Data Streams and Applications in Computer Science

David P. Woodruff
IBM Research Almaden
dpwoodru@us.ibm.com

Abstract

This is a short survey of my work in data streams and related applications, such as communication complexity, numerical linear algebra, and sparse recovery. The goal is give a non-technical overview of results in data streams, and highlight connections between these different areas. It is based on my Presburger lecture award given at ICALP, 2014.

1 The Data Stream Model

Informally speaking, a data stream is a sequence of data that is too large to be stored in available memory. The data may be a sequence of numbers, points, edges in a graph, and so on. There are many examples of data streams, such as internet search logs, network traffic, sensor networks, and scientific data streams (such as in astronomics, genomics, physical simulations, etc.). The abundance of data streams has led to new algorithmic paradigms for processing them, which often impose very stringent requirements on the algorithm's resources.

Formally, in the streaming model, there is a sequence of elements $a_1, \ldots, a_m$ presented to an algorithm, where each element is drawn from a universe $[n] = \{1, \ldots, n\}$. The algorithm is allowed a single or a small number of passes over the stream. In network applications, the algorithm is typically only given a single pass, since if data on a network is not physically stored somewhere, it may be impossible to make a second pass over it. In other applications, such as when data resides on external memory, it may be streamed through main memory a small number of times, each time constituting a pass of the algorithm.

The algorithm would like to compute a function or relation of the data stream. Because of the sheer size, for many interesting problems the algorithm is necessarily randomized and approximate in order to be efficient. One should note that the randomness is in the random coin tosses of the algorithm rather than in the stream. That is, with high probability over the coin tosses of the algorithm, the algorithm should correctly compute the function or relation for any stream that is presented to it. This is more robust than say, if the algorithm assumed particular orderings of the stream that could make the problem easier to solve.

In this survey we focus on computing or approximating *order-independent* functions $f(a_1, \ldots, a_m)$. A function is order-independent if applying any permutation to its inputs results in the same function value. As we will see, this is often the case in numerical applications, such as if one is interested in the number of distinct values in the sequence $a_1, \ldots, a_m$.

One of the main goals of a streaming algorithm is to use as little memory as possible in order to compute or approximate the function of interest. The amount of memory used (in bits) is referred to as the *space complexity* of the algorithm. While it is always possible for the algorithm to store the entire sequence $a_1, \ldots, a_m$, this is usually extremely prohibitive in applications. For example, internet routers often have limited resources; asking them to store a massive sequence of network traffic is infeasible. Another goal of streaming algorithms is their *processing time*, i.e., how often it takes to update their memory contents when presented with a new item in the stream. Often items in streams are presented at very high speeds and the algorithm needs to quickly update the data structures in its memory in order to be ready to process future updates.

For order-independent functions, we can think of the stream as an evolution of an underlying vector $x \in \mathbb{R}^n$. That is, x is initialized to the all zero vector, and when the item i appears in the stream, x undergoes the update

$$x_i \leftarrow x_i + 1,$$

that is, the items i are associated with coordinates of x, and insertions of items cause additive updates to x. The goal when computing an order-independent function f is to compute (or approximate with high probability) $f(x)$, where x is the vector at the end of the stream after all insertions have been performed.

2 Distinct Elements

One of the earliest works on streaming was that of Flajolet and Martin in 1985 [25], who studied the *distinct elements* problem. In this problem, the function $f(a_1, \ldots, a_m)$ is equal to

$$|\{a_1, \ldots, a_m\}|,$$

that is, the cardinality of the set of items $a_1, \ldots, a_m$, which is just the number of distinct items in the sequence $a_1, \ldots, a_m$. This problem is often called the F_0-problem, denoted $F_0(x)$, for reasons we will see in Section 3.

The F_0 problem has many applications in relational databases. One example is query optimization, in which a user wants to select records in a database whose attributes satisfy a certain predicate [56, 29]. This often involves decomposing the query into a sequence of simpler queries, so that the set of records resulting from the execution of these simpler queries coincides with the set of records obtained via executing the original query. One often would like to choose these simpler queries in such a way that the number of distinct records at each intermediate step is small. By solving the F_0 problem at intermediate steps one can choose how to decompose the query so as to reduce the overall computation time.

