Comparing Data Streams Using Hamming Norms (How to Zero In)

Graham Cormode
csrcp@warwick.ac.uk

Mayur Datar
datar@cs.stanford.edu

Piotr Indyk
indyk@lcs.mit.edu

S. Muthukrishnan
muthu@research.att.com

Abstract

Massive data streams are now fundamental to many data processing applications. For example, Internet routers produce large scale diagnostic data streams. Such streams are rarely stored in traditional databases, and instead must be processed "on the fly" as they are produced. Similarly, sensor networks produce multiple data streams of observations from their sensors. There is growing focus on manipulating data streams, and hence, there is a need to identify basic operations of interest in managing data streams, and to support them efficiently.

We propose computation of the Hamming norm as a basic operation of interest. The Hamming norm formalises ideas that are used throughout data processing. When applied to a single stream, the Hamming norm gives the number of distinct items that are present in that data stream, which is a statistic of great interest in databases. When applied to a pair of streams, the Hamming norm gives an important measure of (dis)similarity: the number of unequal item counts in the two streams. Hamming norms have many uses in comparing data streams.

We present a novel approximation technique for estimating the Hamming norm for massive data streams; this relies on what we call the "l_0 *sketch*" and we prove its accuracy. We test our approximation method on a large quantity of synthetic and real stream data, and show that the estimation is accurate to within a few percentage points.

**Proceedings of the 28th VLDB Conference,
Hong Kong, China, 2002**

1 Introduction

Data streams are now fundamental to many data processing applications. For example, telecommunication network elements such as switches and routers periodically generate records of their traffic — telephone calls, Internet packet and flow traces — which are data streams [3, 22, 33]. Atmospheric observations — weather measurements, lightning stroke data and satellite imagery — also produce multiple data streams [34, 39]. Emerging sensor networks produce many streams of observations, for example highway traffic conditions [31, 37]. Sources of data streams — large scale transactions, web clicks, ticker tape updates of stock quotes, toll booth observations — are ubiquitous in daily life.

For many applications that produce data streams, it is useful to visualize the underlying data seen so far or underlying state of the system as a very high dimensional vector (note that in most cases the vector is *not* explicitly represented or materialized). For example, if we consider a data stream created by network flows that originate from a source IP address, the state at any time t can be visualized as a vector $\boldsymbol{a} = a_1 \ldots a_n$ that is indexed by the destination IP address. The entry along each dimension (a_i) represents the total number of flows that were observed between the given source IP address and the destination IP address a_i. The input is usually presented in the order it arrives (rather than sorted on any attribute), and consists of updates only. For instance, in the example above we may not receive data sorted on the destination IP address and each data element would represent an additional flow along an arbitrary destination IP address. Formally, each update to $\boldsymbol{a}$ is represented by a pair (i, d_k), which is interpreted as "add the value d_k to the i-th coordinate". Thus at any time t, the value of a_i is the sum of d_k's that were added to the ith coordinate. The sequence of updates we see on the stream therefore implicitly represents $\boldsymbol{a}$. Thus we call $\boldsymbol{a}$ the (implicit) state vector of the data stream.

Data stream processing entails a special constraint. Despite the exponential growth in the capacity of storage de-

vices, it not common for such streams to be stored. Nor is it desirable or helpful to store them, since the cost of any simple processing — even just sorting the data — would be too great. The main challenge in handling data streams is to perform necessary computations "on the fly" using a small amount of storage, while maintaining low total running time.

Since there is growing focus on manipulating data streams, the database and data processing infrastructure needed to handle stream data is now being investigated. Also, there is a need to identify basic operations of interest in managing data streams, and to support them efficiently. The database community has just begun to investigate the challenges involved [2, 4, 23] complementing the efforts emerging in the other communities — algorithms [12, 14, 25, 26, 28], networking [3], physical sciences [39], and elsewhere.

