

# Adaptive Matrix Vector Product

Santosh S. Vempala\*

David P. Woodruff†

November 6, 2016

## Abstract

We consider the following streaming problem: given a hardwired  $m \times n$  matrix  $A$  together with a  $\text{poly}(mn)$ -bit hardwired string of advice that may depend on  $A$ , for any  $x$  with coordinates  $x_1, \dots, x_n$  presented in order, output the coordinates of  $A \cdot x$  in order. Our focus is on using as little memory as possible while computing  $A \cdot x$ ; we do not count the size of the output tape on which the coordinates of  $A \cdot x$  are written; for some matrices  $A$  such as the identity matrix, a constant number of words of space is achievable. Such an algorithm has to adapt its memory contents to the changing structure of  $A$  and exploit it on the fly. We give a nearly tight characterization, for any number of passes over the coordinates of  $x$ , of the space complexity of such a streaming algorithm. Our characterization is constructive, in that we provide an efficient algorithm matching our lower bound on the space complexity. The essential parameters, *streaming rank* and *multi-pass streaming rank* of  $A$ , might be of independent interest, and we show they can be computed efficiently. We give several applications of our results to computing Johnson-Lindenstrauss transforms. Finally, we note that we can characterize the optimal space complexity when the coordinates of  $A \cdot x$  can be written in any order.

## 1 Introduction

Computing a matrix-vector product is a fundamental and ubiquitous primitive in numerical linear algebra and iterative algorithms in general, e.g., in algorithms for computing spectral quantities and solving regression problems. In many applications, the matrix of interest  $A$  of size  $m \times n$  is fixed in advance, e.g., it might be the adjacency matrix or Laplacian of a graph, or its rows might correspond to a linear code. The problem is to compute the product of this matrix with an unknown vector (or series of vectors) whose coordinates are presented in a stream using as little working memory as possible. Our main goal is to understand the com-

plexity of the matrix-vector product when the matrix is fixed and the coordinates of the vector  $x$  are presented as a data stream in the order  $x_1, \dots, x_n$ . Thus, the algorithm knows the matrix  $A$  in advance, but has to be prepared to compute  $A \cdot x$  for an arbitrary, unknown  $x$ .

**Problem:** *What is the complexity (space and time) of computing the coordinates of  $A \cdot x$  in the correct order, when  $A$  of size  $m \times n$  is known in advance and  $x$  is presented as  $x_1, \dots, x_n$ ?*

By space complexity, we mean the size of the working memory, measured in  $O(\log(mn))$ -bit words, and do not count the space used to write the output. The latter space is not used for any computations. We emphasize that we are interested in the complexity of the problem for each matrix  $A$ , i.e., a per-instance analysis, not just the worst case over all possible matrices in terms of their size.

This streaming complexity problem has several motivations. First, one can view the matrix as being hardwired, and special hardware of size  $O(mn)$  (or perhaps  $\text{poly}(mn)$ ) is built for very efficient products with arbitrary vectors. A second motivation is any type of filtering or packet-processing algorithm applied to a real-time application such as video streaming, e.g., cleaning up a noisy video; here both input and output should be streamed, and the output should be in the correct order. The general application can be viewed as forwarding the entries of  $Ax$  quickly over a communication channel by processing the input in an online manner. Third, it models the computation of  $Ax$  in restricted computational models such as finite-state machines with tiny working memory, a model that has been used to understand what *humans* can compute [7, 8]. The motivation for considering the model we use is the transformation of *challenges* (strings of characters or digits) to passwords, and one of the generic methods considered in [8] is a linear map where the challenge  $x$  is mapped to  $A \cdot x$ . It is natural to require that the output values appear in the correct order, as they represent the characters of a user's password that he/she is generating in

\*Georgia Tech. Supported in part by NSF awards CCF-1217793 and EAGER-1415498. vempala@gatech.edu

†IBM Almaden. Supported in part by the XDATA program of the Defense Advanced Research Projects Agency (DARPA), administered through Air Force Research Laboratory contract FA8750-12-C-0323. dpwoodru@us.ibm.com

his/her head without paper or pencil. For which matrices  $A$  can this computation be (a) carried out without pencil, paper or other aids by a human and (b) still provide security in that an adversary would have to see a large number of challenge-response pairs before being able to correctly respond to a challenge? The first requirement is captured by our model, since humans have very small working memory, but with proportionate effort can memorize lengthy descriptions and procedures.

While the streaming model and sketching-based algorithms are now standard in modern machine learning with large and dynamic data sets, we are not aware of any existing techniques that would address this problem. Our main contribution may be viewed as a structural characterization of a matrix that determines its amenability to matrix-vector products in a stream.

Naïvely, computing  $A \cdot x$  can be done using space  $O(m)$  in one pass: maintain a running sum of all  $m$  output coordinates. This extends easily to space  $O(m/k)$  in  $k$  passes. Can we do better?

The first simple observation is that the space required to compute  $A \cdot x$  in one pass is at most of the order of  $r$ , the rank of  $A$ . To achieve this, we compute the inner product of the input vector  $x$  with  $r$  linearly independent rows of  $A$ . These rows are chosen in advance independent of the input vector  $x$ . Since every row of  $A$  is a fixed linear combination of these  $r$  rows, by applying the fixed combinations to the  $r$  computed inner products, we can get the inner product of each row of  $A$  with  $x$  in the desired order.

Is this the best possible bound? In principle, the answer is no. As a trivial example, suppose  $A$  is the  $n \times n$  identity matrix. Then the rank is  $n$ , but computing  $A \cdot x$  takes space  $O(1)$ , as we simply have to output the coordinates of  $x$  in the order in which we read them. On the other hand, as we will see shortly, *any* 1-pass algorithm for the anti-diagonal (or reversal) matrix needs  $\Omega(n)$  space. This takes us to the central mathematical question we aim to answer: can we characterize the space complexity of computing  $A \cdot x$  in terms of a natural (and hopefully computable) parameter of  $A$ ? Unlike the usual linear algebraic rank, this parameter must depend on the ordering of the columns of  $A$ , as seen in the example of the diagonal and anti-diagonal matrices. We would, in addition, like to match the space complexity with an efficient algorithm.

As a second example, consider the lower-triangular matrix with all entries equal to 1 on and below the diagonal. Then  $A \cdot x$  can be computed by adding the next coordinate of  $x$  to the previous output value, and outputting this sum. This again requires space  $O(1)$  even though the rank is  $n$ . Along these lines, consider the *upper*-triangular matrix with 1s on and above the

diagonal. For this matrix, it is not hard to see that  $O(1)$  space suffices, using two passes over  $x$ : in the first pass, we compute the sum  $s$  of all coordinates of  $x$  and output it. In the second pass, we subtract each coordinate of  $x$  and output until we reach the last coordinate. As we will see later in the paper, computing  $A \cdot x$  in only one pass by any randomized algorithm needs  $\Omega(n)$  space. This raises the more general question of the space complexity of computing  $A \cdot x$  in  $k$  passes.

**Our Results.** We begin with a nearly tight (up to a small constant) characterization of the 1-pass space complexity of computing  $A \cdot x$  for any matrix  $A$  whose entries are reals or from a finite field. Our bounds are nearly optimal for each single  $A$ , not just in the worst case. The required working space is determined, up to a constant, by a parameter we define as the *streaming rank* of  $A$ .

**DEFINITION 1.1.** (*Streaming Rank*) Define the **boundary**  $b(i)$  of a row  $i$  of  $A$  to be the index of the last nonzero column of the  $i$ 'th row, i.e.,  $A_{i,b(i)} \neq 0$ , but  $A_{i,j} = 0$  for all  $j > b(i)$ . For each  $(i, b(i))$ , consider the submatrix  $B_i$  of  $A$  formed by taking rows  $i + 1, \dots, n$  and columns  $1, \dots, b(i) - 1$ . Let  $r = \max_{i \in [n]} \text{rank}(B_i)$ . We refer to  $r$  as the **streaming rank** of  $A$ .

**THEOREM 1.1.** (*1-Pass Characterization*)

1. There is a deterministic 1-pass streaming algorithm for computing  $A \cdot x$  over the field  $\mathbb{F}$  using  $O(r)$  words of space, where  $r$  is the streaming rank of  $A$ .
2. Any randomized algorithm that computes  $A \cdot x$  over the field  $\mathbb{F}$  in the correct order must use  $\Omega(r)$  bits of space.

We assume a word is  $O(\log |\mathbb{F}|)$  bits for finite fields  $\mathbb{F}$ , and  $O(\log(nm))$  bits for the reals. The lower bound can be strengthened to  $\Omega(r \log |\mathbb{F}|)$  bits for finite fields and to  $\Omega(r \log(mn))$  bits for the reals.

In the examples of the identity matrix and the lower-triangular matrix, the streaming rank is 1, while for the upper triangular matrix, the streaming rank is  $n$ . Another illustrative example is the anti-diagonal matrix with ones in the entries  $(i, n + 1 - i)$  (also called the “reversal” matrix since  $A \cdot x$  is simply the entries of  $x$  in reverse order). For this matrix, the streaming rank is  $n$  and any 1-pass algorithm needs  $\Omega(n)$  space.

We then turn to multi-pass algorithms, for which we also obtain a characterization up to a constant factor. The relevant parameter here is the *k-pass streaming rank* defined in Section 3, which generalizes the streaming rank. While this parameter generalizes the streaming rank, there is an important new aspect:

the working memory from previous passes can be used for computations in the current pass. Indeed, if the algorithm were required to clear its memory at the end of each pass, then the  $k$ -pass version can be shown to equal, up to an additive  $O(k \log m)$  loss, to the minimum over all partitions of the rows of  $A$  into  $k$  contiguous subsets of rows, of the maximum 1-pass streaming rank of each subset. Our 1-pass characterization implies that this would give a tight bound on the space complexity.

To define the multi-pass streaming rank, we first extend the definition of (1-pass) streaming rank.

**DEFINITION 1.2.** (*Streaming Rank with Advice*) *The streaming rank with advice of a matrix  $A$  is the smallest integer  $r$  such that for any vector  $x$  whose coordinates are presented in the order  $x_1, \dots, x_n$ , the coordinates of  $Ax$  can be output in the correct order in a single pass over  $x$  using  $r$  pre-computed words (“advice”), that may depend on  $x$ , and  $r$  words of working memory during the computation.*

We note that with no advice, by the first part of Theorem 1.1, this definition of streaming rank is within an absolute constant of the streaming rank.

Being able to use memory from previous passes can be highly beneficial. As a simple example, consider the upper-triangular matrix of all 1s. If memory is cleared after each pass, then with  $s$  space, the algorithm needs  $n/s$  passes, since it can only compute  $s$  output coordinates in each pass. However, if memory from previous passes can be used, then the algorithm simply computes the sum of all coordinates of  $x$  in the first pass, stores it in memory and outputs it; in the second pass, it subtracts each coordinate of  $x$  from the current memory and outputs that. Thus it only needs  $O(1)$  space and 2 passes.

We are now ready to define the  $k$ -pass streaming rank.

**DEFINITION 1.3.** ( *$k$ -pass Streaming Rank*) *The  $k$ -pass streaming rank of an  $m \times n$  matrix  $A$  is the smallest integer  $r$  such that there exist  $k - 1$  integers  $i_0 = 0 < i_1 < \dots < i_{k-1} < m = i_k$ , inducing subsets of rows  $A^1, \dots, A^k$ , where the matrix  $A^j$  consists of the rows  $i_{j-1} + 1, \dots, i_j$ , such that  $A^1$  has streaming rank at most  $r$  and for  $j \geq 2$ , each  $A^j$  has streaming rank-with-advice at most  $r$ .*

Proving the nearly matching upper and lower bounds in terms of this parameter here turns out to be significantly more involved. To do this, we use an alternative and equivalent definition in Section 3. We summarize the bounds below; more details for real matrices and those over finite fields are given in the same section.