It is not too hard to show (see, e.g., [2]) that any algorithm which computes $F_0(x)$ exactly requires $\Omega(n)$ bits of storage. That is, up to a constant factor in the space complexity, the optimal algorithm is simply to maintain a bit vector $\hat{x}$ where $\hat{x}_i = 1$ if and only if item i has appeared in the stream.

Therefore we settle for outputting an approximation to $F_0(x)$ with high probability, that is, the algorithm should with probability at least 99%, output a number Z for which

$$(1-\epsilon)F_0(x) \leq Z \leq (1+\epsilon)F_0(x), \tag{1}$$

where $\epsilon > 0$ is a user-specified accuracy parameter.

It turns out that randomized approximation algorithms achieving (1) can be much more efficient. Namely, it is possible for such algorithms to achieve $O(1/\epsilon^2 + \log n)$ bits of space and $O(1)$ processing time per item [40]. Provided ϵ is not too small, this can provide an exponential improvement in space over the algorithm which simply stores $\hat{x}$.

One may ask how it is even possible to do better than the $\Theta(n)$ space bound of storing $\hat{x}$. We give a brief intuitive argument here. Suppose we were to choose a random hash function $h : [n] \to [B]$, for a number $B = O(1)$ of "hash buckets". For each of the B hash buckets b, we retain only a single bit which indicates whether or not for some item i we have seen in the stream it holds that $h(i) = b$. When seeing item i in the stream, we compute $h(i)$, and set the bit associated with $h(i)$ to equal 1 if it is currently equal to 0; otherwise we leave it set to 1. The idea is that if $F_0 \approx B$, we can estimate F_0 by looking at the fraction of the bits associated with buckets which are equal to 1. If h were truly random, the probability a hash bucket is equal to 0 is $(1-1/B)^{F_0}$, and so the expected number of buckets which are 0 is equal to

$$B \cdot (1-1/B)^{F_0}.$$

If $F_0 \approx B$, when inverting this expression one can obtain a reasonably accurate approximation to F_0. On the other hand, if $F_0 \approx 2B$, say, then if we randomly select half of the items in $[n]$ to apply the hash function to, we can estimate F_0 restricted to this half of items. Since we chose this half randomly, we would expect the true number of distinct items to be twice this value. Similarly, we can create estimates for all scales $F_0 \approx 2^i B$. Of course, this is just an intuition and only for a constant-factor approximation rather than a $(1+\epsilon)$-approximation; getting the optimal parameters requires additional tricks for which we refer the reader to [40].

Despite the efficiency of this algorithm, one could ask if the quadratic dependence on $1/\epsilon$ is necessary, or if this can be further reduced. This would make sense if highly accurate approximations are desired, e.g., if $\epsilon = .001$. It is known that the quadratic dependence is indeed necessary, and moreover, $\Omega(1/\epsilon^2 + \log n)$ bits of space is needed in order to achieve (1), showing the above algorithm is optimal up to constant factors [32, 57]. To prove the lower bound, a new problem called the Gap-Hamming Problem in communication complexity was introduced, as described below.

There is a standard way of proving lower bounds on the memory required of a data stream algorithm by using two-player communication complexity [2]. The basic idea is to have two players, named Alice and Bob, each hold inputs $x, y \in \{0,1\}^n$ to some communication problem g. Alice uses her input x to create a stream $s(x)$, while Bob uses his input to create a stream $s(y)$. If Alg is a streaming algorithm, Alice can run Alg on $s(x)$, and transmit the memory contents of Alg to Bob, who can continue the computation on $s(y)$. If from the output of Alg on the concatenation of streams $s(x)$ and $s(y)$, Bob can solve a problem $g(x,y)$, then the amount of communication must be at least the 1-*way communication complexity* of g, which is the minimum possible number of bits Alice can send to Bob to solve g (where the minimum is over all protocols correct on g with probability at least 99%, of the maximum number of bits Alice sends on any of her inputs and any of her random strings).

In the Gap-Hamming Problem, Alice and Bob are promised that the Hamming distance $\Delta(x,y)$ satisfies:

$$\Delta(x,y) > n/2 + \epsilon n, \text{ or } \Delta(x,y) < n/2 - \epsilon n,$$

We let $g(x,y) = 1$ in the former case, while $g(x,y) = 0$ in the latter case. It was shown in [32, 57] that the randomized 1-way communication complexity of this problem is $\Omega(1/\epsilon^2)$ bits. The proof was simplified in [35], and extended to 2-way communication (in which Alice and Bob can both send messages to each other) in [17], improving both upon the results for 1-way communication as well as earlier partial results for 2-way communication in [13, 14]. The work of [17] was later simplified in [53].