In this paper, we propose *Hamming norm computation* as a basic operation of interest for data stream processing. Consider a data stream with state vector $\boldsymbol{a}$. The *Hamming norm* of vector $\boldsymbol{a}$, written $|\boldsymbol{a}|_H$, is the number of values i for which $a_i \neq 0$. That is, $|\boldsymbol{a}|_H = |\{i|a_i \neq 0\}|$.

There are two compelling reasons for proposing Hamming norms. First, in the special case where $\boldsymbol{a}$ represents a vector of counts, similar to our earlier example, $|\boldsymbol{a}|_H$ is the number of distinct items in the data stream; computing $|\boldsymbol{a}|_H$ on a stream is equivalent to maintaining the number of distinct values in a database relation in the presence of inserts and deletes to it. As such, this is an important problem in traditional databases. Second, when we apply Hamming norm to two (or more) data streams, we get very interesting measures. For two streams that represent state vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ respectively, we may consider the Hamming norm of the sum of the vectors or their difference. The Hamming norm of the sum $|\boldsymbol{a}+\boldsymbol{b}|_H = |\{i|(a_i+b_i) \neq 0\}$ represents the union of the two streams. The Hamming norm of the difference $|\boldsymbol{a}-\boldsymbol{b}|_H = |\{i|a_i \neq b_i\}| = |\{i|(a_i-b_i) \neq 0\}$ is the number of dimensions in which they differ. Both these parameters are of fundamental interest. We will see a few of the large number of uses for the Hamming norm in more detail in Section 2.

Our contributions are as follows.

1. We initiate the study of Hamming norm computations for data streams.

2. We present a novel algorithm for calculating a very small summary for any data stream (what we call the l_0 *sketch*) such that the Hamming norm of that stream can be found up to a user-specified approximation factor (with high probability) using only the l_0 sketches. This is the first known algorithm for the problem of Hamming norm computation. For the special case of estimating the number of distinct items, algorithms are known to exist and we demonstrate that our algorithm can outperform them. Our algorithm has following properties.

   - The approximation factor is a priori guaranteed to be $1 \pm \epsilon$, and the sketch requires only space $O(1/\epsilon^2)$. Note that this is of constant size for a fixed fraction ϵ and is independent of the size of the data stream of the signal.

   - The l_0 sketches can be maintained efficiently in presence of a stream consisting of dynamic updates. We can estimate Hamming norms without requiring to rescan the entire relation even under an arbitrary mixture of additions and deletions.

   - The l_0 sketches can be computed separately and then combined in a number of ways: they can be added or subtracted to find the union or difference of the corresponding streams. Hamming norm information can be computed in time proportional to the size of the l_0 sketch, which is effectively constant. This procedure is therefore fast, much faster than sampling.

   - The l_0 sketch is an embedding of the Hamming norm into a very small number of dimensions. As such l_0 sketches can be used to answer nearest neighbors and other proximity and similarity queries amongst data streams for Hamming norms.

3. Our other contribution is to perform a thorough set of experiments with both synthetic data and real NetFlow data drawn from a large ISP, demonstrating the power of the l_0 sketches on computing various Hamming norms. We show experiments where we estimate Hamming norms accurately, correct to within a few percentage points of the correct answer. For this we use a working space of only 8Kb and handle datasets of tens of megabytes, and we can fix this working space while scaling up to gigabytes of data and larger size with the same accuracy guarantees. For finding the Hamming norm of a single stream, which corresponds to the maintenance of the number of distinct elements under insertions and deletions to the databases, our solution is more accurate than existing methods. For multiple general data streams where both insertions and deletions are allowed to occur arbitrarily within both streams, we present the first known solutions for union and difference problems.

The approach of l_0 sketches to approximate Hamming norms allows us to zero-in to estimate distinct values and differences in massive data streams using a very small summary structure. We motivate Hamming norms in more detail in Section 2 and discuss previous work in Section 3. We present preliminaries in Section 4 and our solution in Section 5. The results of experimental evaluations are shown in Section 6 and conclusions given in Section 7.

2 Motivating Hamming Norms of Data Streams

We will motivate Hamming norms in more detail by considering two applications.

