Join Sizes, Frequency Moments, and Applications

Graham Cormode[1] and Minos Garofalakis[2]

[1] University of Warwick, Department of Computer Science, Coventry CV4 7AL, UK. Email: G.Cormode@warwick.ac.uk
[2] Technical University of Crete, School of Electronic and Computer Engineering, Chania 73100, Greece. Email: minos@softnet.tuc.gr

1 Introduction

In this chapter, we focus on a set of problems chiefly inspired by the problem of estimating the *size of the (equi-)join* between two relational data streams. This problem is at the heart of a wide variety of other problems, both in databases/data streams and beyond. Given two relations, R and S, and an attribute a common to both relations, the *equi-join* between the two relations, $R \bowtie S$, consists of one tuple for each $r \in R$ and $s \in S$ pair such that $r.a = s.a$. Estimating the join size between pairs of relations is a key component in designing an efficient execution plan for a complex SQL query that may contain arbitrary combinations of selection, projection, and join tasks; as such, it forms a critical part of database *query optimization* [15]. The results can also be employed to provide *fast approximate answers* to user queries, to allow, for instance, interactive exploration of massive data sets [12]. The ideas discussed in this chapter are directly applicable in a streaming context, and, additionally, within traditional database management systems where: (1) relational tables are *dynamic*, and queries must be tracked over the stream of updates generated by tuple insertions and deletions; or, (2) relations are truly massive, and *single-pass algorithms* are the only viable option for efficient query processing. Knowing the join size is also a important problem at the heart of a variety of other problems, such as building histogram and wavelet representations of data, and can be applied to problems such as finding frequent items and quantiles, and modeling spatial and high-dimensional data.

Many problems over streams of data can be modeled as problems over (implicit) vectors which are defined incrementally a continuous stream of data updates. For example, given a stream of communications between pairs of people (such as records of telephone calls between a caller and callee), we can capture information about the stream in a vector, indexed by the (number of the) calling party, and recording, say, the number of calls made by that caller.

Each new call causes us to update one entry in this vector (i.e., incrementing the corresponding counter). Queries about specific calling patterns can often be rendered into queries over this vector representation: For instance, to find the number of distinct callers active on the network, we must count the number of non-zero entries in the vector. Similarly, problems such as computing the number of calls made by a particular number (or, range of numbers), the number of callers who have made more than 50 calls, the median number of calls made and so on, can all be posed as function computations over this vector. In an even more dynamic setting, such streaming vectors could be used to track the number of *active* TCP connections in a large Internet Service Provide (ISP) network (say, per source IP address); thus, a TCP-connection open (close) message would have to increment (respectively, decrement) the appropriate counter in the vector.

Translating the join-size problem into the vector setting, we observe that each relation can essentially be represented by a *frequency-distribution vector*, whose i^{th} component counts the number of occurences of join-attribute value i in the relation. (Without loss of generality, we assume the join attributes to range over an integer domain $[U] = \{1, \ldots, U\}$, for some large U.[3]) The join size $|R \bowtie S$ between two streaming relations R and S corresponds to computing the *inner-product* of their frequency-distribution vectors, that is, the sum of the product of the counts of i in R and S over all $i \in [U]$. A special case of this query is the *self-join size* $|R \bowtie R|$: the size of the join between a relation R and itself. This is the sum of the squares of the entries in the corresponding frequency-distribution vector, or, equivalently, the square of its Euclidean (i.e., L_2) norm.

The challenge for designing effective and scalable streaming solutions for such problems is that it is not practical to materialize this vector representation; both the size of the input data stream and the size of the "universe" from which items in the stream are drawn can grow to be very large indeed. Thus, naive solutions that employ $O(U)$ space or time over the update stream are not feasible. Instead, we adopt a paradigm of based on *approximate, randomized estimation algorithms* that can produce answers to such queries with provable guarantees on the accuracy of the approximation while using only small space and time per streaming update.

Beyond join-size approximations, efficiently estimating such vector inner-products and norms has a wide variety of applications in streaming computation problems, including approximating range-query aggregates, quantiles, and heavy-hitter elements, and building approximate histograms and wavelet representations. We briefly touch upon some of these applications later in this chapter, while later chapters also provide more detailed treatments of specific applications of the techniques. Our discussion in this chapter focuses

[3] While our development here assumes a known upper bound on the attribute universe size U, this is not required: U can be learned adaptively over the stream using standard tricks (see, e.g., [13]).

on efficient, *sketch-based* streaming algorithms for join-size and self-join-size estimation problems, based on two influential papers by Alon, Matias, and Szegedy [3], and Alon, Gibbons, Matias, and Szegedy [2]. The remainder of the chapter is structured as follows.

2 Preliminaries and Problem Formulation

Let $\boldsymbol{x}$ be a (large) vector being defined incrementally by a stream of data updates. We adopt the most general (so-called, *"turnstile"* [20]) model of streaming data, where each update in the stream is a pair (i, c), where $i \in [U]$ is the index of the entry being updated and c is a positive or negative number denoting the magnitude of the update; in other words, the (i, c) update sets $\boldsymbol{x}[i] = \boldsymbol{x}[i] + c$. Allowing c values to be either positive or negative implies that $\boldsymbol{x}$ vector entries can decrease as well as increase. Thus, this model allows us to easily represent the "departure" of items (e.g., closed TCP connections) as well as their arrival.