Using the reduction above, these communication lower bounds for the Gap-Hamming Problem imply that any streaming algorithm, even if allowed a constant number of passes, requires $\Omega(1/\epsilon^2)$ bits of memory in order to achieve the guarantee in (1). For the details of this reduction, we refer the reader to [57].

The Gap-Hamming Problem has emerged as an interesting problem in its own right, and has found applications in information complexity [16, 10], distributed functional monitoring [58], linear algebra [21], differential privacy [46], and lower bounds for graph sparsifiers [6].

3 Frequency Moments

The streaming model was revived in 1996 by Alon, Matias, and Szegedy [2] who studied the problem of estimating the *frequency moments* $F_p(x)$, where

$$F_p(x) = \sum_{i=1}^{n} x_i^p.$$

Equivalently, $F_p(x) = \|x\|_p^p$ is the p-th power of the ℓ_p-norm of x. Interpreting 0^0 as 0, it follows that F_0 coincides with the number of non-zero coordinates of x, that is, the number of distinct elements. In general, frequency moments F_p are useful for summarizing the skewness of an underlying empirical distribution [24], while applications of specific frequency moments are given below.

The second moment F_2 is used for computing self-join sizes in databases, and given that it is the square of the Euclidean norm, is a basic primitive in geometric and linear-algebraic streaming algorithms.

The first frequency moment F_1 is just $\sum_{i=1}^{n} x_i$, which is easy to compute by maintaining a counter in the stream. However, often one considers the *turnstile model* of data streams in which one also allows items to be deleted from the stream. That is, instead of having only updates of the form $x_i \leftarrow x_i + 1$, one also has updates of the form $x_i \leftarrow x_i - 1$. This model is useful for summarizing statistics of the difference of two streams, and was coined the turnstile model by Muthukrishnan who was inspired by monitoring statistics of people in a subway station who come and go through turnstiles [49]. In this model, computing F_1 is non-trivial [30, 39, 38] and has found applications to "robust" algorithms, such as robust regression [54, 19, 47, 59].

As in the case for estimating F_0, for estimating F_p one seeks to, with at least 99% probability, output a number Z for which

$$(1 - \epsilon)F_p \leq Z \leq (1 + \epsilon)F_p. \tag{2}$$

By now we are getting close to obtaining optimal bounds for this problem. If the reader has not seen this problem before, he/she may want to pause and conjecture how the space complexity of this problem should depend on p. We saw for $p = 0$ the space complexity is $\Theta(1/\epsilon^2 + \log n)$ bits.

It turns out that F_2 is the "breaking point" for this problem, in that for $0 \leq p \leq 2$, it is possible to achieve the guarantee of (2) using $O(\epsilon^{-2} \log n)$ bits of space [30, 43, 39, 38], but for any $p > 2$, a polynomial $n^{1-2/p}$ bits of space is required [15, 7, 28, 34]. There are near-matching algorithms for $p > 2$, optimal up to small factors in $\epsilon^{-1} \log n$ [23, 33, 8, 11, 48, 5, 26, 3], which have been shown to be optimal up to a constant factor in some regimes [45]. All the algorithms also have very efficient processing times, at most poly$(\log n)$, and sometimes even $O(1)$.

One technique we wish to highlight from the upper bounds is that of "sub-sampling" and finding the "heavy-hitters". This is the main ingredient in obtaining the algorithms for F_p, for $p > 2$, and has had a number of applications to obtaining algorithms for other problems in the streaming model, such as earth-mover distance [4], mixed norms [36], sampling in the turnstile model [48, 5, 37], sparse recovery [51], graph sparsifiers [41], and regression [59]. A very nice result of Braverman and Ostrovsky [12] shows how this technique can be used to near-optimally estimate a large class of functions of the form $\sum_{i=1}^{n} G(x_i)$ with various conditions on G. Note that for F_p, $G(a) = |a|^p$ for $a \in \mathbb{R}$.