**THEOREM 1.2.** *Let  $r$  be the  $k$ -pass streaming rank of  $A$ . Then,*

1. *There is a deterministic  $k$ -pass streaming algorithm computing  $A \cdot x$  using  $O(r)$  words of space.*
2. *Any randomized  $k$ -pass streaming algorithm computing  $A \cdot x$  with probability at least  $3/4$  must use  $r/6 - O(k \log(mn))$  bits of space.*

*We assume a word is  $O(\log |\mathbb{F}|)$  bits for finite fields  $\mathbb{F}$ , and  $O(\log(nm))$  bits for the reals. The lower bound can be strengthened to  $r(\log |\mathbb{F}|)/6 - O(k \log(mn))$  bits for finite fields and to  $r(\log |\mathbb{F}|)/6 - O(k \log(mn))$  bits for the reals.*

We use Theorem 1.2 in Theorem 5.3 to give an example of a matrix  $A$ , for which with  $k$  passes one needs  $\Omega(n/k)$  space to compute  $A \cdot x$ , but with  $k + 1$  passes one can compute  $A \cdot x$  with only  $O(1)$  space. This is the largest separation possible.

These theorems establish the existence of nearly optimal streaming algorithms for computing  $A \cdot x$ . However, since the algorithm depends on  $A$ , it is important to understand the complexity of realizing the algorithm itself. In particular, how efficiently can the multi-pass streaming rank be computed? It will be clear from the analysis that the computation of this parameter for a given matrix realizes the steps of the optimal algorithm for the matrix. In Section 4, we give an efficient algorithm for computing the  $k$ -pass streaming rank. These results are captured in Theorem 4.1.

Our bounds extend naturally to the setting where the output can be buffered, or more precisely, each output bit can be up to  $r$  positions out of order; our characterization of the space complexity remains the same.

We mention two further applications. Using our lower bounds, it follows that well-known sketching matrices such as Gaussian, Subsampled Randomized Hadamard [2, 16, 18], and CountSketch [9] are not amenable to low-space computation of  $A \cdot x$  even if  $A$  is fixed and  $x$  is presented in order, and give no nontrivial tradeoff with the number of passes. Our lower bounds hold more generally for all distributional Johnson-Lindenstrauss transforms. We develop these fully in Section 5. These matrices can be used for *subspace embeddings*, a notion that has led to faster algorithms for least-squares regression and low-rank approximation, see, e.g., [20] for a survey. Here one chooses a random matrix  $A$  from an appropriate family of random matrices so that for a fixed (typically, tall and rectangular) matrix  $X$ , one has  $\|AXy\|_2 = (1 \pm \epsilon)\|Xy\|_2$  simultaneously for all vectors  $y$ . We point out that such

a notion is quite brittle to approximation, e.g., if  $X^i$  denotes the  $i$ -th column of  $X$  and we were to instead compute an approximation to  $AX^i$  for each  $i$ , say  $AX^i + w^i$  for a vector  $w^i$  with  $0 < \|w^i\|_2 \ll \|AX^i\|_2$  for each  $i$ , then the property that  $\|AXy\|_2 = (1 \pm \epsilon)\|Xy\|_2$  simultaneously for all vectors  $y$  would in general not hold, and consequently, for the downstream applications such as regression, one would not achieve relative error. Thus, in this case one cannot settle for an approximate computation to the matrix product  $A \cdot X$ .

Returning to one of our original motivations, we view  $x$  as a vector to be encoded and  $y = A \cdot x$  as a linear encoding of  $x$ . Then our bounds allow us to decide which matrices  $A$  have low-space algorithms that run in  $k$  passes over  $x$ , and how the computation would be realized. In particular tridiagonal matrices (and matrices whose nonzeros take up only a constant number of diagonals), which play a central role in many numerical algorithms [17], need only  $O(1)$  space to compute  $A \cdot x$ . Returning to the example of passwords and human computation [7, 8], our results show that for humanly computable linear password schemes that use only a few passes, the matrix  $A$  must have constant streaming rank. This is because in such models a human algorithm is allowed only a constant amount of working memory. Our theorem characterizes such matrices and also describes the constant-size working memory algorithm that goes with any given matrix. This is a much larger class than constant rank matrices, and includes intuitive ones such as the upper triangular matrix and the tri-diagonal matrix, which lead to interesting password schemas.

Finally, some applications may allow one to output the entries of  $Ax$  in an arbitrary order; we discuss this in Question 2 in Section 6.

**Related work.** There is a large literature on sketching for matrix and linear algebra problems. These are generally worst-case bounds, such as the work by [1]. Related to our applications is the work of [4] (see also [3]) on lower bounds for combinatorial objects such as hash functions and linear codes. We note that our model is different in that we give tight bounds for each single matrix  $A$ . Perhaps surprisingly the bound can be captured in terms of a single parameter of the matrix and can be much better than the rank of the matrix. Our techniques also appear to be quite different.

Our model shares some similarities with work in data streams with read/write tapes [5, 6, 10, 11, 12], though a key difference is that in our model we can only write the next symbol to the output tape and cannot read what we have already written. This is motivated in scenarios in which one wants to quickly transmit the part of the output that one has already

computed, and one is too resource-constrained to keep this information locally. As with such works, a difficulty in proving lower bounds is that we cannot use standard communication complexity arguments due to the output tape not counting towards the space complexity.

**Preliminaries.** Throughout all logarithms are base 2. We will consider the entries of  $A$  and  $x$  to be drawn from either the reals or a finite field, and we denote the field by  $\mathbb{F}$ . In the case of the reals, we consider a word to be  $O(\log(nm))$  bits and we assume the entries of  $A$  and  $x$  are multiples of  $\frac{1}{\text{poly}(nm)}$  and bounded in magnitude by  $\text{poly}(nm)$ , so that they fit in a constant number of words. We can scale all entries by  $\text{poly}(nm)$  so we will in fact assume the entries are integers bounded in magnitude by  $\text{poly}(nm)$ . In the case of finite fields, we will assume the field size is at most  $\text{poly}(nm)$  so again individual entries of the field fit into a machine word.

## 2 Characterization for 1-Pass Algorithms

We begin with the algorithm for computing  $A \cdot x$  in one pass. Recall the matrices  $B_1, \dots, B_m$  defined for each row  $i$  of  $A$  above. For each  $i$ , the algorithm precomputes a subset of row indices such that the corresponding rows form a basis for  $B_i$ . When  $x$  is presented, the algorithm maintains the inner products of the prefix of  $x$  seen so far with the chosen basis of  $B_i$ . We note that the set of rows used in the basis might change drastically from one step to the next. However, these row indices are precomputed, and the only runtime computation is inner products with prefixes of  $x$ . The coordinate  $(Ax)_i$  can be computed from these inner products, since we can write the inner product of any row of  $B_i$  with the first  $i$  coordinates of  $x$  as a linear combination of the inner products with the basis maintained. The full algorithm is described in Fig. 1.

*Proof.* (of Theorem 1.1.) We will prove the algorithm is correct and uses only  $O(r)$  words of memory by induction on  $i$ . Suppose that we have output the first  $i$  coordinates of  $Ax$ , and are currently at the  $b(i)$ -th position of the stream. Suppose inductively we have maintained the inner products of  $x$  with a basis for  $B_i$ . Since  $\text{rank}(B_i) = r$ , this can be done with  $O(r)$  words of space. Now consider trying to output  $(Ax)_{i+1}$ . If  $b(i+1) \leq b(i)$ , then since we have maintained a basis for  $B_i$ , we can immediately output  $(Ax)_{i+1}$ . Otherwise, suppose  $b(i+1) > b(i)$ , and consider the matrix  $B_{i+1}$ . There is a basis for the row space of  $B_{i+1}$  consisting of  $r$  rows of  $B_{i+1}$  and formed by taking  $r$  rows of  $B_i$  and concatenating  $b(i+1) - b(i)$  positions to them. Consider this  $r \times (b(i+1) - 1)$  submatrix  $C$  of  $B_{i+1}$ . Then, as one processes the stream, items  $b(i)+1, \dots, b(i+1)$ , one can maintain the inner products of  $x$  with  $C$  using  $r$  words

**A-times-x:**

Start with  $i = 0$  and an empty basis  $B$  (this will be a subset of rows of  $B_i$ ). Initialize  $b(0) = 1$ . Repeat:

1. If  $b(i+1) \leq b(i)$ , the basis stays the same for  $i+1$ . Else, extend  $B$  to a basis  $B'$  of  $B_{i+1}$  by extending  $r$  rows of  $B_i$  to rows of  $B_{i+1}$ .
2. Compute inner products of  $x$  with the new basis, by recomputing inner products with the first  $b(i)$  columns of each row of  $B'$ ; each such inner product is a fixed linear combination of the previously computed inner products. Then extend this to the full inner products for  $B'$  by advancing along the columns (coordinates of  $x$ ) till  $b(i+1)$ .
3. Compute  $(Ax)_{i+1}$  using a fixed linear combination of the inner products, and output it.

Figure 1: The 1-pass algorithm.

of space, since  $C$  has only  $r$  rows. But the rows of  $C$  form a basis for  $B_{i+1}$ , and so now one can (1) output the first  $i+1$  coordinates of  $Ax$ , (2) one is currently positioned at the  $b(i+1)$ -st position of the stream, and (3) one has maintained the inner products of  $x$  with a basis for  $B_{i+1}$ . This completes the inductive step. By induction, there is a deterministic algorithm succeeding with  $O(r)$  words of space.

Next we show the lower bound, i.e., the space complexity is  $\Omega(r)$  words, where for finite fields we consider a word to be  $\log_2 |\mathbb{F}|$  bits, and for the reals we consider a word to be  $\log_2(mn)$  bits. Let  $i$  be such that  $B_i$  has rank  $r$ . Consider a uniformly random input  $x$  subject to the constraint that  $x_j = 0$  for  $j \geq b(i)$ , that is, for coordinates  $j < b(i)$  the entries of  $x$  are i.i.d. uniformly random words. Then consider the memory contents  $S$  of the streaming algorithm  $Alg$  at the time the algorithm outputs  $(Ax)_i$ .

Since the algorithm is correct with constant probability, it must output  $Ax$  with constant probability. However, at this point in the stream its output is a function only of  $S$ , since it is promised that  $x_j = 0$  for  $j \geq b(i)$  and therefore can generate each of the entries of  $x_j$  for  $j \geq b(i)$  by itself. Since it outputs  $Ax$  with constant probability, this implies there is a (possibly randomized) function which takes as input  $S$  and outputs  $B_i y$  with constant probability, where  $y$  is the  $(b(i) - 1)$ -dimensional vector which agrees with  $x$  on the first  $b(i) - 1$  coordinates. Here we need the following version of Fano's inequality in information theory.

**FACT 2.1.** (*Fano's Inequality*) *Let  $Y$  and  $Z$  be random*

*variables from some joint distribution, and suppose there is a (possibly randomized) function  $f$  for which  $\Pr[f(Z) = Y] \geq 1 - \delta$ . Then,*

$$H(Y | Z) \leq H_b(\delta) + \delta \log(\text{supp}(Y) - 1),$$

*where  $\text{supp}(Y)$  denotes the size of the support of the random variable  $Y$ , where  $H_b(q) = q \log 1/q + (1 - q) \log 1/(1 - q)$  is the binary entropy function, and where the conditional entropy*

$$\begin{aligned} H(Y | Z) &= \sum_{Z=z} \Pr[Z = z] H(Y | Z = z) \\ &= \sum_{Z=z} \Pr[Z = z] \times \\ &\quad \times \sum_{Y=y} \Pr[Y = y | Z = z] \log_2(1/\Pr[Y = y | Z = z]). \end{aligned}$$

We thus have for a row basis  $B$  of  $r$  rows of  $B_i$ ,

$$\begin{aligned} H(y | S) &= H(y, By | S) \\ &= H(By | S) + H(y | By, S) \\ &\leq H(By | S) + H(y | By) \\ &\leq H_2(C) + Crw + H(y | By), \end{aligned}$$

where  $C < 1$  is a constant (probability of failure), and where  $w = \log_2 |\mathbb{F}|$  in the case of a finite field, and  $w = \log_2(mn)$  in the case of the reals (with our assumption that the entries are  $O(\log(mn))$  bit integers). The first equality follows since  $By$  is determined from  $y$ , the second equality follows from the chain rule for entropy, the first inequality follows since conditioning cannot increase the entropy, and the second inequality follows from Fact 2.1.

It follows that the mutual information

$$\begin{aligned} I(S; y) &= H(y) - H(y | S) \\ &= H(By) + H(y | By) - H(y | S) \\ &\geq H(By) - H_2(C) - Crw \\ &\geq \Omega(rw), \end{aligned}$$

where the first equality is the definition of mutual information, the second equality is the chain rule for entropy, the first inequality follows by plugging in our bound above on  $H(y | S)$ , and the second inequality follows from the claim, that we show now, that  $H(By) = rw$ .