2.1 Maintaining distinct values in traditional databases

The Hamming norm of a stream[1] is of large interest in itself. It follows from the definition above that this quantity is precisely the number of distinct items in the stream. For example, let $\boldsymbol{a}$ be a stream representing any attribute of a given database, so a_i is the number of tuples in the database with value i in the attribute of interest. Computing the Hamming norm of $\boldsymbol{a}$ provides the number of distinct values of that attribute taken by the tuples. This is a foundational problem. We consider a traditional database table which is subject to a sequence of insertions and deletions of rows. It is of importance to query optimization and otherwise to know the number of distinct values that each attribute of the table assumes. The importance of this problem is highlighted in [6]: "A principled choice of an execution plan by an optimizer heavily depends on the availability of statistical summaries like histograms and the number distinct values in a column for the tables referenced in the query." Distinct values are also of importance in statistics and scientific computing (see [19, 20, 27]). Unfortunately, it is provably impossible to approximate this statistic without looking at a large fraction of the database (eg via sampling) [6]. Our algorithm avoids this problem by *maintaining* the desired statistics under database updates, so that we never have to *compute* it from scratch.

2.2 Monitoring and Auditing Network Databases

Network managers view information from multiple data stream sources. Routers periodically send traffic information: traces of IP packets and IP flows (which are aggregated IP packet flows) [33]; there are management routine updates: SNMP traps, card/interface/link status updates, route reachability via pings and other alarms [24]; configuration information: topology and various routing tables [3]. Network managers need ways to take this continuous stream of diagnostic information and extract meaningful information. The infrastructure for collecting this information is often error-prone because of unreliable transfer (typically UDP and not TCP is used for data collection); network elements fail (links go down); configuration tables have errors; and data is incomplete (not all network elements might be configured to provide diagnostic data).

Continuous monitoring tools are needed to audit different data sources to ensure their integrity. This calls for "slicing and dicing" different data streams and corroborating them with alternate data sources. We give three examples of how this can be accomplished by computing the Hamming distance between data streams.

1. Let a_i be the number of *transit* IP packets sent from IP address i that enter a part of the network, and b_i be the number of IP packets that exit that part from i. We would like to determine the Hamming Distance between these counts to find out how many transit flows are losing packets within this part of the network.

2. There are published methods for constructing flows from IP packet traces. We can take traces of IP packets, aggregate them, and generate a flow log from this. This can then be compared with the flows generated by routers [10, 33], to spot discrepancies in the network.

3. Denial of Service attacks involve flooding a network with a large number of requests from spoofed IP addresses. Since these addresses are faked, responses are not acknowledged. So the Hamming difference between a vector of addresses which issued requests and those which sent acknowledgements will be high in this situation [32]. The Hamming norm of the difference between these two vectors provides a quick check for the presence of sustained Denial of Service attacks and other network abnormality, and could be incorporated into network monitoring toolkits.

There are many other potential applications of Hamming norm computation such as in database auditing and data cleaning. Data cleaning requires finding columns that are mostly similar [11]; the Hamming norm of columns in a table can quickly identify such candidates, even if the rows are arranged in different orders. It is beyond the scope of this paper to go into detail on all these applications, so we do not elaborate further on them.

3 Prior Work

3.1 Work on Data Streams and Sketches

Previous work on data streams has addressed problems of finding approximately the l_2 (Euclidean) norm and the l_1 norm of massive vectors whose entries are listed in an arbitrary order [1, 14, 28]. We study a related problem, of finding the Hamming norm of a vector, and the Hamming distance between pairs of vectors. No previous results were known for these problems, in their general form as stated here. As mentioned in the introduction, our algorithms are based on the technique called *sketching*. The basic idea is to represent the whole dataset using only very small amount of space, while preserving important information. When combined with data streams, then these sketches must be produced online as the data arrives.