Definition 1 (Join Size). *The* (equi-)join size *of two streaming relations with frequency-distribution vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ (over universe $[U]$) is exactly the* inner product *of $\boldsymbol{x}$ and $\boldsymbol{y}$, defined as $\boldsymbol{x} \cdot \boldsymbol{y} = \sum_{i=1}^{U} \boldsymbol{x}[i]\boldsymbol{y}[i]$.*

(We use the terms "join size" and "inner product" interchangeably in the remainder of this chapter.) The special case of *self-join size* (i.e., inner-product of a vector with itself) is closely related to the notion of *frequency moments* of a data distribution, which can be defined using the same streaming model and concepts. More specifically, let $\boldsymbol{x}$ denote the frequency-distribution vector for a stream R of items (from domain $[U]$); that is, $\boldsymbol{x}[i]$ denotes the number of occurrences of item i in the R stream. Then, the p^{th} *frequency moment* of stream R is given by $F_p(\boldsymbol{x}) = \sum_{i=1}^{U} \boldsymbol{x}[i]^p$.

Observe that F_1 is simply the length of the stream R (i.e., the total number of observed items), and can be easily computed with a single counter. F_0 is the number of distinct items in the R stream (the size of the *set* of items that appear), and is the focus of other chapters in this volume. F_2, the second frequency moment, is also known as the *repeat rate* of the sequence, or as *"Gini's index of homogeneity"* — it forms the basis of a variety of data analysis tasks, and can be used to compute the *surprise index* [14] and the *self-correlation* of the stream.

We extend the definition of frequency moments to the arrival vector model. Now, $F_p(\boldsymbol{x}) = \sum_{i=1}^{U} \boldsymbol{x}[i]^p$; thus, $F_p(\boldsymbol{x}) = \|\boldsymbol{x}\|_p^p$, where $\|\cdot\|_p$ denotes the L_p vector norm (see also the chapter of Cormode and Indyk later in this volume). In particular, $F_2(\boldsymbol{x})$ is the self-join size of a relation whose characteristic vector is $\boldsymbol{x}$, and $\sqrt{F_2(\boldsymbol{x})}$ is the L_2, or Euclidean, norm of the vector $\boldsymbol{x}$.

We give the definition of other relevant functions:

Definition 2. *The* vector distance *between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, both of dimensionality U is given by $\|\boldsymbol{x} - \boldsymbol{y}\|_2 = \sqrt{F_2(\boldsymbol{x} - \boldsymbol{y})}$.*

Definition 3. *An* (ϵ, δ)-relative approximation *of a value* X *returns an answer* x *such that, with probability at least* $1 - \delta$*,*

$$(1 - \epsilon)X \leq x \leq (1 + \epsilon)X$$

Definition 4 (k-wise independent hash functions.). *A family of hash functions* $\mathcal{H}$ *mapping items from* X *onto a set* Y *is said to be* k-wise independent *if, over random choices of* $h \in \mathcal{H}$*, we have*

$$\mathsf{Pr}[h(x_1) = h(x_2) = \ldots = h(x_k)] = \frac{1}{|Y|^k}$$

In other words, for up to k items, we can treat the results of the hash function as independent random events, and reason about them correspondingly. Here, we will make use of families of 2-wise (pairwise) independent hash functions, and 4-wise independent hash functions. Such hash functions are easy to implement: the family $\mathcal{H}_2 = \{ax + b \mod P \mod |Y|\}$, where P is a prime and a and b are picked uniformly at random from $0 \ldots P - 1$ is pairwise independent onto $\{0 \ldots |Y| - 1\}$ [5, 19]. More generally, the family

$$\mathcal{H}_k = \{\sum_{i=0}^{k-1} c_i x^i \mod P \mod |Y|\}$$

is k-wise independent for c_is picked uniformly from $\{0 \ldots |Y| - 1\}$ [23]. Various efficient implementations of such functions have been given, especially for the case of $k = 2$ and $k = 4$ [21, 22].

3 AMS Sketches

In their 1996 paper, Alon, Matias and Szegedy gave an algorithm to give an (ϵ, δ)-approximation of the self-join size. The algorithm computes a data structure, where each entry in the data structure is computed through an identical procedure but with a different 4-wise independent hash function for each entry. Each entry can be used to find an estimate of the self-join size that is correct in expectation, but can be far from the correct value. Carefully combining all estimates gives a result that is an (ϵ, δ)-approximation as required. The resulting data structure is often called an *AMS (or, "tug-of-war") sketch*, since the data structure concisely summarizes, or 'sketches' a much larger amount of information.

