 0)$.*

Proof. In this case, we have no α-heavy row for $\alpha = 1 - \frac{\epsilon}{8}$, which implies that $u_i \leq \alpha\|g[W]\|_1 = \left(1 - \frac{\epsilon}{8}\right)\|g[W]\|_1$ for each row i in the matrix $g[W]$. In this case, we show the probability that Algorithm 1 returns a false positive is small. That is, with high probability, in each iteration ℓ of the loop the algorithm sets $b_\ell = 2$, and hence it returns $(-1, 0)$. We split this case into three additional disjoint and exhaustive subcases, defined as follows:

1. For each row i, we have $u_i \leq \frac{1}{16}\|g[W]\|_1$.
2. There exists a row i with $u_i > \frac{1}{16}\|g[W]\|_1$ and $\forall j \neq i$ we have $u_j \leq \frac{\epsilon}{128}u_i$.
3. There exist two distinct rows i, j where $u_i > \frac{1}{16}\|g[W]\|_1$ and $u_j > \frac{\epsilon}{128}u_i$.

We define $X = \sum_i h_i^\ell u_i$ and $Y = \sum_i \bar{h}_i^\ell u_i$, where $h_i^\ell = H_\ell(i)$ and $\bar{h}_i^\ell = \bar{H}_\ell(i)$. Hence, we have $X = \|g[W_1]\|_1$ and $Y = \|g[W_0]\|_1$, and moreover $X + Y = \|g[W]\|_1$ (recall that $g[W_1] = g[I_{T_1}A]$ and $g[W_0] = g[I_{T_0}A]$).

In the first subcase, where there is no $\frac{1}{16}$-heavy row, we can apply Lemma 15 to the vector $(u_1, \ldots, u_n)$ to get that:

$$\Pr\left[\left(X \leq \frac{\|g[W]\|_1}{4}\right) \cup \left(Y \leq \frac{\|g[W]\|_1}{4}\right)\right] \leq \frac{1}{4}.$$

By definition of $BA2$, the following holds for $y_1 = BA2(A, T_1)$ and $y_0 = BA2(A, T_0)$ except with probability $2\delta_2$, where δ_2 is the success probability of $BA2$:

$$\frac{\|g[W_1]\|_1}{r(n)} \leq y_1 \leq r(n)\|g[W_1]\|_1\ , \quad \frac{\|g[W_0]\|_1}{r(n)} \leq y_0 \leq r(n)\|g[W_0]\|_1.$$

Hence, except with probability $\frac{1}{4} + 2\delta_2$, we have the following constraints on y_0 and y_1:

$$y_0 \leq r(n)Y \leq r(n) \cdot \frac{3}{4} \cdot \|g[W]\|_1 \leq 3r(n)X \leq 3y_1 r^2(n) \leq \tau(n, \epsilon) \cdot y_1, \text{ and}$$
$$y_1 \leq r(n)X \leq r(n) \cdot \frac{3}{4} \cdot \|g[W]\|_1 \leq 3r(n)Y \leq 3y_0 r^2(n) \leq \tau(n, \epsilon) \cdot y_0,$$

in which case we set $b_\ell = 2$. If δ_2 is some small constant, say $\delta_2 \leq \frac{1}{32}$, then for a fixed iteration ℓ, we set $b_\ell = 2$ except with probability $\frac{5}{16}$. Now, applying the Chernoff bound, we can show that the probability of having more than a $\frac{2}{5}$-fraction of iterations ℓ with $b_\ell \neq 2$ is at most an inverse polynomial. Hence, in this subcase the algorithm outputs $(-1, 0)$, except with inverse polynomial probability.