The idea of sub-sampling is similar to what was described in Section 2 for the distinct elements problem. Namely, we randomly keep a 2^{-i} fraction of items in $[n]$, for each value of $i \in [\log n]$, and if S_i is the subset of items kept for a given i, we apply a hashing procedure only to those items in S_i. Here, though, the hashing is a bit different than that for the distinct elements problem. Namely, we use the CountSketch data structure proposed by Charikar, Chen, and Farach-Colton [18]; see also the related work of Thorup and Zhang [55]. For each item $j \in [n]$, we associate a random sign $\sigma_j \in \{-1, 1\}$. We also choose a random hash function $h : [n] \to [B]$, as in the distinct elements problem. Next, we randomly partition $[n]$ into B buckets using h, and for the k-th such bucket we maintain the counter:

$$c_k = \sum_{j \in S_i, h(j)=k} \sigma(j) \cdot x_j.$$

A nice property of this data structure is that we can obtain an unbiased estimate of x_j as $\sigma(j) \cdot c_{h(j)}$. Indeed, looking at the contents of the bucket $h(j)$, we have

$$\begin{aligned} \mathbf{E}[\sigma(j) \cdot c_{h(j)}] &= \mathbf{E}[\sigma(j)^2 x_j + \sum_{j \neq j', h(j')=h(j)} \sigma(j)\sigma(j') x_j x_{j'}] \\ &= x_j + \sum_{j \neq j', h(j')=h(j)} x_j x_{j'} \mathbf{E}[\sigma(j)\sigma(j')]] \\ &= x_j, \end{aligned}$$

where the second equality follows from linearity of expectation and that $\sigma(j)^2 = 1$ since $\sigma(j) \in \{-1, 1\}$, while the third equality follows from the fact that for $j \neq j'$, $\mathbf{E}[\sigma(j)\sigma(j')] = 0$. The role of h is not so clear from the above analysis of the

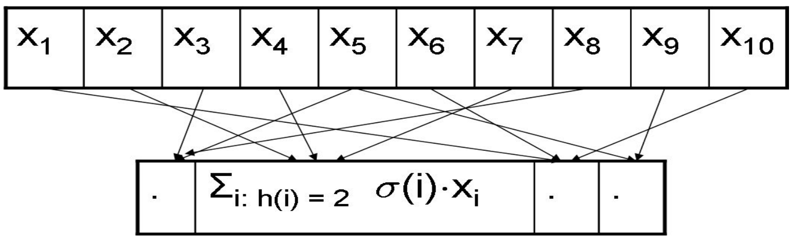

Figure 1: The CountSketch data structure with $n = 10$.

expectation, but becomes clear when looking at the variance. Indeed, one can show that the variance of the estimate is $O(F_2(x^{S_i})/B)$, where x^{S_i} denotes the vector for which $x_j^{S_i} = x_j$ for $j \in S_i$, and $x_j^{S_i} = 0$ for $j \notin S_i$.

The above data structure, when combined with sub-sampling, can be used to obtain the guarantee of (2). This data structure can also be used to find "heavy hitters", since if x_j^2 is much larger than $O(F_2(x)/B)$, one can obtain an accurate estimate to $|x_j|$. By performing independent repetitions of this data structure, one can in fact recover the indices j for which $|x_j|$ is large.

Another feature is that this data structure is a *linear map*. That is, it can be expressed as $S \cdot x$, where S is a matrix with B rows and n columns. Given the vector x as input, the data structure simply performs matrix-vector multiplication. Since the mapping is linear, it is easy to update in the presence of insertions and deletions of items in the stream. Indeed, if $x_i \leftarrow x_i + 1$, this corresponds to $S \cdot x \leftarrow S \cdot x + S^i$, where S^i is the i-th column of S. Similarly, if $x_i \leftarrow x_i - 1$, this corresponds to $S \cdot x \leftarrow S \cdot x - S^i$. This makes the above algorithms particularly useful for processing streams in the turnstile model. This property also is very useful for numerical linear algebra applications, as we show in Section 5.

4 Characterization of Turnstile Streaming Algorithms

As noted, the algorithm for F_p can actually be represented as computing $S \cdot x$, where S is a matrix drawn from a structured family of random matrices *before* *the stream begins*. The algorithm also works in the turnstile model. Curiously, all known algorithms in the turnstile model have the following form:

1. Choose a random matrix S independent of x before the stream begins.
2. Maintain the "linear sketch" $S \cdot x$ in the stream.
3. Output an arbitrary function of $S \cdot x$.