To build one element of the sketch, the algorithm takes a 4-wise hash function $h : [1 \ldots U] \rightarrow \{-1, +1\}$ and computes $Z = \sum_{i=1}^{U} h(i)\boldsymbol{x}[i]$. Note that this is easy to maintain under the turnstile streaming model: initialize $Z = 0$, and for every update in the stream (i, c) set $Z = Z + c * h(i)$. This algorithm is given in pseudo-code as UPDATE in Figure 1.

UPDATE(i, c, z)
Input: item i, count c, sketch z
1: **for** $j = 1$ to w **do**
2: **for** $k = 1$ to d **do**
3: $z[j][k] += h_{j,k}(i) * c$

ESTIMATE$F_2(z)$
Input: sketch z
1: **Return** ESTIMATEJS(z, z)

ESTIMATEJS(x, y)
Input: sketch x, sketch y
Output: estimate of $\boldsymbol{x} \cdot \boldsymbol{y}$
1: **for** $j = 1$ to w **do**
2: $avg[j] = 0$;
3: **for** $k = 1$ to d **do**
4: $avg[j] += x[j][k] * y[j][k]/w$;
5: **Return**(median(avg))

Fig. 1. AMS Algorithm for estimating join and self-join size

3.1 Second Frequency Moment Estimation

To estimate the self-join size, we compute Z^2.

Lemma 1. $\mathsf{E}(Z^2) = F_2(\boldsymbol{x})$

Proof.

$$\begin{aligned}\mathsf{E}(Z^2) &= \mathsf{E}((\sum_{i=1}^{U} h(i)\boldsymbol{x}[i])^2) \\ &= \mathsf{E}(\sum_{i=1}^{U} h(i)^2\boldsymbol{x}[i]^2) + \mathsf{E}\sum_{1\le i<j\le U} 2h(i)h(j)\boldsymbol{x}[i]\boldsymbol{x}[j]) \\ &= \sum_{i=1}^{U} \boldsymbol{x}[i]^2 + 0 = F_2(\boldsymbol{x})\end{aligned}$$

The proof relies critically on the properties of h: $h(i)^2 = 1$ for all i, but since h is 4-wise independent then the outcomes $h(i) = h(j)$ and $h(i) = -h(j)$ are equally likely (for $j \neq i$) and so in expectation $h(i)h(j)$ is zero.

Lemma 2. $\mathsf{Var}(Z^2) \le 2F_2(\boldsymbol{x})^2$

Proof.

$$
\begin{aligned}
\mathsf{Var}(Z^2) &= \mathsf{E}(Z^4) - \mathsf{E}(Z^2)^2 \\
&= \mathsf{E}((\sum_{i=1}^{U} h(i)\boldsymbol{x}[i])^4 - (\sum_{i=1}^{U} \boldsymbol{x}[i]^2)^2 \\
&= \mathsf{E}((\sum_{i=1}^{U} h(i)^4\boldsymbol{x}[i]^4 + \sum_{1\le i<j\le U} 6h(i)^2h(j)^2\boldsymbol{x}[i]^2\boldsymbol{x}[j]^2 \\
&\quad + \sum_{i,i\neq j\neq k} 12h(i)^2h(j)h(k)\boldsymbol{x}[i]^2\boldsymbol{x}[j]\boldsymbol{x}[k] + \sum_{1\le i\neq j\le U} 4h^3(i)h(j)\boldsymbol{x}[i]^3\boldsymbol{x}[j] \\
&\quad + \sum_{1\le i<j<k<l\le U} 12h(i)h(j)h(k)h(l)\boldsymbol{x}[i]\boldsymbol{x}[j]\boldsymbol{x}[k]\boldsymbol{x}[l]) \\
&\quad -(\sum_{i=1}^{U} \boldsymbol{x}[i]^4 + \sum_{1\le i<j\le U} 2\boldsymbol{x}[i]^2\boldsymbol{x}[j]^2) \\
&= \sum_{i=1}^{U} \boldsymbol{x}[i]^4 + \sum_{1\le i<j\le U} 6\boldsymbol{x}[i]^2\boldsymbol{x}[j]^2 - (\sum_{i=1}^{U} \boldsymbol{x}[i]^4 + \sum_{1\le i<j\le U} 2\boldsymbol{x}[i]^2\boldsymbol{x}[j]^2) \\
&= 4\sum_{1\le i<j\le U} \boldsymbol{x}[i]^2\boldsymbol{x}[j]^2) \le 2F_2^2
\end{aligned}
$$

This shows that each estimate is correct in expectation and has bounded variance. Again, in expectation many of the cross-terms (eg. $h(i)h(j)h(k)h(l)$) are zero, by the 4-wise independence of the hash function h. In order to give tight guarantees about the accuracy of this procedure, we make use of a few statistical results about the average and median of random variables.

Fact 1 (Variance Reduction) *Let X_i be independent and identically distributed random variables. Then*

$$\mathsf{Var}(\sum_{i=1}^{w} \frac{X_i}{w}) = \frac{1}{w}\mathsf{Var}(X_1)$$

In other words, taking the average of w copies of an estimator reduces the variance by a factor of w.