In the second subcase, we have $u_i > \frac{1}{16}\|g[W]\|_1$ and, for all $j \neq i$, $u_j \leq \frac{\epsilon}{128}u_i$. Then, since u_i is not $\left(1 - \frac{\epsilon}{8}\right)$-heavy with respect to $g[W]$, we have:

$$u_j \leq \frac{\epsilon}{128} \cdot u_i \leq \frac{1}{16}(\|g[W]\|_1 - u_i).$$

Hence, we can apply Lemma 15 to the vector $U = (u_1, \ldots, u_{i-1}, 0, u_{i+1}, \ldots, u_n)$ (since $\|U\|_1 = \|g[W]\|_1 - u_i$, and moreover each entry in U is at most $\frac{1}{16}\|U\|_1$). Letting $X' = \sum_{j \neq i} h_j^\ell u_j$ and $Y' = \sum_{j \neq i} \bar{h}_j^\ell u_j$, we get that:

$$\Pr\left[\left(X' \leq \frac{1}{4} \cdot \|U\|_1\right) \cup \left(Y' \leq \frac{1}{4} \cdot \|U\|_1\right)\right] \leq \frac{1}{4}.$$

This implies that $X \geq X' > \frac{1}{4}(\|g[W]\|_1 - u_i) \geq \frac{\epsilon}{32}\|g[W]\|_1$ and $Y \geq Y' > \frac{1}{4}(\|g[W]\|_1 - u_i) \geq \frac{\epsilon}{32}\|g[W]\|_1$. Moreover, except with probability $2\delta_2$, y_1 and y_0 are good approximations to $\|g[W_1]\|_1$ and $\|g[W_0]\|_1$, respectively. Thus, except with probability $\frac{1}{4} + 2\delta_2$, we have:

$$\begin{aligned} y_0 \leq r(n)Y \leq r(n)\left(1 - \frac{\epsilon}{32}\right)\|g[W]\|_1 &\leq r(n)\left(1 - \frac{\epsilon}{32}\right)\cdot\frac{32}{\epsilon}\cdot X \\ &\leq \frac{32r^2(n)}{\epsilon}\cdot y_1 \leq \tau(n,\epsilon)\cdot y_1, \text{ and} \\ y_1 \leq r(n)X \leq r(n)\left(1 - \frac{\epsilon}{32}\right)\|g[W]\|_1 &\leq r(n)\left(1 - \frac{\epsilon}{32}\right)\cdot\frac{32}{\epsilon}\cdot Y \\ &\leq \frac{32r^2(n)}{\epsilon}\cdot y_0 \leq \tau(n,\epsilon)\cdot y_0. \end{aligned}$$

This implies that, for a fixed iteration ℓ, the algorithm sets $b_\ell = 2$ except with probability $\frac{1}{4} + 2\delta_2$. Applying the Chernoff bound again, we see that the probability of having more than a $\frac{2}{5}$-fraction of iterations ℓ with $b_\ell \neq 2$ is at most an inverse polynomial. Thus, in this subcase, the algorithm outputs $(-1, 0)$ except with inverse polynomial probability.

We now consider the last subcase, where $u_i > \frac{1}{16}\|g[W]\|_1$ and there exists $j \neq i$ such that $u_j > \frac{\epsilon}{128}u_i$. Note that the probability that i and j get mapped to different matrices is given by $\Pr[H_\ell(i) \neq H_\ell(j)] = \frac{1}{2}$. Assume without loss of generality that $H_\ell(j) = 1$ (the case that $H_\ell(j) = 0$ is symmetric). In the event that i and j get mapped to different matrices and y_1, y_0 are good approximations to $\|g[W_1]\|_1, \|g[W_0]\|_1$ respectively, which happens with probability at least $\frac{1}{2} - 2\delta_2$, we have:

$$\begin{aligned} y_1 &\geq \frac{X}{r(n)} \geq \frac{u_j}{r(n)} \geq \frac{\epsilon}{128r(n)}\cdot u_i \geq \frac{\epsilon}{128r(n)}\cdot\frac{1}{16}\cdot\|g[W]\|_1 \\ &\geq \frac{\epsilon}{2048r(n)}\cdot Y \geq \frac{\epsilon}{2048r^2(n)}\cdot y_0 \implies y_0 \leq \frac{2048r^2(n)}{\epsilon}\cdot y_1 \leq \tau(n,\epsilon)\cdot y_1, \text{ and} \\ y_0 &\geq \frac{Y}{r(n)} \geq \frac{u_i}{r(n)} \geq \frac{\epsilon}{128r(n)}\cdot u_i \geq \frac{\epsilon}{128r(n)}\cdot\frac{1}{16}\cdot\|g[W]\|_1 \\ &\geq \frac{\epsilon}{2048r(n)}\cdot X \geq \frac{\epsilon}{2048r^2(n)}\cdot y_1 \implies y_1 \leq \frac{2048r^2(n)}{\epsilon}\cdot y_0 \leq \tau(n,\epsilon)\cdot y_0. \end{aligned}$$

Thus, except with probability at most $\frac{1}{2} + 2\delta_2$, the algorithm sets $b_\ell = 2$ for each iteration ℓ. We apply the Chernoff bound again to get that $b_\ell = 2$ for at least a $\frac{2}{5}$-fraction of iterations, except with inverse polynomial probability. Hence, the algorithm outputs $(-1, 0)$ except with inverse polynomial probability. ◀

▶ **Lemma 24.** *If the matrix $g[I_H A]$ does not have a key row but has an α-heavy row i_0, where $\alpha = 1 - \frac{\epsilon}{8}$, then Algorithm 1 either returns $(-1, 0)$ or returns a $(1 \pm \epsilon)$-approximation of $u_{I_H A, i_0}$ and the corresponding row i_0 with high probability.*

Proof. We know there is an α-heavy row, but not a key row. Note that there cannot be more than one α-heavy row for $\alpha = 1 - \frac{\epsilon}{8}$. If the algorithm returns $(-1, 0)$, then the lemma holds (note the algorithm is allowed to return $(-1, 0)$ since there is no key row). If the algorithm returns a pair of the form $(i, BA1(A, H))$, we know from Theorem 13 that the approximation of the weight of the α-heavy row is a $(1 \pm \epsilon)$-approximation of $\|g[W]\|_1$ as long as $BA1$ succeeds, which happens except with inverse polylogarithmic probability (the argument that the approximation is good follows similarly as in Lemma 22). We need only argue that we return the correct index, i_0. Again, the argument follows similarly as in Lemma 22. In particular, if $H_\ell(i) = b_\ell$ for a fixed iteration ℓ, then at least one of y_0, y_1 is

a bad approximation or $H_\ell(i_0) = H_\ell(i)$, which happens with probability at most $2\delta_2 + \frac{1}{2}$ (where δ_2 is the failure probability of $BA2$). We then apply the Chernoff bound, similarly as before. ◀

With Lemmas 22, 23, and 24, we are done proving that Algorithm 1 fits the description of Definition 14, except with inverse polylogarithmic probability.

C Recursive Sketches

In this section, we give relevant notation and describe the Recursive Sum algorithm found in [11]. We first give some definitions which will be useful when describing the algorithm, the first of which will help us define a cover.

▶ **Definition 25.** Let Ω be a finite set of real numbers. For any positive integer t, we define $Pairs_t$ to be the set of all sets of pairs of the form:

$$\{(i_1, w_1), \ldots, (i_t, w_t)\}, \quad \text{where } 1 \leq i_1 < i_2 < \cdots < i_t \leq n, i_j \in \mathbb{Z}, w_j \in \Omega.$$

We also further define

$$Pairs = \emptyset \cup \left(\bigcup_{t=1}^{n} Pairs_t \right).$$

We now provide the definition of a cover.

▶ **Definition 26.** We say a non-empty set $Q \in Pairs_t$ for some $t \in [n]$ (i.e., $Q = \{(i_1, w_1), \ldots, (i_t, w_t)\}$) is an (α, ϵ)-cover with respect to the vector $V = (v_1, \ldots, v_n)$ (where each $v_i \geq 0$) if the following is true:

1. $\forall j \in [t]$: $(1 - \epsilon)v_{i_j} \leq w_j \leq (1 + \epsilon)v_{i_j}$.
2. $\forall i \in [n]$: If v_i is α-heavy then $\exists j \in [t]$ such that $i_j = i$ (here, α-heavy means $v_i \geq \alpha \sum_j v_j$).