One cannot help but ask if the optimal algorithm for computing *any function or relation* in the turnstile model has the above form.

In recent work [44] with Li and Nguyen we show that the answer is yes, up to a $\log n$ factor. More precisely, up to a $\log n$ factor in the space complexity, the optimal algorithm samples a matrix S uniformly from $O(n \log n)$ hardwired matrices before the stream begins, maintains $S \cdot x$ while processing the stream, and then outputs a function of $S \cdot x$ at the end of the stream.

We do want to point out some caveats with this result. One is that the algorithm can't necessarily store S in small space, nor compute the output function given $S \cdot x$ in small space. This can be handled by allowing the algorithm to be non-uniform, so that the algorithm need only sample uniformly from a fixed set of $O(n \log n)$ hardwired matrices. One also needs to hardwire all the possible answers given $S \cdot x$. Despite this, it turns out that when proving lower bounds these restrictions don't matter, and the above characterization gives a simpler proof strategy for proving lower bounds than using 1-way communication complexity, the latter of which was discussed in Section 2. Indeed, this characterization allows one to prove lower bounds for linear sketches, which then imply the same lower bounds in the streaming model. We refer the reader to [44] for further details.

5 An Application to Linear Algebra

One well-studied problem in the numerical linear algebra community is the least squares regression problem. The goal of this line of work is to fit a number of observed points to a line, or more generally to a hyperplane, while minimizing the sum of squared vertical distances to the hyperplane. Cutting to the chase, one can express the least squares regression problem using the following matrix notation. Given an $n \times d$ matrix A and an $n \times 1$ vector b, compute

$$\mathrm{argmin}_{x \in \mathbb{R}^d} \|Ax - b\|_2.$$

Typically $n \gg d$, i.e., the problem is over-constrained and there need not exist a solution to the equation $Ax = b$. We thus settle for a randomized approximation: with probability at least 99%, output a vector $x' \in \mathbb{R}^d$ for which

$$\|Ax' - b\|_2 \leq (1 + \epsilon)\min_x \|Ax - b\|_2. \tag{3}$$

A beautiful work of Tamás Sarlós [52] observed that if S is a random projection onto a low dimensional subspace, one can apply the following "Sketch and Solve Paradigm":

1. Compute $S \cdot A$ and $S \cdot b$.
2. Output $x' = \text{argmin}_{x \in \mathbb{R}^d} \|SAx - Sb\|_2$.

The role of S in the above is to reduce A from an $n \times d$ matrix to an $n' \times d$ matrix where $n' \ll n$ and often is independent of n. Similarly, the vector b is replaced by an $n' \times 1$ vector. Thus, the "hard part" of solving the regression problem is done in the much lower-dimensional sketch space, replacing A with SA and b with Sb, for which classical algorithms are now more efficient. One can choose S so that SA is a $\text{poly}(d/\epsilon) \times d$ matrix.

The bottleneck in the approach above is computing the matrix-matrix product $S \cdot A$. Sarlós observed that if one chooses S to be a *fast Johnson-Lindenstrauss transform* [1], then one can compute $S \cdot A$ in only $O(nd \log n)$ time, leading to an overall algorithm for regression in $O(nd \log n) + \text{poly}(d/\epsilon)$ time. If $n \gg d$, this is a considerable improvement over the classical algorithm for least squares regression which takes $\Theta(nd^2)$ time (or $O(nd^{1.376})$ using fast matrix multiplication).

In recent work with Clarkson [22], we show that if instead of choosing S to be a fast Johnson-Lindenstrauss transform, if we choose S to be a CountSketch matrix, as described in Section 3, the correctness of the Sketch and Solve Paradigm still holds, that is, (3) holds. Moreover, SA is a $\text{poly}(d/\epsilon) \times d$ matrix, with a larger $\text{poly}(d/\epsilon)$ factor, but the regression problem in the sketch space $\min_x \|SAx - Sb\|_2$ can be solved much more efficiently than in the original space.