To see this claim, fix a square submatrix  $C$  of  $B$  of full rank, and write  $y = y^C + y^{-C}$  where  $y^C$  agrees with  $y$  on coordinates indexed by  $C$  and is 0 on the remaining coordinates. By definition,  $y^C$  and  $y^{-C}$  are independent. Then we have  $By = By^C + By^{-C}$ , and

consider any fixing of  $By^{-C}$ . Then there is a bijection between the values of  $y^C$  and  $By^C$ , and of  $By^C$  and  $By$ . Hence,  $H(By) = rw$ .

But  $I(S; y) \leq H(S) \leq |S|$ , where  $|S|$  denotes the length of  $S$ , and so  $|S| = \Omega(rw)$  bits.

### 3 Characterization for Multi-Pass Algorithms

To give a multi-pass characterization, we define the notion of adaptively fitting a matrix. We note that the definition here is more technical, and used here for the purpose of the proof, though it is equivalent to the one given in Section 1.

**DEFINITION 3.1.** (*Adaptively Fitting a Matrix*) Given an  $m \times n$  matrix  $A$  over a field  $\mathbb{F}$ , suppose we specify a column  $b(i)$  for each row  $i \in [m]$  so that  $b(i) \leq b(i+1)$  for all  $i$ . Let  $T$  be an  $r$ -dimensional subspace of  $\mathbb{F}^n$ . We say that  $T$   **$r$ -adaptively fits**  $A$  at  $(b(1), \dots, b(m))$  if we can generate a sequence of  $m$  subspaces  $T_0 = T, T_1, \dots, T_m$  such that for each  $i = 1, \dots, m$ ,

1.  $T_i$  is a subspace of  $\mathbb{F}^{n-b(i)}$  of dimension at most  $r$ , which contains the  $(n - b(i))$ -length suffix of the  $i$ -th row of  $A$  (i.e., the vector consisting of the last  $n - b(i)$  coordinates of the  $i$ -th row of  $A$ ), and
2. a basis for  $T_i$  can be generated from a basis for  $T_{i-1}$  by restricting  $r$  vectors in  $T_{i-1}$  to their last  $n - b(i)$  coordinates.

**DEFINITION 3.2.** (*Multi-Pass Streaming Rank*) We define the  **$k$ -pass streaming rank** to be the smallest integer  $r$  such that there exist  $k - 1$  integers  $i_0 = 0 < i_1 < i_2 < \dots < i_{k-1} < m = i_k$ , which we call **breakpoints**, inducing subsets of rows  $A^1, \dots, A^k$  of  $A$  where  $A^j$  consists of the rows  $i_{j-1} + 1, i_{j-1} + 2, \dots, i_j$ , such that for each  $A^j$ , each row  $i$  of  $A^j$  has a boundary point  $b(i)$  satisfying:

1.  $b(i) \leq b(i+1)$ . Further, if  $j = 1$ , then  $b(i)$  is the rightmost non-zero entry of row  $i$ .
2. Letting  $L_i$  be the contiguous submatrix of  $A^j$  with lower left entry equal to  $A_{i_j, 1}$  and upper right entry equal to  $A_{i, b(i)}$ , we have  $\text{rank}(L_i) \leq r$ .

Further, it is required that there exist subspaces  $U^2, U^3, \dots, U^k$  such that  $U^j$   $r$ -adaptively fits  $A^j$  at the boundary points in  $A^j$ . We say that a set of breakpoints and boundary points having the above properties is a **realization** of the  $k$ -pass streaming rank.

We now prove the the upper and lower bounds of the main theorem separately. For the upper bound, we use the following algorithm. The pre-computations needed to establish the  $k$ -pass streaming rank of the

matrix  $A$  and its parameters can be carried out in polynomial time, as we show in Theorem 4.1. Here we focus on the streaming algorithm, assuming that we have pre-computed the breakpoints and partition of  $A$  into  $k$  subsets of rows, one for each pass, (row) bases for the matrices  $U^2, \dots, U^k$  and for the submatrices  $L^1, L^2, \dots$  that are induced by each  $A^j$ .

We note that the information going from one pass to the next is  $O(r)$  words of memory (inner products with a basis for  $U^j$ ) and is the “advice”. Thus, proving the upper bound with this definition implies an  $O(r)$  upper bound on the  $k$ -pass streaming rank as defined earlier.

**THEOREM 3.1.** (*Multipass Upper Bound*) Let  $r$  be the  $k$ -pass streaming rank of  $A$  over the field  $\mathbb{F}$ . There is a deterministic  $k$ -pass streaming algorithm computing  $A \cdot x$  using  $O(r)$  words of space.

*Proof.* Let  $r$  be the  $k$ -pass streaming rank of  $A$  and let  $i_0, \dots, i_k$  be the associated breakpoints. In the  $j$ -th pass we will output  $A^j x$ . Unlike the proof for 1-pass algorithms, we will now maintain two bases in each pass  $j > 1$  corresponding to a basis for  $L_i$  at each point  $(i, b(i))$  of  $A^j$  and a basis corresponding to adaptively fitting  $A^j$  by  $U^j$ . We will also maintain information for the  $(j+1)$ -st pass. We now give the details.

In the first pass, we run the algorithm of Theorem 1.1. In parallel, we maintain the inner products of  $x$  with a basis for the subspace  $U^2$  in the definition of the  $k$ -pass streaming rank. Notice that this can be done using  $O(r)$  words of space.

Suppose inductively that we are in the  $j$ -th pass,  $j \geq 2$ , we have just output the value  $(A \cdot x)_i$ , we are currently at the  $b(i)$ -th position of the stream, and we have maintained the inner products of  $x$  with a basis for both  $L_i$  and with a basis for  $U_i^j$ , where  $U_i^j$   $r$ -adaptively fits  $A^j$  at the boundary points in  $A^j$ , and  $U_i^j$  is the  $i$ -th subspace in the sequence of subspaces given by Definition 3.1. Since the  $k$ -pass streaming rank is  $r$ , these inner products can be stored using  $O(r)$  words of space.

Now consider trying to output  $(Ax)_{i+1}$ . We can write this inner product as  $\langle \ell, x \rangle + \langle d, x \rangle + \langle u, x \rangle$ , where  $\ell$  is the vector which agrees with the  $i$ -th row of  $A$  in positions  $1, 2, \dots, b(i)$  and is 0 otherwise,  $d$  is the vector which agrees with the  $i$ -th row of  $A$  in positions  $b(i), b(i)+1, \dots, b(i+1)$  and is 0 otherwise, and  $u$  is the vector which agrees with the  $i$ -th row of  $A$  in positions  $b(i+1)+1, b(i+1)+2, \dots, n$ .

Since we have maintained the inner products of a basis for  $L_i$  with  $x$ , and  $\ell$  is in the row span of  $L_i$ , we can compute  $\langle \ell, x \rangle$  from the information we have

### **$k$ -pass Matrix-vector Product**

In the first pass run the 1-pass algorithm and output  $A^1 \cdot x$ . During this pass also compute the inner products of  $x$  with a basis of  $U^2$ .

In the  $j$ 'th pass,  $2 \leq j \leq k$ :

1. Output  $(A \cdot x)_i$  when the  $b(i)$ 'th entry of  $x$  is reached. To do this, maintain inner products of  $x$  with a basis for  $L_i$ , inner products with a basis for  $U_i^j$  (the last  $n - b(i)$  coordinates of each row of  $U^j$ ).
2. To compute  $(A \cdot x)_{i+1}$ , let  $\ell, d, u$  be  $n$ -vectors induced by the  $i$ 'th row of  $A$  as follows:  $\ell$  is equal to the row in the first  $b(i)$  coordinates and zero in the rest,  $d$  agrees with it in positions  $b(i) + 1, \dots, b(i + 1)$  (and zero in the rest), and  $u$  is zero till  $b(i + 1)$  and equal to the  $i$ 'th row from  $b(i + 1) + 1$  till the end.
3. Compute  $\langle \ell, x \rangle$  viewing  $\ell$  as a linear combination of the basis of  $L_i$ , and its inner products with  $x$ .
4. Since  $u$  can be viewed as a linear combination of rows of  $U_i^j$  restricted to the last  $n - b(i + 1)$  coordinates, compute  $\langle u, x \rangle$  by subtracting off the inner product of  $x$  with the coordinates of the row basis of  $U^j$  in positions  $b(i) + 1, \dots, b(i + 1)$ , then taking the appropriate linear combination to obtain  $\langle u, x \rangle$ .
5. Compute  $(A \cdot x)_{i+1} = \langle \ell, x \rangle + \langle d, x \rangle + \langle u, x \rangle$ . The term  $\langle d, x \rangle$  is computed during the pass from positions  $b(i) + 1$  to  $b(i + 1)$  (it is zero if  $b(i) = b(i + 1)$ ).
6. Update the inner products with a basis for  $L_{i+1}$  as in the 1-pass algorithm. We have already updated the inner products for  $U_{i+1}^j$  when computing  $\langle u, x \rangle$ .
7. For  $j < k$ , during the  $j$ 'th pass, using  $r$  additional words of memory, compute the inner products of  $x$  with a basis for  $U^{j+1}$ .

Figure 2: *Alg*: The  $k$ -pass algorithm.

maintained inductively. As we process the stream from the  $b(i)$  to the  $b(i + 1)$ -th position, we can maintain  $\langle d, x \rangle$ . Finally, since  $U^j$  adaptively fits  $A^j$ , we already have a basis  $T$  of  $r$  vectors in  $U_i^j$  so that if we restrict to their final  $n - b(i + 1)$  coordinates, we can generate  $u$  as a linear combination of the vectors in this basis.

As we process the stream from the  $b(i)$  to the  $b(i + 1)$ -st position, we can subtract off the inner product of  $x$  with the  $b(i) + 1, b(i) + 2, \dots, b(i + 1)$ -st coordinates of each of the vectors in this basis. We can then compute  $\langle u, x \rangle$  by taking a linear combination of the remaining inner products with vectors in this basis. Hence, we can output  $\langle u, x \rangle$ .

It remains to show how to maintain the inductive hypothesis. Since we have maintained the inner products of a basis for  $L_i$  with  $x$ , and the streaming rank is at most  $r$ , we can form a basis for  $L_{i+1}$  by taking  $r$  rows of  $L_i$  and concatenating  $b(i + 1) - b(i)$  positions to them. As in the 1-pass characterization, consider this  $r \times (b(i + 1) - 1)$  submatrix  $C$  of  $B_{i+1}$ . Then, as one processes the stream items  $b(i) + 1, \dots, b(i + 1)$ , we can maintain  $C$  using  $O(r)$  words of space, since  $C$  has only  $r$  rows. Since the rows of  $C$  form a basis for  $B_{i+1}$ , we have the inner products of  $x$  with a basis for  $L_{i+1}$ .

To maintain the inner products of  $x$  with a basis for  $U_{i+1}^j$ , as described above we can subtract off the inner product of  $x$  with the  $b(i) + 1, b(i) + 2, \dots, b(i + 1)$ -st coordinates of each vector in the set  $T$  defined above, which is then a basis for  $U_{i+1}^j$ . Hence, we have maintained the inner products of  $x$  with a basis for  $U_{i+1}^j$  in  $O(r)$  words of space. During this pass, we also compute the inner products of  $x$  with a basis for  $U^{j+1}$  using  $r$  additional words of memory.

This completes the inductive step and the proof.

**THEOREM 3.2.** (*Multipass Lower Bound*) *Any randomized  $k$ -pass streaming algorithm computing  $A \cdot x$  over the field  $\mathbb{F}$  and succeeding with probability at least  $3/4$  must use  $r/6 - O(k \log(mn))$  bits of space.*

*The lower bound can be strengthened to  $(r/6) \log p - O(k \log(mn))$  bits in the case the field has  $p$  elements (for some prime power  $p$ ), and to  $(r/6) \log(mn) - O(k \log(mn))$  in the case the field is the reals with  $O(\log(nm))$  bits of precision.*

The proof intuition is as follows. We first argue that it suffices to prove lower bounds over finite fields  $GF(p)$ , since from these we will also be able to obtain a lower bound over the reals with  $O(\log(nm))$  bits of precision. We put a uniform distribution on  $x \in GF(p)^n$ . We then apply Yao's minimax principle, which implies we can assume the algorithm for computing  $A \cdot x$  is deterministic and succeeds with probability at least  $3/4$  over our distribution on  $x$ . We now lower bound the space complexity of this algorithm in terms of the  $k$ -pass streaming rank of  $A$ .