Fact 2 (The Chebyshev inequality) *Given a random variable X,*

$$\mathsf{Pr}[|X - \mathsf{E}(X)| \ge k] \le \frac{\mathsf{Var}(X)}{k^2}$$

Fact 3 (Application of Chernoff Bounds) *Let R be a range of values $R = [R_{min} \ldots R_{max}]$, and let Y_i be $d = 4\log 1/\delta$ independent and identically distributed random variable such that $\mathsf{Pr}[Y_i \notin R] \le \frac{1}{8}$. Then*

$$\mathsf{Pr}[(\mathrm{median}_{i=1}^{d} Y_i) \notin R] \le \delta$$

That is, if there is constant probability that each Y_i falls within the desired range R, then taking the median of $O(\log 1/\delta)$ copies of Y_i reduces the failure probability to δ.

For details of these facts, see a standard text such as [19]. For the final fact, observe that we can define an indicator variable for each Y_i that is 0 if Y_i falls within the range R and is 1 otherwise. The expectation of the sum of these indicator variables is $\frac{1}{2}\log 1/\delta$. However, if the median of the Y_is is not within range, then at least half the Y_is must have fallen outside the range; hence the sum of the indicator variables must be at least $2\log 1/\delta$. Applying Chernoff bounds gives the derived result.

We can now apply these facts to show the accuracy of the estimation procedure for F_2:

Theorem 1. *An (ϵ, δ)-approximation of F_2, the self-join size, can be computed in space $O(\frac{1}{\epsilon^2}\log 1/\delta)$ machine words in the streaming model. Each update takes time $O(\frac{1}{\epsilon^2}\log 1/\delta)$.*

Proof. Applying the Chebyshev inequality to the average of $w = \frac{16}{\epsilon^2}$ copies of the estimate Z generates a new estimate Y such that

$$\mathsf{Pr}[|Y - F_2| \leq \epsilon F_2] \leq \frac{\mathsf{Var}(Y)}{\epsilon^2 F_2^2} = \frac{\mathsf{Var}(Z)}{c\epsilon^2 F_2^2} = \frac{2F_2^2}{(16/\epsilon^2)\epsilon^2 F_2^2} = \frac{1}{8}$$

Hence, applying the Chernoff bound result from Fact 3 to the median of $4\log 1/\delta$ copies of the average Y gives the probability of the results being outside the range of ϵF_2 from F_2 as δ. The space required is that to maintain $O(\frac{1}{\epsilon^2}\log 1/\delta)$ copies of the original estimate. Each of these requires a counter and a 4-wise independent hash function, both of which can be represented with a constant number of machine words under the standard RAM model.

In summary, this shows that an (ϵ, δ)-approximation of F_2 can be computed using space that is essentially independent of the size of the stream or the dimensionality of the vector. The complete algorithm is given in Figure 1.

3.2 Vector Difference Estimation

The results for self-join size estimation can be applied to the problem of measuring the distance between two vectors. The result follows almost as an immediate corollary of the previous theorem, combined with the structure of the sketch. Observe that the difference between $\boldsymbol{x}$ and $\boldsymbol{y}$ as given in Definition 2 can be thought of as the self-join size of a single vector whose ith entry is $\boldsymbol{x}[i] - \boldsymbol{y}[i]$. The sketch of this vector is given by $\sum_i h(i)(\boldsymbol{x}[i] - \boldsymbol{y}[i])$. This can be rewritten as $\sum_i h(i)\boldsymbol{x}[i] - \sum_i h(i)\boldsymbol{y}[i]$. In other words, the sketch of the difference is the difference of the sketches. Thus, by subtracting the sketches and then applying the estimation procedure we can get an (ϵ, δ) approximation of the difference between the vectors.

This relies on the *linearity* of the sketching operation: any linear transformation (scaling, addition, subtraction etc.) to the original vector can be applied on the sketch and the result is the sketch of the modified vector. This was used implicitly to show that the sketch can be updated dynamically under streaming updates. Such linearity properties have also been used in a variety of techniques for streaming data based on sketches. See the discussion in Section 4 for some examples.

3.3 Join Size Estimation

Lemma 3. *Let Z_x be an entry of a sketch computed for the vector $\boldsymbol{x}$, and let Z_y be an entry of a sketch computer for $\boldsymbol{y}$ using the same hash function. The estimate is correct in expectation, i.e.* $\mathsf{E}(Z_x * Z_y) = \boldsymbol{x} \cdot \boldsymbol{y}$.

Proof.

$$\begin{aligned}\mathsf{E}(Z_x * Z_y) &= \mathsf{E}(\sum_{i=1}^{U} h(i)^2 \boldsymbol{x}[i]\boldsymbol{y}[i] + \sum_{1 \le i \ne j \le U} h(i)h(j)\boldsymbol{x}[i]\boldsymbol{y}[j]) \\ &= \sum_{i=1}^{U} \boldsymbol{x}[i]\boldsymbol{y}[i] + 0 = \boldsymbol{x} \cdot \boldsymbol{y}\end{aligned}$$

Lemma 4. $\mathsf{Var}(Z_x * Z_y) \le F_2(\boldsymbol{x})F_2(\boldsymbol{y})$

Proof.