Importantly, the matrix-matrix product $S \cdot A$ can now be computed in only $O(\text{nnz}(A))$ time, where $\text{nnz}(A)$ denotes the number of non-zero entries of A. This gives an overall running time for regression of $O(\text{nnz}(A)) + \text{poly}(d/\epsilon)$. In a number of applications A is a sparse matrix, with $O(n)$ non-zero entries, and this gives overall running time $O(n) + \text{poly}(d/\epsilon)$, improving the $O(nd \log n) + \text{poly}(d/\epsilon)$ time algorithm one would get using fast Johnson Lindenstrauss transforms. The running time is also optimal for any algorithm achieving the guarantee of (3), up to the $\text{poly}(d/\epsilon)$ factors. A number of followup works improved the $\text{poly}(d/\epsilon)$ factors [9, 42, 47, 50], making the algorithms applicable even when n is not too much larger than d.

6 Conclusions

Techniques for data streams give efficient ways of compressing big data, which is a broadly applicable goal in computer science. We have seen one application of the

CountSketch data structure to speeding up algorithms in numerical linear algebra. Another application of CountSketch is to sparse recovery, or compressed sensing. Here there is an unknown vector x that is known to be well-approximated by a sparse vector y, and the goal is to recover an approximation to y given only a linear sketch $S \cdot x$ of x for a matrix S with a small number of rows of the user's choice. Letting S be the matrix implementing CountSketch, or variants of this, leads to algorithms for sparse recovery achieving the optimal number of measurements in the classical randomized setting [27], as well as further improvements when one allows the rows of S to be chosen adaptively [31].

There are many other applications of data streams to computer science which I did not cover, such as to graph algorithms (e.g., lower bounds for sparsification [6]) and machine learning (e.g., sampling techniques for classification and minimum enclosing ball [20]). Given the nature of this survey, it is somewhat focused on my own contributions. There are many other extremely interesting topics in data streams I did not cover, but hopefully this at least gives a glimpse into the techniques developed in this area.

References

[1] Nir Ailon and Bernard Chazelle. The fast Johnson–Lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, 39(1):302–322, 2009.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[3] Alexandr Andoni. High frequency moment via max stability. Available at http://web.mit.edu/andoni/www/papers/fkStable.pdf.

[4] Alexandr Andoni, Khanh Do Ba, Piotr Indyk, and David P. Woodruff. Efficient sketches for earth-mover distance, with applications. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 324–330, 2009.

[5] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *FOCS*, pages 363–372, 2011.

[6] Alexandr Andoni, Robert Krauthgamer, and David P. Woodruff. The sketching complexity of graph cuts. *CoRR*, abs/1403.7058, 2014.

[7] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *JCSS*, 68(4):702–732, 2004.

[8] Lakshminath Bhuvanagiri, Sumit Ganguly, Deepanjan Kesh, and Chandan Saha. Simpler algorithm for estimating frequency moments of data streams. In *SODA*, pages 708–713, 2006.

[9] Jean Bourgain and Jelani Nelson. Toward a unified theory of sparse dimensionality reduction in euclidean space. *CoRR*, abs/1311.2542, 2013.

[10] Mark Braverman, Ankit Garg, Denis Pankratov, and Omri Weinstein. Information lower bounds via self-reducibility. In *Computer Science - Theory and Applications - 8th International Computer Science Symposium in Russia, CSR 2013, Ekaterinburg, Russia, June 25-29, 2013. Proceedings*, pages 183–194, 2013.

[11] Vladimir Braverman and Rafail Ostrovsky. Recursive sketching for frequency moments. *CoRR*, abs/1011.2571, 2010.

[12] Vladimir Braverman and Rafail Ostrovsky. Zero-one frequency laws. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 281–290, 2010.

[13] Joshua Brody and Amit Chakrabarti. A multi-round communication lower bound for gap Hamming and some consequences. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*, pages 358–368, 2009.

[14] Joshua Brody, Amit Chakrabarti, Oded Regev, Thomas Vidick, and Ronald de Wolf. Better gap-Hamming lower bounds via better round elimination. In *RANDOM*, pages 476–489, 2010.

[15] Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *CCC*, pages 107–117, 2003.

[16] Amit Chakrabarti, Ranganath Kondapally, and Zhenghui Wang. Information complexity versus corruption and applications to orthogonality and gap-Hamming. In *RANDOM*, pages 483–494, 2012.

[17] Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of gap-Hamming-distance. *SIAM J. Comput.*, 41(5):1299–1317, 2012.

[18] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

[19] Kenneth L. Clarkson, Petros Drineas, Malik Magdon-Ismail, Michael W. Mahoney, Xiangrui Meng, and David P. Woodruff. The fast Cauchy transform and faster robust linear regression. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 466–477, 2013.