We also define the following index set, and some other notation that is useful for the algorithm.

▶ **Definition 27.** For a non-empty set $Q \in Pairs$, we define $Ind(Q)$ to be the set of indices of Q. More formally, for $Q \in Pairs$, we let $Ind(Q) = \{i : \exists j \leq t$ such that, for the j^{th} pair (i_j, w_j) of Q, we have $i_j = i\}$. For $i \in Ind(Q)$, we denote by $w_Q(i)$ the corresponding approximation. More formally, if $i = i_j$, then $w_Q(i) = w_j$. Note that, since $i_j < i_{j+1}$, this is a valid definition. For completeness, we let $w_Q(i) = 0$ for $i \notin Ind(Q)$ and $Ind(\emptyset) = \emptyset$.

▶ **Definition 28.** We say $H : [n] \to \{0, 1\}$ is a pairwise independent zero-one vector if the zero-one entries are uniformly distributed and pairwise independent. In particular, we have $\Pr[H(i) = 0] = \Pr[H(i) = 1] = \frac{1}{2}$, and moreover the entries are pairwise independent.

Using notation from [11], for a vector $V = (v_1, \ldots, v_n)$, we let $|V|$ denote the L_1 norm of V, $|V| = \sum_{i=1}^{n} v_i$. Note that the product of two pairwise independent zero-one vectors H_1 and H_2 is simply given by the Hadamard product. Moreover, we let F_0 denote the 0^{th} frequency moment, so that $F_0(V)$ counts the number of distinct elements in the vector V. In the following algorithm, we let $HH(D, \alpha, \epsilon, \delta)$ be an algorithm that produces an (α, ϵ)-cover with respect to the vector $V = V(D)$ with probability at least $1 - \delta$ (where $V(D)$ is a vector of dimensionality n defined by the stream D). For some integer parameter ϕ, let $H_1, \ldots, H_\phi$ be i.i.d. random vectors with zero-one entries that are uniformly distributed and pairwise independent. We define vectors V_j for $0 \leq j \leq \phi$ via the following iterative process: $V_0 = V$,

Algorithm 3 Recursive Sum (D, ϵ)

1. Generate $\phi = O(\log(n))$ pairwise independent zero-one vectors $H_1, \ldots, H_\phi$. Denote by D_j the stream $D_{H_1 H_2 \cdots H_\phi}$
2. Compute, in parallel, $Q_j = HH(D_j, \frac{\epsilon^2}{\phi^3}, \epsilon, \frac{1}{\phi})$
3. If $F_0(V_\phi) > 10^{10}$ then output 0 and stop. Otherwise, compute precisely $Y_\phi = |V_\phi|$
4. For each $j = \phi - 1, \ldots, 0$, compute

$$Y_j = 2Y_{j+1} - \sum_{i \in Ind(Q_j)} (1 - 2h_i^j) w_{Q_j}(i)$$

5. Output Y_0

and $V_j = HAD(V_{j-1}, H_j)$ for $j = 1, \ldots, \phi$. We denote by h_i^j the i^{th} entry of H_j. For a function $H : [n] \to \{0, 1\}$, define D_H to be a substream of the stream D that contains only elements $i \in D$ such that $H(i) = 1$.

We are now ready to define the Recursive Sum algorithm (Algorithm 6 from [11]). Theorem 4.1 from [11]:

▶ **Theorem 29.** *Algorithm 3 computes a* $(1 \pm \epsilon)$*-approximation of* $|V|$ *and errs with probability at most* 0.3. *The algorithm uses* $O(\log(n)\mu(n, \frac{\epsilon^2}{\log^3(n)}, \epsilon, \frac{1}{\log(n)}))$ *bits of memory, where* μ *is the space required by the above algorithm* HH.