We argue, by averaging, that there is a set  $I$  of breakpoints — that is, the identities of the final positions of  $Ax$  output by the algorithm *Alg* after each pass — that are the same for a large subset of possible

$x$ . Note that breakpoint here is well-defined and does not depend on any particular assumption on  $Alg$ , which is an arbitrary deterministic algorithm. That is, the  $j$ -th breakpoint is simply the last entry of  $Ax$  which is output after the  $j$ -th pass (recall all such entries need to be output in order).

Although we can fix the breakpoints by an averaging argument, creating submatrices  $A^1, \dots, A^k$  for which  $Alg$  outputs  $A^j x$  in the  $j$ -th pass, the notion of boundary points from a lower bound perspective is more involved. We define a boundary point of a row  $i$  in  $A^j$  to be the point  $b(i)$  for which  $\langle A_i^j, x \rangle$  is output, that is, after having read  $x_1, x_2, \dots, x_{b(i)}$  in the  $j$ -th pass, then  $\langle A_i^j, x \rangle$  is output. Unfortunately now the  $b(i)$  are random variables, which depend on  $x$ , and unlike the breakpoints, there are too many possibilities of boundary points inside of each  $A^j$  to fix the boundary points in advance.

However, we know that since the  $k$ -pass streaming rank is at least  $r$ , there exists a  $j$  for which for any  $x$ , if we look at the boundary points  $b(i_{j-1} + 1), \dots, b(i_j)$  for  $A^j$  of  $Alg$  on input  $x$ , there is a row  $i$  of  $A^j$  for which either (1)  $L_i$  has rank at least  $r$ , or (2) there is no  $r$ -dimensional subspace of  $\mathbb{F}^n$  which fits  $A^j$  at  $(b(i_{j-1} + 1), \dots, b(i_j))$ . Either half the time, over  $x$  which realize this set of breakpoints, we are in case (1) or in case (2). We handle these cases separately, showing although we cannot apply an averaging argument to fix all the boundary points themselves, in the first case we can apply an averaging argument to fix the first boundary point with a certain property, while in the second case it will suffice to argue about an extreme case, when all boundary points are as far to the right as possible.

*Proof.* We first argue that it suffices to consider only fields  $\mathbb{F}$ , which we denote by  $GF(p)$  for a prime power  $p$ . Indeed, if we establish the claimed lower bound for such fields, given an algorithm which computes  $A \cdot x$  over the reals, we can compute  $A \cdot x \pmod p$  for a prime  $p = \text{poly}(mn)$ . In the remainder of the proof we fix a finite field  $GF(p)$ .

We place the uniform distribution  $\mu$  on  $x \in GF(p)^n$ . By Yao's minimax principle, it suffices to prove a lower bound for a deterministic algorithm  $Alg$  computing  $A \cdot x$  with probability at least  $3/4$  over  $x \sim \mu$ . The number of possibilities of breakpoints - that is, the identities of the final positions of  $Ax$  output by  $Alg$  after each pass - of the algorithm  $Alg$ , is  $m^{k-1}$ . Note that the breakpoints may depend on  $x$ .

CLAIM 3.1. (*Typical Input Set*) *There exists a set  $I \subseteq \{0, 1, 2, \dots, p-1\}^n$  with  $|I| \geq p^n / (12m^{k-1})$  for which*

1. For all  $x \in I$ , the breakpoints  $i_0, i_1, \dots, i_k$ , i.e., the identities of the final positions of  $Ax$  output by  $Alg$  after each pass, are the same,
2. Letting  $\mu_I$  be the uniform distribution on  $x \in I$ , the probability that  $Alg$  outputs  $A \cdot X$ , over  $X \sim \mu_I$ , is at least  $2/3$ .

*Proof.* There are at most  $m^{k-1}$  possibilities of breakpoints. For a set of breakpoints for which at most  $p^n / (12m^{k-1})$  inputs  $x \in GF(p)^n$  realize that set of breakpoints, we call that set of breakpoints *small*. Then the total number of inputs realizing a small set of breakpoints is at most  $p^n / (12m^{k-1}) \cdot m^{k-1} \leq p^n / 12$ . It follows that the success probability with respect to the uniform distribution on the remaining  $11 \cdot p^n / 12$  inputs  $x \in GF(p)^n$  is at least  $3/4 - 1/12 = 2/3$ . In particular, there exists a set  $I$  of breakpoints for which at least  $p^n / (12m^{k-1})$  inputs  $x \in GF(p)^n$  realize that set of breakpoints, and for which the success probability of the algorithm on  $x \in I$  is at least  $2/3$ .

In the remainder of the proof, we fix a set  $I$  guaranteed by Claim 3.1 with corresponding breakpoints  $i_0, i_1, \dots, i_k$ .

We define a boundary point of a row  $i$  in  $A^j$  to be the point  $b(i)$  for which  $\langle A_i^j, x \rangle$  is output, that is, after having read  $x_1, x_2, \dots, x_{b(i)}$  in the  $j$ -th pass, then  $\langle A_i^j, x \rangle$  is output. The  $b(i)$  are random variables, which depend on  $x$ . Since the  $k$ -pass streaming rank is at least  $r$ , there exists a  $j$  for which for any  $x$ , if we look at the boundary points  $b(i_{j-1} + 1), \dots, b(i_j)$  for  $A^j$  of  $Alg$  on input  $x$ , there is an  $i$  for which either (1)  $L_i$  has rank at least  $r$ , or (2) there is no  $r$ -dimensional subspace of  $\mathbb{F}^n$  which fits  $A^j$  at  $(b(i_{j-1} + 1), \dots, b(i_j))$ . Note that we can choose the same  $j$  for all  $x$ , since otherwise by choosing this set of breakpoints there would be a way of choosing boundary points for each  $j$  so that the  $k$ -pass streaming rank were less than  $r$  (indeed, if for each  $A^j$  even only for a single  $x$  there were a choice for which (1) and (2) did not happen, then for each  $A^j$  we could use the boundary points chosen by  $Alg$  for that  $x$  and contradict the  $k$ -pass streaming rank being at least  $r$ ). Note also that there may be an input  $x$  which occurs in both cases (1) and (2); we *assign* the input  $x$  to case (1) if that happens; this assignment necessarily partitions all possible  $x$  into the two cases.

Let  $\mathcal{E}$  be the event that an  $x \sim \mu_I$  is assigned to case (1) in the previous paragraph. We either have  $\Pr[\mathcal{E}] \geq 1/2$  or  $\Pr[\neg\mathcal{E}] \geq 1/2$ . If we are assigned to the first case then also  $\Pr_{\mu_I}[Alg \text{ succeeds} \mid \mathcal{E}] \geq 1/3$ , as otherwise  $\Pr_{\mu_I}[Alg \text{ succeeds}] < 1/2 \cdot 1 + 1/2 \cdot 1/3 = 2/3$ , a contradiction. Similarly, if we are assigned to the second case then also  $\Pr_{\mu_I}[Alg \text{ succeeds} \mid \neg\mathcal{E}] \geq 1/3$ .



We now handle each of these cases in turn.

**Case 1.** Let  $\mu_{I,1}$  be the uniform distribution on  $I$ , conditioned on being assigned to Case 1. Consider the *first* row  $i$  in  $A^j$  for which the choice  $b(i)$  of  $Alg$  on input  $x$  causes  $L_i$  to have rank at least  $r$ . There are at most  $mn$  choices of the pair  $(i, b(i))$ . Let  $\mathcal{F}_{i,b(i)}$  be the event that  $(i, b(i))$  is the pair corresponding to the first  $i$  for the  $x$  chosen, that is, row  $i$  in  $A^j$  causes the choice  $b(i)$  and  $L_i$  has rank at least  $r$ . We say  $(i, b(i))$  is *heavy* if  $\Pr[\mathcal{F}_{i,b(i)}] \geq \frac{1}{6mn}$ .

We have, all probabilities being taking over  $x \sim \mu_{I,1}$ , that

$$\begin{aligned} \frac{1}{3} &\leq \Pr[Alg \text{ succeeds} \mid \mathcal{E}] \\ &= \sum_{i,b(i)} \Pr[Alg \text{ succeeds} \mid \mathcal{E}, \mathcal{F}_{i,b(i)}] \\ &\quad \cdot \Pr[\mathcal{F}_{i,b(i)} \mid \mathcal{E}] \\ &\leq \sum_{\text{heavy } i,b(i)} \Pr[Alg \text{ succeeds} \mid \mathcal{E}, \mathcal{F}_{i,b(i)}] \\ &\quad \cdot \Pr[\mathcal{F}_{i,b(i)} \mid \mathcal{E}] + mn \cdot \frac{1}{6mn} \\ &\leq \sum_{\text{heavy } i,b(i)} \Pr[Alg \text{ succeeds} \mid \mathcal{E}, \mathcal{F}_{i,b(i)}] \\ &\quad \cdot \Pr[\mathcal{F}_{i,b(i)} \mid \mathcal{E}] + \frac{1}{6}, \end{aligned}$$

and consequently,  $\sum_{\text{heavy } i,b(i)} \Pr[Alg \text{ succeeds} \mid \mathcal{E}, \mathcal{F}_{i,b(i)}] \Pr[\mathcal{F}_{i,b(i)} \mid \mathcal{E}] \geq \frac{1}{6}$ . It follows that there exists an  $(i, b(i))$  for which  $\Pr_{\mu_{I,1}}[\mathcal{F}_{i,b(i)} \mid \mathcal{E}] \geq \frac{1}{6mn}$  and

$$(3.1) \quad \Pr_{\mu_{I,1}}[Alg \text{ succeeds} \mid \mathcal{E}, \mathcal{F}_{i,b(i)}] \geq \frac{1}{6}.$$

We fix this  $(i, b(i))$  in what follows.

Consider the matrix  $L_i$ , and let  $B$  be a square  $r \times r$  full rank submatrix. Let  $\mu_B$  denote the distribution on coordinates of  $X$  indexed by the columns of  $B$  when  $X$  is drawn from  $\mu_{I,1}$  and conditioned on  $\mathcal{F}_{i,b(i)}$  holding. Consider the quantity  $H(B \cdot Y)$ , where  $Y \sim \mu_B$  and  $H(\cdot)$  denotes the Shannon entropy of a random variable. Since  $B$  is invertible, given  $B \cdot Y$  one can reconstruct  $Y$  so  $H(B \cdot Y) = H(Y)$ . We can think of choosing  $Y$  by first sampling  $X \sim \mu_{I,1}$  conditioned on  $\mathcal{F}_{i,b(i)}$  holding, and then restricting to the columns indexed by  $B$ . Let  $X \setminus Y$  denote  $X$  restricted to coordinates not in  $B$ . Note that  $X$  is uniformly random over a set of size at least

$$(3.2) \quad \frac{p^n}{12m^{k-1}} \cdot \frac{1}{2} \cdot \frac{1}{6mn} = \frac{p^n}{144m^kn},$$

and so we have,

$$\begin{aligned} &n \log p - k \log(m) - \log(144n) \\ &\leq H(X) \\ &\leq H(X \setminus Y) + H(Y \mid X \setminus Y) \\ &\leq n \log p - r \log p + H(Y \mid X \setminus Y) \end{aligned}$$

where the first inequality uses (3.8), the second inequality uses subadditivity of the entropy, and the final inequality uses that  $H(X \setminus Y)$  is maximized when each of the  $n - r$  coordinates are independent elements of  $GF(p)$ . Hence,

$$(3.3) \quad H(Y \mid X \setminus Y) \geq r \log p - k \log(m) - \log(144n)$$

Let  $S$  be the state of  $Alg$  in the  $j$ -th pass at the time the algorithm reads  $X_{b(i)}$ . Here we use Fano's Inequality, stated earlier in Fact 2.1.