[20] Kenneth L. Clarkson, Elad Hazan, and David P. Woodruff. Sublinear optimization for machine learning. *J. ACM*, 59(5):23, 2012.

[21] Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 205–214, 2009.

[22] Kenneth L Clarkson and David P Woodruff. Low rank approximation and regression in input sparsity time. In *STOC*, 2013.

[23] Don Coppersmith and Ravi Kumar. An improved data stream algorithm for frequency moments. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 151–156, 2004.

[24] David J. DeWitt, Jeffrey F. Naughton, Donovan A. Schneider, and S. Seshadri. Practical skew handling in parallel joins. In *18th International Conference on Very Large Data Bases, August 23-27, 1992, Vancouver, Canada, Proceedings.*, pages 27–40, 1992.

[25] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[26] Sumit Ganguly. Polynomial estimators for high frequency moments. *CoRR*, abs/1104.4552, 2011.

[27] Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss. For-all sparse recovery in near-optimal time. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 538–550, 2014.

[28] Andre Gronemeier. Asymptotically optimal lower bounds on the nih-multi-party information complexity of the and-function and disjointness. In *STACS*, pages 505–516, 2009.

[29] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Arun N. Swami. Selectivity and cost estimation for joins based on random sampling. *J. Comput. Syst. Sci.*, 52(3):550–569, 1996.

[30] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

[31] Piotr Indyk, Eric Price, and David P. Woodruff. On the power of adaptivity in sparse recovery. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 285–294, 2011.

[32] Piotr Indyk and David P. Woodruff. Tight lower bounds for the distinct elements problem. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 283–288, 2003.

[33] Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *STOC*, pages 202–208, 2005.

[34] T. S. Jayram. Hellinger strikes back: A note on the multi-party information complexity of and. In *APPROX-RANDOM*, pages 562–573, 2009.

[35] T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of Hamming distance. *Theory of Computing*, 4(1):129–135, 2008.

[36] T. S. Jayram and David P. Woodruff. The data stream space complexity of cascaded norms. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 765–774, 2009.

[37] Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 49–58, 2011.

[38] Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 745–754, 2011.

[39] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1161–1178, 2010.

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

[41] Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *CoRR*, abs/1407.1289, 2014.

[42] Mu Li, Gary L. Miller, and Richard Peng. Iterative row sampling. In *FOCS*, pages 127–136, 2013.

[43] Ping Li. Estimators and tail bounds for dimension reduction in l_α $(0 < \alpha \leq 2)$ using stable random projections. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 10–19, 2008.

[44] Yi Li, Huy L. Nguyen, and David P. Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *STOC*, pages 174–183, 2014.

[45] Yi Li and David P. Woodruff. A tight lower bound for high frequency moment estimation with small error. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, pages 623–638, 2013.

[46] Andrew McGregor, Ilya Mironov, Toniann Pitassi, Omer Reingold, Kunal Talwar, and Salil P. Vadhan. The limits of two-party differential privacy. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:106, 2011.

[47] Xiangrui Meng and Michael W. Mahoney. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 91–100, 2013.

[48] Morteza Monemizadeh and David P. Woodruff. 1-pass relative-error l_p-sampling with applications. In *SODA*, 2010.

[49] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[50] Jelani Nelson and Huy L Nguyên. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *FOCS*, 2013.

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

[52] T. Sarlós. Improved approximation algorithms for large matrices via random projections. In *FOCS*, 2006.

[53] Alexander A. Sherstov. The communication complexity of gap Hamming distance. *Theory of Computing*, 8(1):197–208, 2012.

[54] Christian Sohler and David P. Woodruff. Subspace embeddings for the l_1-norm with applications. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 755–764, 2011.

[55] Mikkel Thorup and Yin Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 615–624, 2004.

[56] Kyu-Young Whang, Brad T. Vander Zanden, and Howard M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Trans. Database Syst.*, 15(2):208–229, 1990.

[57] David P. Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 167–175, 2004.

[58] David P. Woodruff and Qin Zhang. Tight bounds for distributed functional monitoring. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 941–960, 2012.

[59] David P. Woodruff and Qin Zhang. Subspace embeddings and l_p-regression using exponential random variables. In *COLT 2013 - The 26th Annual Conference on Learning Theory, June 12-14, 2013, Princeton University, NJ, USA*, pages 546–567, 2013.