$$\begin{aligned}\mathsf{Var}(Z_x * Z_y) &= \mathsf{E}(Z_x^2 Z_y^2) - \mathsf{E}(Z_x Z_y)^2 \\ &= \mathsf{E}\left(\sum_{i=1}^{U} h(i)^4 \boldsymbol{x}[i]^2 \boldsymbol{y}[i]^2 + \sum_{1 \le i \ne j \le U} h(i)^2 h(j)^2 \boldsymbol{x}[i]^2 \boldsymbol{y}[j]^2 \right. \\ &\quad \left. + \sum_{1 \le i < j \le U} 4h(i)^2 h(j)^2 \boldsymbol{x}[i]\boldsymbol{y}[i]\boldsymbol{x}[j]\boldsymbol{y}[j]\right) - (\boldsymbol{x} \cdot \boldsymbol{y})^2 \\ &= \sum_{i=1}^{U} (\boldsymbol{x}[i]\boldsymbol{y}[i])^2 + \sum_{1 \le i \ne j \le U} (\boldsymbol{x}[i]\boldsymbol{y}[j])^2 + \sum_{1 \le i < j \le U} 4\boldsymbol{x}[i]\boldsymbol{y}[i]\boldsymbol{x}[j]\boldsymbol{y}[j] \\ &\quad -(\sum_{1 \le i \le U} (\boldsymbol{x}[i]\boldsymbol{y}[i])^2 + \sum_{1 \le i < j \le U} (2\boldsymbol{x}[i]\boldsymbol{y}[i]\boldsymbol{x}[j]\boldsymbol{y}[j])) \\ &\le \sum_{1 \le i < j} (\boldsymbol{x}[i]\boldsymbol{y}[j])^2 + 2 \sum_{1 \le i < j \le U} (2\boldsymbol{x}[i]\boldsymbol{y}[i]\boldsymbol{x}[j]\boldsymbol{y}[j]) \\ &\le \sum_{i=1}^{U} \boldsymbol{x}[i]^2 \sum_{j=1}^{U} \boldsymbol{y}[j]^2 + (\sum_{i=1}^{U} \boldsymbol{x}[i]\boldsymbol{y}[i])^2 \\ &\le 2 \sum_{i=1}^{U} \boldsymbol{x}[i]^2 \sum_{j=1}^{U} \boldsymbol{y}[j]^2 = 2F_2(\boldsymbol{x})F_2(\boldsymbol{y})\end{aligned}$$

Applying the Chebyshev Inequality to the average of $w = \frac{16}{\epsilon^2}$ copies of this estimate, and then taking the median of $d = 4 \log 1/\delta$ such averages, as in the proof of Theorem 1 allows us to state the following theorem:

Theorem 2. *Using space* $O(\frac{1}{\epsilon^2} \log 1/\delta)$ *space we can output an estimate of* $\boldsymbol{x} \cdot \boldsymbol{y}$ *so that*

$$\mathsf{Pr}[|(\boldsymbol{x} \cdot \boldsymbol{y}) - est| \leq \epsilon \sqrt{F_2(\boldsymbol{x}) F_2(\boldsymbol{y})}] \geq 1 - \delta$$

Note that for the special case of when $\boldsymbol{x} = \boldsymbol{y}$, then the above Theorem 2 reduces to Theorem 1. However, this is not an (ϵ, δ)-approximation since in general $\sqrt{F_2(\boldsymbol{x}) F_2(\boldsymbol{y})} > (\boldsymbol{x} \cdot \boldsymbol{y})$. In order to get such an approximation, we need to increase w by a factor of $(\boldsymbol{x} \cdot \boldsymbol{y})^2/(F_2(\boldsymbol{x}) F_2(\boldsymbol{y}))$. This may be possible if we have *a priori* bounds on these quantities, but since we are trying to approximate $\boldsymbol{x} \cdot \boldsymbol{y}$, we cannot know this quantity exactly in advance. In general, we cannot hope for much stronger results due to the following negative result:

Theorem 3. *Guaranteeing an* (ϵ, δ) *approximation of* $(\boldsymbol{x} \cdot \boldsymbol{y})$ *requires* $\Omega(U)$ *space in the worst case.*

Proof. We reduce from the problem of testing whether two sets have any element in common, and use the communication complexity of this problem to argue a space bound for the streaming problem.

Consider two arbitrary sets X and Y, both of which are subsets of $[1 \ldots U]$. There are two people who wish to collaborate to compute a function of X and Y: X is held by one party and Y by the other. A well-known result from communication complexity states that determining whether there exists i such that $i \in X \wedge i \in Y$ requires communication between the two parties that is linear in U, even under a probabilistic model [18]. This is known as the disjointness problem, since the answer is either that the sets are disjoint (i.e. $X \cap Y = \emptyset$) or not disjoint.