Since  $Alg$  outputs  $A^j \cdot X$  in the  $j$ -th pass with probability at least  $1/6$  by (3.1), it follows that from  $S, X_{b(i)+1}, X_{b(i)+2}, \dots, X_n$ , one can compute  $A^j \cdot X$  and therefore also  $L_i \cdot X_{\leq b(i)}$ , where  $X_{\leq b(i)}$  is the  $b(i)$ -dimensional vector which agrees with  $X$  on its first  $b(i)$  coordinates. Note that it is possible to compute  $L_i \cdot X_{\leq b(i)}$  since one is given  $X_{b(i)+1}, X_{b(i)+2}, \dots, X_n$  which can be used to subtract off an appropriate value for each entry in  $A^j \cdot X$ . If one is additionally given  $X \setminus Y$ , one can then compute  $Y$  by restricting to rows of  $L_i \cdot X_{\leq b(i)}$  indexed by rows of  $B$ , and subtracting off the appropriate values for each such row based on knowledge of  $X \setminus Y$ . Applying Fact 2.1,

$$(3.4) \quad H(Y \mid S, X \setminus Y) \leq H_b\left(\frac{1}{6}\right) + \frac{5r \log p}{6}$$

$$(3.5) \quad \leq \frac{5r \log p}{6} + O(1).$$

Therefore,

$$\begin{aligned} |S| &\geq H(S) \geq H(S \mid X \setminus Y) \geq I(S; Y \mid X \setminus Y) \\ &= H(Y \mid X \setminus Y) - H(Y \mid S, X \setminus Y) \\ &\geq r \log p - k \log m - \log(144n) - \frac{5r \log p}{6} - O(1) \\ &= \frac{r \log p}{6} - O(k \log(mn)), \end{aligned}$$

where the first inequality follows since the expected length of  $S$  is at least its entropy, the second inequality follows since conditioning does not increase entropy, the third inequality follows by definition of the mutual information, the first equality follows by writing the mutual information in a different way, and the fourth inequality follows by combining (3.3) and (3.4).

**Case 2.** Let  $\mu_{I,2}$  be the uniform distribution on  $I$ , conditioned on being assigned to Case 2. We will say an  $\ell$ -suffix of an  $n$ -dimensional vector is the  $\ell$ -dimensional vector obtained by taking the last  $\ell$  coordinates of the vector. For each  $x$  in the support of  $\mu_{I,2}$ , it causes  $\text{Alg}$  to choose boundary points  $(b(i_{j-1}) + 1, \dots, b(i_j))$  for which there is no  $r$ -dimensional subspace of  $\mathbb{F}^n$  which fits  $A^j$  at  $(b(i_{j-1}) + 1, \dots, b(i_j))$ . We have the following claim.

**CLAIM 3.2.** (*Effects of No  $r$ -Adaptive Fit*) *Suppose there is no subspace  $T$  which  $r$ -adaptively fits  $A^j$  at  $(b(i_{j-1}) + 1, \dots, b(i_j))$ . Then there exist rows of  $A^j$ , indexed by  $z_1, \dots, z_{r+1}$ , with  $z_1 < z_2 < \dots < z_{r+1}$  for which for all  $\ell \in \{2, 3, 4, \dots, r+1\}$ , the  $(n - b(z_\ell))$ -suffix of  $A^j_{z_\ell}$  is not in the span of the  $n - b(z_\ell)$  suffixes of the vectors  $A^j_{z_1}, \dots, A^j_{z_{\ell-1}}$ .*

*Proof.* Suppose not. Then we will find a subspace  $T$  which  $r$ -adaptively fits  $A^j$  at  $(b(i_{j-1}) + 1, \dots, b(i_j))$ . We include the first row of  $A^j$  in  $T$ . If the  $(n - b(i_{j-1} + 2))$ -suffix of the second row of  $A^j$  is not in the span of the  $(n - b(i_{j-1} + 1))$ -suffix of the first row of  $A^j$ , then we also include the second row of  $A^j$  in  $T$ . Otherwise, we do not change  $T$ . In general, when processing the  $i$ -th row of  $A^j$ , if the  $(n - b(i))$ -suffix of the  $i$ -th row of  $A^j$  is in the span of the  $(n - b(i))$ -suffixes of the vectors in  $T$ , we do not change  $T$ , otherwise we add the  $i$ -th row of  $A^j$  to  $T$ .

Whenever we add a row  $i$  of  $A^j$  to  $T$ , it must be the case that the  $(n - b(i))$ -suffix of the row is not in the span of the  $(n - b(i))$ -suffixes of the rows already in  $T$ . Hence, we can add at most  $r$  rows to  $T$ , as otherwise the rows added to  $T$  would give a set of rows satisfying the claim. However, if we add at most  $r$  rows to  $T$ , this implies the dimension of  $T$  after processing all rows in  $A^j$  is at most  $r$ . Moreover, by construction  $T$   $r$ -adaptively fits  $A^j$  at  $(b(i_{j-1}) + 1, \dots, b(i_j))$ . This is a contradiction to the premise of the claim, and so there exist rows indexed by  $z_1, \dots, z_{r+1}$ , as desired.

For a given  $x$  in the support of  $\mu_{I,2}$ , we can apply Claim 3.2 to obtain row indices  $z_1 < z_2 < \dots < z_{r+1}$ . The number of possible such choices of indices is at most  $\binom{m}{r+1}$ , which will be too large to average over, since a given fixing of these may reduce the entropy of  $x$  by an additive  $O(r \log(m/r))$  factor.

Instead, we argue that there is a single choice  $z_1 < z_2 < \dots < z_{r+1}$  which works for every  $x$  in the support of  $\mu_{I,2}$ . To see this, suppose we choose  $b^*(i_{j-1} + 1)$  to be the furthest point to the right in row  $i_{j-1} + 1$  for which  $\text{rank}(L_{i_{j-1}+1}) \leq r$ , where  $L_{i_{j-1}+1}$  is the contiguous submatrix of  $A^j$  with lower left corner in the  $(i_j, 1)$ -th position, and upper right

corner in the  $(i_{j-1} + 1, b^*(i_{j-1} + 1))$ -th position. Having chosen  $b^*(i_{j-1} + 1)$ , we then choose  $b^*(i_{j-1} + 2)$  to be the furthest point to the right in row  $i_{j-1} + 2$  for which  $\text{rank}(L_{i_{j-1}+2}) \leq r$ . Note that by construction,  $b^*(i_{j-1} + 2) \geq b^*(i_{j-1} + 1)$ . Repeating this argument, we find a sequence  $b^*(i_{j-1} + 1) \leq b^*(i_{j-1} + 2) \leq \dots \leq b^*(i_j)$ . The important observation is that for any possible set of boundary points  $b(i_{j-1} + 1) \leq b(i_{j-1} + 2) \leq \dots \leq b(i_j)$  chosen by  $\text{Alg}$  on some input  $x$ , we necessarily have  $b(i_{j-1} + w) \leq b^*(i_{j-1} + w)$  for all  $w = 1, 2, \dots, i_j - i_{j-1}$ . Indeed, otherwise we would have  $\text{rank}(L_{i_{j-1}+w}) > r$  for some such  $w$ , contradicting that we assigned  $x$  to case 2 (since recall, if  $x$  is in both cases 1 and 2, then it is assigned to case 1).

Since the  $k$ -pass streaming rank is at least  $r$ , and by our choice of  $A^j$ , we necessarily have that there is no subspace  $T$  which  $r$ -adaptively fits  $A^j$  at  $(b^*(i_{j-1} + 1), \dots, b^*(i_j))$ . By Claim 3.2, we can obtain row indices  $z_1 < z_2 < \dots < z_{r+1}$  for which for all  $\ell \in \{2, 3, 4, \dots, r+1\}$ , the  $(n - b^*(z_\ell))$ -suffix of  $A^j_{z_\ell}$  is not in the span of the  $n - b^*(z_\ell)$  suffixes of the vectors  $A^j_{z_1}, \dots, A^j_{z_{\ell-1}}$ . But note that if  $A^j_{z_\ell}$  is not in the span of the  $n - b^*(z_\ell)$  suffixes of the vectors  $A^j_{z_1}, \dots, A^j_{z_{\ell-1}}$ , then  $A^j_{z_\ell}$  is also not in the span of the  $n - b(z_\ell)$  suffixes of the vectors  $A^j_{z_1}, \dots, A^j_{z_{\ell-1}}$  if  $b(z_\ell) \leq b^*(z_\ell)$ . Consequently, by the observation in the previous paragraph, it follows for any  $x$  which is assigned to case 2, for whichever boundary points  $b(i_{j-1} + 1) \leq b(i_{j-1} + 2) \leq \dots \leq b(i_j)$  chosen by  $\text{Alg}$  on input  $x$ , we necessarily have that for all  $\ell \in \{2, 3, 4, \dots, r+1\}$ , the  $(n - b(z_\ell))$ -suffix of  $A^j_{z_\ell}$  is not in the span of the  $n - b(z_\ell)$  suffixes of the vectors  $A^j_{z_1}, \dots, A^j_{z_{\ell-1}}$ .

We will need to construct a **special basis**  $\mathcal{B}$  of  $GF(p)^n$  to prove our lower bound. We include standard unit vectors  $e_1, \dots, e_{b(z_1)}$  in  $\mathbb{B}$ . Now, we have that  $A^j_{z_1}$  is not in the span of  $e_1, \dots, e_{b^*(z_1)}$ , so we include  $A^j_{z_1}$  in the basis  $\mathcal{B}$ . Next, for each standard unit vector  $e_{b^*(z_1)+1}, e_{b^*(z_1)+2}, \dots, e_{b^*(z_2)}$ , we in turn include it in  $\mathcal{B}$  if it is linearly independent from the vectors already in  $\mathcal{B}$ . This list of vectors may be empty, if for example  $b^*(z_1) = b^*(z_2)$ . Next, we have that  $A^j_{z_2}$  is not in the span of  $\mathbb{B}$ , since its  $(n - b^*(z_2))$ -suffix is not in the span of the  $(n - b^*(z_2))$ -suffix of  $A^j_{z_1}$  together with  $e_\ell$  for any  $\ell \leq b^*(z_2)$ . We repeat this process, resulting in a basis  $\mathcal{B}$  which contains  $A^j_{z_1}, \dots, A^j_{z_{r+1}}$  together with standard unit vectors. We let  $\mathcal{B}_i$  denote the  $i$ -th basis vector,  $i \in [n]$ , and we let  $T \subset [n]$  be the subset of  $i \in [n]$  of size  $r+1$  for which  $\mathcal{B}_i = A^j_{z_\ell}$  for some  $\ell \in [r+1]$ .

Let the random variable  $S$  denote the state of  $\text{Alg}$  at the beginning of the  $j$ -th pass. Let  $\mathcal{B} \cdot X$  denote the inner products of the vectors in  $\mathcal{B}$  with  $X$ . Let  $X \leq \alpha$  be the subset of the first  $\alpha$  coordinates of  $X$ , inclusive.

We have,

$$\begin{aligned}
I(S; X) &= I(S; \mathcal{B} \cdot X) \\
&= \sum_{i=1}^n I(S; \mathcal{B}_i \cdot X \mid \mathcal{B}_1 \cdot X, \dots, \mathcal{B}_{i-1} \cdot X) \\
&\geq \sum_{i \in T} I(S; \mathcal{B}_i \cdot X \mid \mathcal{B}_1 \cdot X, \dots, \mathcal{B}_{i-1} \cdot X) \\
&\quad + \sum_{i \notin T} I(S; \mathcal{B}_i \cdot X \mid \mathcal{B}_1 \cdot X, \dots, \mathcal{B}_{i-1} \cdot X) \\
(3.6) &= \sum_{\ell=1}^{r+1} I(S; A_{z_\ell}^j \cdot X \mid A_{z_1}^j \cdot X, \dots, A_{z_{\ell-1}}^j \cdot X, X \leq b^*(z_\ell))
\end{aligned}$$

where the first equality uses that  $\mathcal{B} \cdot X$  and  $X$  determine each other, the second equality uses the chain rule for mutual information, the first inequality uses that mutual information is non-negative, and the final equality uses the way in which our special basis  $\mathcal{B}$  was constructed.

Using the definition of mutual information,

$$\begin{aligned}
I(S; A_{z_\ell}^j \cdot X \mid A_{z_1}^j \cdot X, \dots, A_{z_{\ell-1}}^j \cdot X, X \leq b^*(z_\ell)) \\
&= H(A_{z_\ell}^j \cdot X \mid A_{z_1}^j \cdot X, \dots, A_{z_{\ell-1}}^j \cdot X, X \leq b^*(z_\ell)) \\
&\quad - H(A_{z_\ell}^j \cdot X \mid S, A_{z_1}^j \cdot X, \dots, A_{z_{\ell-1}}^j \cdot X, X \leq b^*(z_\ell)).
\end{aligned}$$

Recall that  $S$  was defined to be the state of  $Alg$  at the beginning of the  $j$ -th pass. Whenever  $Alg$  succeeds in outputting  $A_{z_\ell}^j \cdot x$  for some  $\ell$ , it does so using its state in the  $j$ -th pass after reading the first  $b(z_\ell)$  entries of  $X$ . This resulting state is a (possibly randomized) function of  $S$ , which is the state of  $Alg$  at the beginning of the  $j$ -th pass, together with  $X \leq b(z_\ell)$ . Since  $b(z_\ell) \leq b^*(z_\ell)$ , this resulting state is also a (possibly randomized) function of  $S$  together with  $X \leq b^*(z_\ell)$ , and so applying Fact 2.1, we have

$$\begin{aligned}
H(A_{z_\ell}^j \cdot X \mid S, A_{z_1}^j \cdot X, \dots, A_{z_{\ell-1}}^j \cdot X, X \leq b^*(z_\ell)) \\
(3.7) \leq H_b \left( \frac{1}{3} \right) + \frac{\log(p-1)}{3}.
\end{aligned}$$

Although we cannot lower bound  $H(A_{z_\ell}^j \cdot X \mid A_{z_0}^j \cdot X, \dots, A_{z_{\ell-1}}^j \cdot X, X \leq b^*(z_\ell))$  for each  $j$  individually, by (3.6) it will be enough to lower bound

$$\begin{aligned}
\sum_j H(A_{z_\ell}^j \cdot X \mid A_{z_0}^j \cdot X, \dots, A_{z_{\ell-1}}^j \cdot X, X \leq b^*(z_\ell)) \\
= \sum_{i \in T} H(\mathcal{B}_i \cdot X \mid \mathcal{B}_1 \cdot X, \dots, \mathcal{B}_{i-1} \cdot X, X \leq b^*(z_\ell)).
\end{aligned}$$

Note that  $X$  is uniformly random over a set of size at least

$$(3.8) \quad \frac{p^n}{12m^{k-1}} \cdot \frac{1}{2} = \frac{p^n}{24m^{k-1}},$$

and so we have,

$$\begin{aligned}
n \log p - (k-1) \log(m) - \log(24) &= H(X) \\
&= H(\mathcal{B}X) \\
&= \sum_{i \in T} H(\mathcal{B}_i \cdot X \mid \mathcal{B}_1 \cdot X, \dots, \mathcal{B}_{i-1} \cdot X) \\
&\quad + \sum_{i \notin T} H(\mathcal{B}_i \cdot X \mid \mathcal{B}_1 \cdot X, \dots, \mathcal{B}_{i-1} \cdot X) \\
&\leq \sum_{i \in T} H(\mathcal{B}_i \cdot X \mid \mathcal{B}_1 \cdot X, \dots, \mathcal{B}_{i-1} \cdot X) \\
&\quad + (n-r-1) \log p,
\end{aligned}$$

from which it follows that

$$\begin{aligned}
\sum_j H(A_{z_\ell}^j \cdot X \mid A_{z_0}^j \cdot X, \dots, A_{z_{\ell-1}}^j \cdot X, X \leq b^*(z_\ell)) \\
(3.9) \geq (r+1) \log p - O(k \log m).
\end{aligned}$$

Combining (3.6), (3.7), and (3.9),  $|S| \geq H(S) \geq I(S; X) \geq (r+1) \log p - O(k \log m) - (r+1)/3 \log(p-1) - O(r) \geq (r/2) \log p - O(k \log m)$ .

Hence, in either case, the size  $|S|$  of the memory contents must be at least  $(r/6) \log p - O(k \log(mn))$  bits, which completes the proof.

#### 4 Computing the Best Multi-Pass Algorithm

Here we give an algorithm for computing the optimal multi-pass streaming algorithm. The rough idea is that one can always push the breakpoints down and the boundary points to the right if possible. This leads to a greedy algorithm. We start with several lemmas.

**LEMMA 4.1.** (*Pushing Breakpoints Down*) *Consider a set of breakpoints  $i_0 = 0 < i_1 < i_2 < \dots < i_{k-1} < m = i_k$ , inducing subsets of rows  $A^1, \dots, A^k$  of a matrix  $A$  with  $k$ -pass streaming rank  $r$ . Suppose we replace  $i_j$  with  $i_j + 1$  for some  $j \in \{1, 2, \dots, k-1\}$ . Further, suppose after replacing  $i_j$  with  $i_j + 1$ , the matrix  $A^j$  still satisfies the following properties: each row  $i$  of  $A^j$  has a boundary point  $b(i)$  satisfying:*

1.  $b(i) \leq b(i+1)$ . Further, if  $j = 1$ , then  $b(i)$  is the rightmost non-zero entry of row  $i$ .
2. Letting  $L_i$  be the contiguous submatrix of  $A^j$  with lower left entry equal to  $A_{i_j, 1}$  and upper right entry equal to  $A_{i, b(i)}$ , we have  $\text{rank}(L_i) \leq r$ .

Further, if  $j > 1$ , then there exists a subspace  $U^j$  that  $r$ -adaptively fits  $A^j$ .

Then the new set of breakpoints and boundary points is a realization of the  $k$ -pass streaming rank of  $A$ .

*Proof.* First observe that only the submatrices  $A^j$  and  $A^{j+1}$  change by such a transformation, so all  $A^{j'}$  for

$j' \notin \{j, j+1\}$  still satisfy properties (1) and (2) above, together with the adaptive fitting property. By assumption,  $A^j$  also satisfies these properties. Thus, it suffices to show  $A^{j+1}$  satisfies properties (1) and (2) above, together with the adaptive fitting property.

Suppose  $U^{j+1}$  is a subspace which  $r$ -adaptively fits  $A^{j+1}$  before replacing  $i_j$  with  $i_j + 1$ . Then  $U^{j+1}$  still  $r$ -adaptively fits  $A^{j+1}$  after replacing  $i_j$  with  $i_j + 1$  at the same boundary points, with the boundary point for row  $i_j$  removed from the set of boundary points. Further,  $\text{rank}(L_i)$  is preserved for each row of  $A^{j+1}$  after replacing  $i_j$  with  $i_j + 1$ , using the same set of boundary points before replacing  $i_j$  with  $i_j + 1$ , except for the first row which was removed. It follows that  $A^{j+1}$  satisfies properties (1) and (2) above.

**DEFINITION 4.1.** (*Critical Breakpoints*) We say a set of breakpoints  $i_0 = 1 < i_1 < i_2 < \dots < i_{k-1} < m = i_k$  is **critical** if there exists a set of boundary points which is a realization of the  $k$ -pass streaming rank of  $A$ , but replacing any  $i_j$ ,  $j \in \{1, 2, \dots, k-1\}$  with  $i_j + 1$  is not a realization of the  $k$ -pass streaming rank of  $A$ .

By repeatedly applying Lemma 4.1, it follows that there exists a critical set of breakpoints. Further, Lemma 4.1 implies optimality of the following greedy algorithm for determining if the  $k$ -pass streaming rank is at most  $r$ : (1) choose  $i_1$  as large as possible so that the streaming rank of  $A^1$  is at most  $r$ , (2) having chosen  $i_1, \dots, i_j$  inductively, choose  $i_{j+1}$  as large as possible so that  $A^{j+1}$  has a set of boundary points  $b(i)$  satisfying:

1.  $b(i) \leq b(i+1)$ . Further, if  $j = 1$ , then  $b(i)$  is the rightmost non-zero entry of row  $i$ .
2. Letting  $L_i$  be the contiguous submatrix of  $A^j$  with lower left entry equal to  $A_{i_j, 1}$  and upper right entry equal to  $A_{i, b(i)}$ , we have  $\text{rank}(L_i) \leq r$ .

Further, if  $j > 1$ , then there exists a subspace  $U^j$  that  $r$ -adaptively fits  $A^j$ .

It remains to show how to implement this algorithm in polynomial time. To do so, we can, for each possible value of  $i_{j+1}$  with  $i_j < i_{j+1} \leq m$ , test if it satisfies the properties above (we can perform a binary search on possible  $i_{j+1}$  to further reduce the running time). To test property (1), note that the boundary points for  $j = 1$  are determined, so for each row we can just check if the matrix  $L_i$  has rank at most  $r$ . To test property (2), we need the following.

**LEMMA 4.2.** (*Pushing Boundary Points to the Right*) Given a subset of rows  $A^j$ ,  $j > 1$ , defined as the rows between  $i_{j-1} + 1$  and  $i_j$  in  $A$ , together with a set of boundary points  $b(i)$  for each row  $i$  of  $A^j$  with

$b(i) \leq b(i+1)$  for all  $i$  and  $\text{rank}(L_i) \leq r$  for all  $i$ , if there is a row  $i$  for which  $b(i) < b(i+1)$  and we replace  $b(i)$  with  $b(i)+1$ , then if  $\text{rank}(L_i) \leq r$  after this transformation, then if  $U^j$   $r$ -adaptively fits  $A^j$  before replacing  $b(i)$  with  $b(i)+1$ , then  $U^j$   $r$ -adaptively fits  $A^j$  after replacing  $b(i)$  with  $b(i)+1$ .

*Proof.* To prove the lemma it is enough to show that the  $(n - (b(i) + 1))$ -suffix of  $A_i^j$  is in the span of the  $(n - (b(i) + 1))$ -suffixes of a basis for  $U^j$ . This follows immediately, since the  $(n - b(i))$ -suffix of  $A_i^j$  is in the span of the  $(n - b(i))$ -suffixes of  $U^j$ .

Lemma 4.2 implies to test property (2), we can for each row  $i$  of  $A^j$ , choose  $b(i)$  to be the rightmost point so that  $\text{rank}(L_i) \leq r$ . After having done this, to check if there is a subspace  $U^j$  which  $r$ -adaptively fits  $A^j$  at the chosen boundary points, we can do so greedily. Namely, we initialize  $U^j$  to the empty space. When processing row  $i$ , we check if the  $(n - b(i))$ -suffix is in the span of the  $(n - b(i))$ -suffixes of a basis for  $U^j$ . If not, then we add  $A_i^j$  to  $U^j$ . If the dimension of  $U^j$  ever exceeds  $r$ , it follows that there is no space which  $r$ -adaptively fits  $A^j$  at these boundary points; otherwise we have found such a space.

By performing a binary search on  $r$  one can find the  $k$ -pass streaming rank, the set of breaking points, the boundary points, and the matrices  $L_i$  and (bases for) subspaces  $U^j$  above. This gives a polynomial time algorithm for finding the best  $k$ -pass streaming algorithm and establishes the following theorem.

**THEOREM 4.1.** (*Constructive Procedure for  $k$ -Passes*) Given an  $m \times n$  matrix  $A$ , we can in  $\text{poly}(mn)$  time find a set of breaking points and boundary points realizing the  $k$ -pass streaming rank  $r$  of  $A$ , together with the associated (bases for) subspaces  $U^j$  which  $r$ -adaptively fit each  $A^j$ ,  $j > 1$ .

**Remark.** By optimizing the running time of the procedure in Theorem 4.1, it is possible to achieve  $\tilde{O}(mn^3)$  time, where  $\tilde{O}(f) = f \cdot \text{poly}(\log f)$ . We note that this is done only once before the algorithm begins, and is only a factor of  $\tilde{O}(n)$  worse than computing the matrix rank, without using algorithms for fast matrix multiplication. Both this algorithm and algorithms for computing matrix rank can be further sped up by using algorithms for fast matrix multiplication [19].