First, we show that if we can approximate join size, then we can answer disjointness queries. Let $\boldsymbol{x}[i] = 1 \iff i \in X$, and zero otherwise; similarly, let $\boldsymbol{y}[j] = 1 \iff j \in Y$, otherwise $\boldsymbol{y}[j] = 0$. Now observe that $(X \cap Y = \emptyset) \iff (\boldsymbol{x} \cdot \boldsymbol{y} = 0)$. Hence, computing the join size exactly means that we can answer disjointness queries. More strongly, any approximation of $\boldsymbol{x} \cdot \boldsymbol{y}$ also allows us to answer disjointness queries, since to approximate $\boldsymbol{x} \cdot \boldsymbol{y} = 0$ we must output '0', and if $\boldsymbol{x} \cdot \boldsymbol{y} \neq 0$, then no approximation can correctly output '0'. Thus any algorithm that approximates $\boldsymbol{x} \cdot \boldsymbol{y}$ must use $\Omega(U)$ bits of communication if $\boldsymbol{x}$ is held by one party and $\boldsymbol{y}$ by the other.

Now, we show how this bound applies to the streaming context. Suppose all of $\boldsymbol{x}$ arrives in the stream first, and then $\boldsymbol{y}$ arrives in the stream next. Consider the state (memory contents) held by any streaming algorithm to approximate $\boldsymbol{x} \cdot \boldsymbol{y}$ after $\boldsymbol{x}$ has been seen. Imagine sending all the state to another copy of the algorithm which then receives the stream $\boldsymbol{y}$. If the algorithm correctly approximates $\boldsymbol{x} \cdot \boldsymbol{y}$, then the size of the data communicated must be $\Omega(U)$ bits. Hence, the space used by the algorithm must be at least $\Omega(U)$ bits.

Nevertheless, the results obtained by this estimation procedure in practice often give very good estimates of the join size between relations. In particular, it tends to significantly outperform solutions based on sampling, which do not give the guarantee of being correct in expectation.

4 Applications and Extensions

The generality of the join size aggregate and the simplicity and flexibility of the sketching technique, means that the sketching method has been used as the basis of a wide variety of other streaming algorithms. Rather than attempt to give a comprehensive survey of such techniques, we outline a few examples to illustrate the ways that this data structure has been applied and modified.

4.1 Point Estimation, Range Queries and Wavelets

The problem of point estimation is, given a vector $\boldsymbol{x}$ specified as a data stream, to accept queries which specify a particular entry, i, and to return an estimate of $\boldsymbol{x}[i]$. Clearly, one cannot give exact answers, or guarantee very fine accuracy since to do so would allow one to recover the whole vector in the worst case. However, they can be well approximated as a corollary of the previous theorem.

Corollary 1. *Point queries can be answered using the same sketch structure in space* $O(\frac{1}{\epsilon^2} \log 1/\delta)$ *with error less than* $\epsilon\sqrt{F_2(\boldsymbol{x})}$ *with probability at least* $1 - \delta$.

Proof. Observe that a point query can be specified as a join size query, $\boldsymbol{x} \cdot I_i$, where $I_i[i] = 1$, and $I_i[j] = 0$ for $j \neq i$. Applying Theorem 2, we find that we can answer point-estimation queries with error at most $\epsilon\sqrt{F_2(\boldsymbol{x})}$ with probability at least $1 - \delta$.

This is quite a strong guarantee, since typically we make require accuracy in terms of $\epsilon F_1(\boldsymbol{x})$, and for any $\boldsymbol{x}$, $\sqrt{F_2(\boldsymbol{x})} \leq F_1(\boldsymbol{x})$. Point Estimation is at the heart of many algorithms for finding "heavy hitters": items in the data stream which occur very frequently. (This is explored further in a chapter by Charikar later in this volume.)

In a similar way, one can answer arbitrary *range queries* of the form $R(i, j) = \sum_{k=i}^{j} \boldsymbol{x}[k]$ by reducing this to an inner product with a vector I_i^j, where $I_i^j[k] = 1 \iff i \leq k \leq j$ and zero elsewhere. However, the error increases linearly with $|j - i + 1|$, making the accuracy too weak for large ranges. A standard approach is then to decompose any arbitrary range into at most $O(\log U)$ *dyadic ranges*, each of which has length a power of two, and begins at a multiple of its own length. Each dyadic range query can be treated as a point query, and hence arbitrary range queries can be answered to the same

accuracy as point queries, with a blow-up in space polynomial in $\log U$ (see, for example, [13]).

Computing approximate (Haar) wavelet co-efficients and approximate hostogram summaries [9] can also make use of point, range and related queries, and hence techniques to approximate the wavelet coefficients of a signal presented in a streaming fashion make extensive use of sketch data structures. (More details of the methods used and the results obtained are given in a chapter by Muthukrishnan and Strauss later in this volume.)

4.2 Faster Implementations Using Hashing

One potential disadvantage of the sketching scheme is the time cost to process each update. For $w = O(\frac{1}{\epsilon^2})$ and $d = O(\log 1/\delta)$ copies of the estimate, we must update wd entries in the sketch for every update. For small values of ϵ this can be a large overhead, and means that such an approach does not easily scale to very high speed data stream environment without special purpose hardware. However, a simple hashing trick has been applied to increase the speed significantly [6, 8, 7, 22]. Instead of keeping w copies of the estimate and taking the average of their estimates, the key idea is to use a second hash function f which maps each item i onto $[1 \ldots w]$, and only updating the estimate $f(i)$. To produce an estimate of the join or self-join size, the sum of the estimates is used instead of the average. Mathematically this gives the same expectation and variance as the original method, but requires only $O(d)$ estimates to be modified for each update instead of $O(wd)$. This means that the cost is essentially independent of the accuracy parameter ϵ, and depends only on $\log 1/\delta$, which is typically quite small in practice.