We outline the idea for achieving  $\tilde{O}(mn^3)$  time. We can do binary search on the value of the multi-pass streaming rank  $r$ , incurring one  $\log(mn)$  factor. For a given value of  $r$ , after choosing the first  $j$  breakpoints  $i_1, \dots, i_j$ , we have to choose the next one  $i_{j+1}$ , which we guess as  $i_j + 2^\ell$  for non-negative integers  $\ell$  until we

fail. Then we perform binary search between  $i_j + 2^{\ell-1}$  and  $i_j + 2^\ell$ , incurring another  $\log(mn)$  factor and never overshooting the true breakpoint by more than a factor of 2. This is better than naïve binary search since the time to find the  $(j+1)$ -st breakpoint will be shown to be  $\tilde{O}((i_{j+1} - i_j)n^3)$  below, rather than  $\tilde{O}(mn^3)$  as one would obtain with a naïve binary search.

For each guess  $\ell$  and resulting submatrix, we push the boundary points as far to the right as possible so that the rank of the submatrix to the left and underneath the boundary points is at most  $r$ . Naïvely we obtain a sequence of matrices (the ones underneath and to the left of the current boundary point) with at most  $i_{j+1} - i_j$  rows for which the number  $c_t$  of columns in the submatrix corresponding to the  $t$ -th boundary point is increasing. In order to find the boundary points it will take time  $O(\sum_t (i_{j+1} - i_j)c_t^3)$ , since we have to try  $c_t$  matrices to greedily obtain the  $t$ -th boundary point, and each has cost  $O(i_{j+1} - i_j)c_t^2$  to compute its rank (which can be further sped up using fast matrix multiplication). Here,  $\sum_t c_t = n$  and so the cost is maximized at  $O(i_{j+1} - i_j)n^3)$ .

Having chosen the boundary points, we use Lemma 4.2 to check if we have the  $r$ -adaptive fitting property. This is done by greedily choosing the vectors of the suffixes and ensuring, given a new suffix, if it is in the span of previous ones. Naïvely, at some point in this process one will have up to  $i_{j+1} - i_j$  suffixes, each of dimension at most  $n$ , above the current row, and one needs to check if the current suffix is in their span, which can be viewed as checking if an  $s$ -dimensional vector is in the row span of an  $O(i_{j+1} - i_j) \times s$  matrix for some  $s \leq n$ . Naïvely, it takes  $O((i_{j+1} - i_j)n^2)$  time to check this. Using sketching techniques though, this time complexity can be reduced, e.g., a number of sketching techniques (see [20] for a survey) can reduce the matrix to an  $\tilde{O}(n) \times n$  matrix in  $\tilde{O}((i_{j+1} - i_j)n)$  time (e.g., the Subsampled Randomized Hadamard Transform), so that given the current suffix, one can check if it is in the span of previous suffixes in  $\tilde{O}(n^3)$  time. There are  $(i_{j+1} - i_j)$  suffixes to try which gives  $\tilde{O}((i_{j+1} - i_j)n^3)$  time.

Summing over all  $j$  gives a total running time of  $\tilde{O}(mn^3)$ .

## 5 Applications and Tradeoffs

We first consider several  $m \times n$  sketching matrices  $A$ , with  $m \leq n$ , and use our characterization to show there are no non-trivial tradeoffs for computing  $A \cdot x$ , even when allowing multiple passes over  $x$ . Note that one may want to compute a matrix-vector product with such sketching matrices *exactly* to preserve relative error for certain applications, such as for regression in linear

algebra, see, e.g., [20].

**THEOREM 5.1.** (*Sketching Lower Bounds*) *Any  $k$ -pass streaming algorithm for outputting the coordinates of  $A \cdot x$  in order, where  $A$  is an  $m \times n$  Gaussian (with entries rounded to  $O(\log(nm))$  bits of precision), Subsampled Randomized Hadamard Transform, or CountSketch matrix, requires  $Cm(\log(n))/k - O(k \log n)$  bits of space, for an absolute constant  $C > 0$ . In the case of CountSketch, we also assume  $m/k \geq C' \log(n)$  for an absolute constant  $C' > 0$ .*

*Proof.* By Theorem 3.2, it suffices to show the  $k$ -pass streaming rank of these matrices is  $\Omega(m/k)$ .

For any possibility of breakpoints of an  $m \times n$  matrix  $A$ , there necessarily exists an  $A^j$  with at least  $m/k$  rows. For this  $A^j$  consider the boundary point  $b(i)$  of the middle row  $i$  of  $A^j$ . Then either  $L_i$  has at least  $n/2 - 1$  columns, or the matrix  $P_i$  defined with corners at  $(i, b(i)+1)$  and  $(i_{j-1}+1, n)$  has at least  $n/2 - 1$  columns. Note that both  $L_i$  and  $P_i$  have at least  $m/(2k) - 1$  rows.

We now consider each of the types of matrices in the theorem statement. For Gaussian matrices, standard results imply that with high probability, any submatrix has full rank, and therefore  $\text{rank}(L_i)$  or  $\text{rank}(P_i)$  is at least  $m/(2k) - 1$ . If  $\text{rank}(L_i)$  is at least this large, this implies the  $k$ -pass streaming rank is  $\Omega(m/k)$ , by definition. On the other hand, if  $\text{rank}(P_i)$  is at least  $m/(2k) - 1$ , this implies any subspace which adaptively fits  $A^j$  must have dimension at least  $\text{rank}(P_i) \geq m/(2k) - 1$ , and so again the  $k$ -pass streaming rank is  $\Omega(m/k)$ .

For the Subsampled Randomized Hadamard Transform, we recall that these matrices can be factored as  $P \cdot H \cdot D$ , where  $D$  is a diagonal matrix with each diagonal entry equal to  $+1$  or to  $-1$ ,  $H$  is the  $n \times n$  Hadamard matrix, and  $P$  chooses  $m$  out of the  $n$  rows of  $H \cdot D$ . Note that  $\text{rank}(PHD) = \text{rank}(PH)$ . We use the following fact about Hadamard matrices [15].

**FACT 5.1.** (*Hadamard Submatrices*) *Every  $a \times b$  submatrix  $A$  of  $H$  has rank at least  $ab/n$ .*

Since  $PH$  is itself a submatrix of  $H$ , we can apply Fact (5.1) to conclude that either  $\text{rank}(L_i)$  or  $\text{rank}(P_i)$  is at least  $(n/2 - 1)(m/(2k) - 1)/n = \Omega(m/k)$ , and so as in the Gaussian case, its  $k$ -pass streaming rank is  $\Omega(m/k)$ .

In an  $m \times n$  CountSketch matrix  $A$ , in each column we uniformly choose the location of a single non-zero entry, and that entry is equal to either  $1$  or  $-1$  with equal probability. It follows that the rank of any submatrix of  $A$  is equal to the number of distinct columns. Define a *contiguous* submatrix of  $A$  to be a submatrix of  $A$  rooted at two entries  $(i, j)$ ,  $(i', j')$  in  $A$  with  $i < i'$  and  $j < j'$ , and containing all

entries  $(c, d)$  of  $A$  for which  $i \leq c \leq i'$  and  $j \leq d \leq j'$ . The number of contiguous submatrices of  $A$  is  $O(m^2 n^2)$ . For an  $r \times s$  contiguous submatrix, the expected number of distinct columns is independent of which  $r \times s$  contiguous submatrix chosen, so let us choose the upper left  $r \times s$  submatrix. The expected number of distinct columns is then  $\mathbf{E}[\sum_{i=1}^r X_i]$ , where  $X_i$  is an indicator that the column  $e_i$  or  $-e_i$  is chosen. We have  $\mathbf{E}[X_i] = 1 - (1 - 1/m)^s$ , and so

$$\mathbf{E}[\sum_{i=1}^r X_i] = r(1 - (1 - 1/m)^s).$$

In our case,  $r \geq m/(2k) - 1$  and  $s \geq n/2 - 1$ . Since  $m \leq n \leq 3s$ , it follows that  $\mathbf{E}[\sum_{i=1}^r X_i] = \Omega(r) = \Omega(m/k)$ . By Azuma's inequality,  $\sum_{i=1}^r X_i$  will be  $\Omega(m/k)$  with probability  $1 - \exp(-\Omega(m/k))$ . Assuming  $m/k \geq C' \log(n)$  for an absolute constant  $C' > 0$ , by a union bound we have for all contiguous submatrices with at least  $m/(2k) - 1$  rows and at least  $n/2 - 1$  columns, their rank is  $\Omega(m/k)$ . Hence, the  $k$ -pass streaming rank is  $\Omega(m/k)$ .

We now consider lower bounds for *any matrices* which satisfy the popular distributional Johnson Lindenstrauss Transform, which is a key tool in dimensionality reduction applications.

**DEFINITION 5.1.** *Let  $A$  be an  $m \times n$  matrix such that for any fixed vector  $x \in \mathbb{R}^n$ , one has*

$$\Pr[\|Ax\|_2^2 = (1 \pm \epsilon)\|x\|_2^2] \geq 1 - \delta.$$

*Note that  $A$  is a random variable, and the probability is taken over  $A$ . We call such an  $A$  a distributional Johnson-Lindenstrauss Transform.*

We will use the following fact.

**FACT 5.2.** *(see, e.g., [13, 14]) Suppose  $n \geq C\epsilon^{-2} \log(1/\delta)$ , where  $C > 0$  is a sufficiently large constant. Then there exists an absolute constant  $D > 0$  such that any distributional Johnson Lindenstrauss Transform  $A$  requires  $m \geq D\epsilon^{-2} \log(1/\delta)$ . We can assume  $D\epsilon^{-2} \log(1/\delta)$  is an even integer by weakening this lower bound by at most an additive 1.*

*We note that it is well-known that there exists an absolute constant  $C' \geq D > 0$ , such that  $m \leq C'\epsilon^{-2} \log(1/\delta)$  is achievable, though we will not use this fact below.*

Our main theorem is the following.

**THEOREM 5.2.** *Let  $\delta > 0$  be smaller than a sufficiently small constant  $\delta_0 > 0$ . Suppose  $n \geq \frac{2EC}{D}\epsilon^{-2} \log(1/\delta)$ ,*

*where  $C, D > 0$  are the constants of Fact 5.2, and  $E \geq D$  is an arbitrary constant. Let  $A$  be a distributional Johnson-Lindenstrauss transform with  $m = E\epsilon^{-2} \log(1/\delta)$ . Then any 1-pass streaming algorithm computing  $Ax$  requires  $\Omega(\epsilon^{-2} \log(1/\delta))$  words of space.*

*Proof.* Consider a distributional Johnson-Lindenstrauss Transform  $A$  with  $m = E\epsilon^{-2} \log(1/\delta)$ . We show that with non-zero probability over  $A$ , the streaming rank of  $A$  is  $\Omega(\epsilon^{-2} \log(1/\delta))$ , which implies by Theorem 1.1 that any 1-pass streaming algorithm computing  $Ax$  requires  $\Omega(\epsilon^{-2} \log(1/\delta))$  words of space.

For a row  $i$  of  $A$  we define  $e(i) = \max_{j \leq i} b(j)$ , where  $b(j)$  is the boundary of row  $j$ . Notice that for the  $k$ -pass streaming rank, we simply defined  $b(j) \geq b(i)$  whenever  $j \geq i$ , whereas for the 1-pass streaming rank we instead defined  $b(j)$  to be the rightmost non-zero entry of row  $j$ , and so we could have  $b(j) < b(i)$  for  $j > i$  in the single pass definition. However, we can w.l.o.g. assume  $b(j) \geq b(i)$  whenever  $j > i$  for our 1-pass definition as well since the maximum, over  $i$ , of the rank of a submatrix of  $A$  with corners  $(m, 1)$  and  $(i, e(i))$ , is at most the streaming rank. This follows since if  $b(j) > b(i)$  for a  $j < i$ , then the submatrix with corners  $(m, 1)$  and  $(i, b(j))$  has rank at most as large as the submatrix with corners  $(m, 1)$  and  $(j, b(j))$ , which is at most the streaming rank. We will thus lower bound the maximum, over  $i$ , of the rank of a submatrix of  $A$  with corners  $(m, 1)$  and  $(i, e(i))$  to obtain our  $\Omega(\epsilon^{-2} \log(1/\delta))$  lower bound.

We can assume, w.l.o.g., that  $E$  is an integer multiple of  $D$ , where  $D > 0$  is the constant of Fact 5.2. Indeed, since we are proving the theorem for an arbitrary integer constant  $E$ , we can pad  $A$  with zero rows to ensure this condition is satisfied.

Set  $t = 2E/D$ , which is an integer since  $D$  divides  $E$ . Our assumptions imply  $m/t$  is an integer, since  $m/t$  is equal to  $(D\epsilon^{-2} \log(1/\delta))/2$ , and recall in Fact 5.2 that  $D\epsilon^{-2} \log(1/\delta)$  is an even integer. Further, we can assume  $n/t$  is an integer, by at most reducing  $n$  by a factor of  $t$ , which is fine since we already assume  $n \geq tC\epsilon^{-2} \log(1/\delta)$  for a sufficiently large constant  $C > 0$ , and so by reducing  $n$  by a factor of  $t$ , we will still have  $n$  large enough to apply Fact 5.2.

Consider the sequence of entries in  $A$  of the form  $(im/t, in/t)$ , where  $i = 1, 2, \dots, t-1$ . For a given matrix  $A$ , say that it is *well-spread* if there exists an  $i$  for which  $e((i+1)m/t) - e(im/t) \geq n/t$ .

First, suppose with probability at least  $1/2$ , we have that  $A$  is not well-spread. Call this event  $\mathcal{E}$ . Conditioned on  $\mathcal{E}$ , then  $A$  necessarily has the property that all entries in the submatrix with corners  $((t-1)m/t, (t-1)n/t)$  and  $(1, n)$  are zero. Note that even conditioned on  $\mathcal{E}$ , the submatrix of  $A$  with corners

$((t-1)m/t, (t-1)n/t)$  and  $(m, n)$  is a distributional Johnson-Lindenstrauss Transform with probability at least  $1 - 2\delta$  for any fixed  $x$  whose support is in the set  $\{(t-1)n/t, (t-1)n/t+1, \dots, n\}$ . By Fact 5.2, since this submatrix has  $m/t$  rows, it satisfies

$$\frac{m}{t} \geq D\epsilon^{-2} \log\left(\frac{1}{2\delta}\right),$$

which for  $\delta \leq 1/8$ , implies

$$m \geq \frac{2tD}{3}\epsilon^{-2} \log\left(\frac{1}{\delta}\right),$$

which using our definition of  $t = 2E/D$  implies

$$m \geq \frac{4E}{3}\epsilon^{-2} \log\left(\frac{1}{\delta}\right),$$

which is a contradiction to  $m = E\epsilon^{-2} \log(1/\delta)$ .

Hence, we can assume with probability at least  $1/2$  that  $A$  is well-spread. By averaging, this implies there exists an  $i^*$  for which  $e((i^*+1)m/t) - e(i^*m/t)$  is at least  $m/t$ , with probability at least  $1/(2t)$ . Let  $\mathcal{F}$  denote the event that  $e((i^*+1)m/t) - e(i^*m/t) \geq m/t$ .

Consider those vectors  $x$  with support in the set  $S = \{e(i^*m/t) + 1, e(i^*m/t) + 2, \dots, e((i^*+1)m/t)\}$ . Consider the submatrix of  $A$  with columns in the set  $S$ . Then even conditioned on  $\mathcal{F}$ , this submatrix is a distributional Johnson-Lindenstrauss Transform with probability at least  $1 - 2t\delta$  for any fixed  $x$  with support in the set  $S$ . By Fact 5.2, the rank of  $A$  restricted to these columns must be at least  $D\epsilon^{-2} \log\left(\frac{1}{2t\delta}\right)$ . Note, though, that all entries in the submatrix in rows  $j$  for  $j < i^*m/t$  are zero. As there are only  $m/t$  rows between  $i^*m/t$  and  $(i^*+1)m/t$ , it follows that the rank of the submatrix of  $A$  with corners  $(m, e(i^*m/t) + 1)$  and  $((i^*+1)m/t, e((i^*+1)m/t))$  is at least

$$\begin{aligned} & D\epsilon^{-2} \log\left(\frac{1}{2t\delta}\right) - \frac{m}{t} \\ & \geq D\epsilon^{-2} \log\left(\frac{1}{\delta}\right) - D\epsilon^{-2} \log(2t) - \frac{mD}{2E} \\ & \geq \frac{D}{2}\epsilon^{-2} \log\left(\frac{1}{\delta}\right) - D\epsilon^{-2} \log\left(\frac{4E}{D}\right) \\ & = \Omega\left(\epsilon^{-2} \log\left(\frac{1}{\delta}\right)\right), \end{aligned}$$

where the final equality follows for  $\delta > 0$  smaller than a small enough constant  $\delta_0 > 0$ , since  $\frac{4E}{D}$  is a constant. As the rank of this submatrix lower bounds the streaming rank, the proof is complete.

We conclude with a largest possible separation for  $k$  versus  $k+1$  passes.

**THEOREM 5.3.** *For any  $k \geq 1$ , there exists a matrix  $A$  for which one can compute  $A \cdot x$  deterministically in  $k+1$  passes with  $O(1)$  words of space, but with only  $k$  passes one needs  $n/(18k) - O(k \log n)$  words of space.*

*Proof.* Let  $A$  be an  $n \times n$  matrix and suppose  $k \mid n$ . Suppose  $A$  is an “anti-block-diagonal” matrix, that is, it is an anti-diagonal matrix with  $k$  blocks, each of size  $n/k$ , along the anti-diagonal. Each block is an  $n/k \times n/k$  identity matrix added to an  $n/k \times n/k$  all 1s matrix. We will show that with  $k+1$  passes, one can compute  $A \cdot x$  deterministically with  $O(1)$  words of space. On the other hand, with only  $k$  passes one requires  $n/(18k) - O(k \log n)$  words of space. This is the largest possible separation (up to the constant 18 and the additive  $O(k \log n)$  factor) since one can compute  $A \cdot x$  for any  $n \times n$  matrix  $A$  in  $k$  passes using  $O(n/k)$  words of space.

Suppose one is given  $k+1$  passes to compute  $A \cdot x$ . In the first pass one computes the sum  $\sigma_1$  of the last  $n/k$  bits in  $x$ . In the second pass one computes both  $\sigma_2$ , the sum of the bits in positions  $n - 2n/k, n - (2n/k - 1), \dots, n - (n/k + 1)$  of  $x$ , and one also outputs each of the last  $n/k$  bits of  $x$  added with  $\sigma_1$ . In general, in the  $i$ -th pass,  $2 \leq i \leq k+1$ , one computes  $\sigma_i$ , the sum of the bits in positions  $n - in/k + 1, n - in/k + 2, \dots, n - (i-1)n/k$  (if  $i = k+1$  one does not compute such a sum), as well as outputs each of the bits of  $x$  in positions  $n - (i-1)n/k + 1, \dots, n - (i-2)n/k$  added to  $\sigma_{i-1}$ . Note that the total space is  $O(1)$  words.

On the other hand suppose one is given only  $k$  passes. By Theorem 3.2 it suffices to show the  $k$ -pass streaming rank is at least  $n/(3k)$ . Consider a set of breakpoints realizing the  $k$ -pass streaming rank. If the first breakpoint is larger than  $n/(2k)$ , then since  $b(1) = n$ , we have that  $\text{rank}(L_1) \geq n/(2k)$ , and we are done. Otherwise suppose that the first breakpoint is at most  $n/(2k)$ . Notice that between two breakpoints, there cannot be two blocks along the anti-diagonal each with  $n/(2k)$  rows, as otherwise if we consider the bottom row  $i$  of the upper block, either  $L_i$  has rank at least  $n/(2k)$ , or the matrix  $U_i$  with lower left corner  $(i, b(i))$  and upper right corner  $(j, n)$ , where  $j$  is the breakpoint defining this submatrix, has rank at least  $n/(2k) - 2 \geq n/(3k)$ , which says that any subspace which adaptively fits this submatrix has dimension at least  $n/(3k)$ .

It follows that between two breakpoints, there can be at most one block along the anti-diagonal with at least  $n/(2k)$  rows. But after the first pass there are still  $k$  such blocks, which means after the  $k$ -th pass there is still one such block, which implies the algorithm has not finished. It follows that the  $k$ -pass streaming rank is at least  $n/(3k)$ , as desired.

## 6 Future Directions and Open Questions

There are a number of questions for future work, and we conclude with a few such questions here:

1. It would be interesting to generalize our results to notions of approximate matrix-vector product. While approximation does not suffice for some applications, such as those which use subspace embeddings to achieve relative error, it could suffice for other applications. Quantifying an appropriate notion of approximation is an important task.
2. Although it may not be possible to output the entries of  $A \cdot x$  out of order in some applications, in others it may be possible. Our characterization extends to the model in which one is allowed to choose the ordering to output the coordinates of  $A \cdot x$ , in the preprocessing stage before seeing any particular  $x$ . Indeed, for each permutation of the rows of  $A$  we characterize the complexity as the streaming rank of that permutation of  $A$ , so the complexity equals the minimum over permutations, of the streaming rank of the permuted  $A$ . As an example, when  $A$  is a permutation matrix, one can permute it to the identity matrix to obtain constant streaming rank. Although our characterization applies, our preprocessing algorithm is no longer efficient as one may need to try many permutations of  $A$ . It would be interesting to determine the time complexity of finding a permutation of the rows of  $A$  which minimizes or nearly minimizes the streaming rank of the permuted  $A$ .

## References

- [1] Karl R. Abrahamson. Time-space tradeoffs for algebraic problems on general sequential machines. *J. Comput. Syst. Sci.*, 43(2):269–289, 1991.
- [2] Nir Ailon and Bernard Chazelle. The fast johnson-lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, 39(1):302–322, 2009.
- [3] Ziv Bar-Yossef. *The complexity of massive data set computations*. PhD thesis, University of California at Berkeley, 2002.
- [4] Ziv Bar-Yossef, Luca Trevisan, Omer Reingold, and Ronen Shaltiel. Streaming computation of combinatorial objects. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity, Montréal, Québec, Canada, May 21-24, 2002*, pages 165–174, 2002.
- [5] Paul Beame and Trinh Huynh. The value of multiple read/write streams for approximating frequency moments. *TOCT*, 3(2):6, 2012.
- [6] Paul Beame, T. S. Jayram, and Atri Rudra. Lower bounds for randomized read/write stream algorithms. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 689–698, 2007.
- [7] Jeremiah Blocki. *Usable Human Authentication: A Quantitative Treatment*. PhD thesis, Carnegie Mellon University, 2014.
- [8] Manuel Blum and Santosh Vempala. Publishable humanly usable secure password creations schemas. In *Proc. of the 3rd AAAI Conference on Human Computation and Crowdsourcing*, 2015.
- [9] Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.
- [10] Martin Grohe, André Hernich, and Nicole Schweikardt. Randomized computations on large data sets: Tight lower bounds. *CoRR*, abs/cs/0703081, 2007.
- [11] Martin Grohe, André Hernich, and Nicole Schweikardt. Lower bounds for processing data with few random accesses to external memory. *J. ACM*, 56(3), 2009.
- [12] Martin Grohe and Nicole Schweikardt. Lower bounds for sorting with few random accesses to external memory. In *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13-15, 2005, Baltimore, Maryland, USA*, pages 238–249, 2005.
- [13] T. S. Jayram and David P. Woodruff. Optimal bounds for johnson-lindenstrauss transforms and streaming problems with subconstant error. *ACM Transactions on Algorithms*, 9(3):26, 2013.
- [14] Daniel M. Kane, Raghu Meka, and Jelani Nelson. Almost optimal explicit johnson-lindenstrauss families. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011, Princeton, NJ, USA, August 17-19, 2011. Proceedings*, pages 628–639, 2011.
- [15] Satyanarayana V. Lokam. Spectral methods for matrix rigidity with applications to size-depth tradeoffs and communication complexity. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*, pages 6–15, 1995.
- [16] Yichao Lu, Paramveer S. Dhillon, Dean P. Foster, and Lyle H. Ungar. Faster ridge regression via the subsampled randomized hadamard transform. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 369–377, 2013.
- [17] Gilbert Strang. *Linear algebra and its applications*. Thomson, Brooks/Cole, Belmont, CA, 2006.
- [18] Joel A. Tropp. Improved analysis of the subsampled randomized hadamard transform. *Advances in Adaptive Data Analysis*, 3(1-2):115–126, 2011.
- [19] Virginia Vassilevska Williams. Multiplying matrices



faster than coppersmith-winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898, 2012.

- [20] David P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1-2):1–157, 2014.