4.3 Multidimensional and Spatial Data

We have so far phrased the discussion in terms of join sizes and large vectors. However, one canb model other structures with sketches. By appropriate linearization, one can make a sketch of a large matrix of values, and similar higher dimensional structures. Such modeling is necessary in order to estimate the size of multi-way joins (see the chapter of Dobra et al. in this volume).

More generally, one can also apply the sketching technique to summarize spatial data. Now, the input consists of (a stream of) objects in low dimensional space (say, two or three dimensions). The notion of a spatial join is to count the number of object pairs that fall within a certain (specified as part of the query) distance of each other. Other natural queries are to ask for the approximate number of points falling within a given range, eg, a rectangular or cuboid region.

Sketch data structures have been applied to these problems [10]. The key technique is to build sketches from the query specifications in such a way that the approximate join size between the query sketch and data sketch is an unbiased estimator for the answer to the query. For example, suppose the

input consists of a set of one-dimensional ranges $[a \dots b]$. To answer the query of how many intervals contain a given point c, we can simple pose the point query I_c. To count how many ranges overlap with a query range, we can compute the sum of how many of the ranges contain each end point of the query range, and how many of the end points of the ranges are contained within the query range. Assuming that the query end points do not coincide with points of the input[4], this query returns exactly twice the number of intersections, and so can be approximated correctly in expectation.

In order to bound the error from using sketches, and ensure that updates are fast to process, ranges are not represented directly, but using the "dyadic range decomposition" described in Section 4.1. This approach can be generalized from one dimensional data to points and rectangles in the plane and in three dimensional space, etc. This shows some nice features of the sketching approach: it is sufficiently general that it can be applied to a variety of different streaming scenarios. Using techniques such as the dyadic range decomposition, these ideas can be implemented effiicently and with bounded error per update.

4.4 Further Extensions and Applications

We outline some of the many other applications in data streams that sketch-based techniques have found:

- **Vector L_1 Difference.** For some applications, rather than the L_2 distance between two vectors, $\sqrt{F_2(\boldsymbol{x} - \boldsymbol{y})}$, it is necessary to compute the L_1 difference, $\sum_{i=1}^{U} |\boldsymbol{x}[i] - \boldsymbol{y}[i]|$. This can be reduced to F_2 by representing the vectors in unary notation, since $L_1(\boldsymbol{x} - \boldsymbol{y}) = F_2(\boldsymbol{x} - \boldsymbol{y})$ if $\boldsymbol{x}$ and $\boldsymbol{y}$ are binary (zero/one) vectors. However, in order to compute this transformation efficiently, new methods are needed to quickly compute the sum of 4-wise hash functions, rather than explicitly creating the unary representation of large vector entries [11].
- **Triangle Counting in Graphs.** Many data streams represent graphs, presented as streams of edges, and the streaming challenge is to compute properties of the induced graphs. The number of triangles, also known as the clustering coefficient, occurs in a variety of applications, but seems challenging to compute when the edges forming each triangle can be arbitrarily interleaved in the stream. However, by a careful transformation, the number of triangles can be expressed as a function of appropriately defined frequency moments F_0, F_1 and F_2 [4]. New techniques are required to efficiently update the sketches as each edge requires a large number of updates to be applied, but these updates can be described concisely as a range of values.

[4] This assumption can be removed with some careful manipulation: see [10]

- **Change Detection.** In many large scale monitoring applications, the fundamental question is whether the current observations are in line with predicted behavior, or whether they appear to be at odds with what was expected. Such applications are generally known as "change detection". A general approach was suggested in [17]: build sketches of recent data, and then apply various standard modelling techniques to combine these into a sketch of the predicted behavior. The new data can then be observed, and tested against the prediction: either in terms of individual item counts or by comparing the whole stream. This approach relies crucially on the linearity properties of the sketch transformation, so the predicted sketch can be obtained by applying the prediction model to the historical sketches. Thus, the whole method can be carried out in small space and at high speeds in the streaming model.
- **Quantiles under insertions and deletions.** The problem of tracking quantiles over a stream of input items drawn from $[1 \dots U]$ is, given ϕ, return an item whose rank is (approximately) ϕN. Various techniques have been proposed for this problem when the stream consists of insertions of items only. However, when the input stream may also contain deletions of items that have previously appeared, these techniques do not apply. Observe that the quantiles query can be restated as a range query: if we can estimate how many items fall in the range $[1 \dots R]$, then we can binary search for the value of R whose range contains ϕN points. This range query can be answered using the techniques of Section 4.1. The reduction to dyadic ranges makes updates and queries reasonably fast, and the error is bounded by $\epsilon \log U \sqrt{F_2(\boldsymbol{x})}$. Because deletions can be processed as negative updates to sketches, it is easy to see that deletion operations are handled correctly. Full details of this approach to finding quantiles using sketches are in [13, 7].
- **Tracking Queries over Distributed Streams.** Large-scale stream processing applications rely on continuous, event-driven monitoring, and are often inherently *distributed*, with several remote monitor sites observing their local, high-speed data streams and exchanging information through a communication network. This distribution of the data naturally implies critical *communication constraints* that typically prohibit continuously centralizing all the streaming updates, due to volume and speed of the streams that can easily overwhelm the underlying network. Monitoring queries over such *distributed streams* raises a host of new challenges. A property of AMS sketches that makes them particularly interesting in this setting is that, due to their linear nature, they are naturally *composable* through simple vector addition. In other words, given two "parallel" AMS sketches (built using the same 4-wise hash functions) over two different streams, the sketch of the combined stream (i.e., the union of the two streams) is simply the component-wise summation of their sketches. More details on the distributed streaming model and results can be found in a chapter by Garofalakis later in this volume.

5 Concluding Remarks

The original paper describing the sketching technique discussed here was published in 1996 [3], and showed the F_2 application. A subsequent paper in 1999 [2] extended the results to join and self-join size. The original "AMS" paper considers a broad range of problems based on the frequency moments, and has come to be viewed as one of the foundational works on data streams, even though this term is not explicitly used by the authors. In addition to the results on F_2, the authors also give space efficient algorithms for all frequency moments F_k, $k \in \mathcal{N}$, and lower bounds for the problems showing that, for $k \geq 6$, space polynomial in n is required. This led to a sequence of papers in the theoretical computer science computer science which has focussed on improving the upper and lower bounds for the frequency moments problem for $k \geq 3$, culmininating in recent results showing essentially tight upper and lower bounds for these problems.

The result can also be thought of in terms of embedding vectors into lower dimensional spaces. The Johnson-Lindenstrauss lemma [16] proved that their exists embeddings of vectors in Euclidean space into a Euclidean space of (smaller) dimension $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ which preserves distances up to a $(1 \pm \epsilon)$ factor. We can view the sketch technique here as an explicit embedding into a Euclidean-like space (the operations of averaging and median finding mean that we cannot treat the sketches as vectors in Euclidean space), which is computable in a data stream setting with small space and limited (four-wise) randomness. The results of Achlioptas [1] show that, if we assume full randomness, then we can also operate in Euclidean space.

Lastly, several efficient implementations of sketch data structures have been made and published on the Internet (e.g. `http://www.cs.rutgers.edu/~muthu/massdall-code-index.html`). These can be freely modified and used as the basis of more complex data stream algorithms.

References

1. D. Achlioptas. Database-friendly random projections. In *Proceedings of ACM Principles of Database Systems*, pages 274–281, 2001.
2. N. Alon, P. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of the Eighteenth ACM Symposium on Principles of Database Systems*, pages 10–20, 1999.
3. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 20–29, 1996. Journal version in *Journal of Computer and System Sciences*, 58:137–147, 1999.
4. Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 623–632, 2002.

5. J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.
6. M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Procedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 693–703, 2002.
7. G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In *Latin American Informatics*, pages 29–38, 2004.
8. Graham Cormode and Minos Garofalakis. "Approximate Continuous Querying over Distributed Streams". *ACM Transactions on Database Systems*, 33(2), June 2008.
9. Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. "Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches". *Foundations and Trends in Databases*, 4(1-3), 2012.
10. Abhinandan Das, Johannes Gehrke, and Mirek Riedewald. "Approximation Techniques for Spatial Data". In *Proc. of the 2004 ACM SIGMOD Intl. Conference on Management of Data*, June 2004.
11. J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L_1-difference algorithm for massive data streams. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 501–511, 1999.
12. Minos Garofalakis and Phillip B. Gibbons. "Approximate Query Processing: Taming the Terabytes". Tutorial in *27th Intl. Conf. on Very Large Data Bases*, Roma, Italy, September 2001.
13. A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *Proceedings of the International Conference on Very Large Data Bases*, pages 454–465, 2002.
14. I. J. Good. Surprise indexes and p-values. *Journal of Statistical Computer Simulations*, 32:90–92, 1989.
15. Yannis E. Ioannidis and Stavros Christodoulakis. "Optimal Histograms for Limiting Worst-Case Error Propagation in the Size of Join Results". *ACM Transactions on Database Systems*, 18(4):709–748, December 1993.
16. W.B. Johnson and J. Lindenstrauss. Extensions of Lipshitz mapping into Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.
17. B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation and applications. In *Proceedings of the ACM SIGCOMM conference on Internet measurement*, pages 234–247, 2003.
18. E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.
19. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
20. S. Muthukrishnan. Data streams: Algorithms and applications. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2003.
21. M. Thorup. Even strongly universal hashing is pretty fast. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 496–497, 2000.
22. M. Thorup and Y. Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 615–624, 2004.

23. Mark N. Wegman and J. Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.