The sketching technique has its roots in the field of mathematics called functional analysis [30]. We consider

[1] To simplify the exposition, here and in the remainder of this paper we will write "a norm of a stream" instead of "a norm of the implicit state vector of a stream".

in particular the application of sketching to approximate l_p norms of the data. The l_p norm of a vector $\boldsymbol{a}$ is equal to

$$||\boldsymbol{a}||_p = (\sum_i |a_i|^p)^{\frac{1}{p}}$$

Sketching was first applied to tracking approximate size of a self-join of a relation in [1]; in our language, this corresponds to maintaining the l_2 norm of the vector represented by a stream. The technique of Alon, Matias and Szegedy [1] was later generalized by Indyk [28] to maintain the l_p norm of the stream vector for any $p \in (0, 2]$. In the context of databases, the group of techniques covered by the umbrella term "sketching" have been applied to finding representative trends in massive one and multidimensional time series data [9, 29]. They have also been applied to multidimensional histograms [38] and data cleaning [11]. But our concept of l_0 sketch is a novel approach to estimating the Hamming norm and l_0 sketches have not been used previously in the literature.

Besides comparing multiple streams there has been work on the problem of one pass clustering of data streams [26]. There has been a lot of work in computing over data streams for purposes such as set resemblance, data mining, creating histograms and so on [8, 13, 25]. Particularly relevant is some recent work [19, 21], which study the problem of finding the size of the union of two streams. Here, the streams define multisets of elements, and it is the size of the union of the supporting sets that is of interest. Their method is only applicable to streams which consist solely of inserts — they fail when deletions are allowed. The method we present solves this problem when both insertions and deletions are allowed to occur arbitrarily within both streams.

3.2 Maintaining Distinct Elements Estimates

There have been two main styles of approach to counting distinct elements: these are sampling based, and synopsis based. Sampling attempts to do a small amount of probing of a complete list of the items in an attempt to find how many distinct values there are [5, 6, 7, 27]. However sampling based approaches are known to be inaccurate and substantial lower bounds on the sample size required have been shown [6], proving that a large fraction of the data must be sampled. The alternative paradigm consists of synopsis based approaches which keep a small summary of the stream, and update this every time an item is added or removed. We focus on these synopsis methods, since they can work in our data streams model, whereas sampling is not suited to dynamic modification of the data. The most widely applicable synopsis method is that of Flajolet and Martin [17, 18], which we describe in outline to enable comparison with our algorithm.

The algorithm is shown in Figure 1. The crucial part is the set of m hash functions $hash_j$, which map item values onto the range $[1 \ldots \log n]$. $hash_j$ is designed so that the probability $\Pr[hash_j(i) = \ell] = 2^{-\ell}$. Intuitively this

```
initialise c[1,1]...c[m,log n] = 0
for all tuples (i, dk) do
  for j = 1 to m do
    c[j, hash_j(i)] = c[j, hash_j(i)] + dk
for j = 1 to m do
  for k = log n downto 1 do
    if c[j, k] = 0 then
      minzero = k
  total = total + minzero
return(1.2928 × 2^(total/m))
```

Figure 1: The Flajolet-Martin algorithm for computing the Hamming norm of a stream

procedure works because if the probability that any item is mapped onto counter ℓ is $2^{-\ell}$, then if there are d non-zero entries in the vector, then we expect $d/2$ to be mapped to the first entry, $d/4$ to be mapped to the second, and so on until there are none expected in the $(\log_2 d)$th. Several repetitions of this procedure are done independently, and the result scaled by an appropriate factor (1.2928).

Theorem 3.1 *Theorem 2 of [17].*
This procedure gives an unbiased estimate of the number of distinct values that are seen in a stream.

This solves the problem of finding the number of distinct values in a stream of values, which as we know it is the Hamming norm of that stream. However, this method fails to find the Hamming norm of general vectors since it relies on the input conforming to certain conditions: the result is not defined if the implicit vector $\boldsymbol{a}$ has any entries that are negative. This can lead to decreasing a counter below zero, or to producing an estimate of the number of distinct elements that is highly inaccurate. In particular then, this method cannot be used to find the Hamming norm of the difference of two streams, $|\boldsymbol{a} - \boldsymbol{b}|_H$.

3.3 Database work on Network Monitoring

Database issues in network monitoring are beginning to get explored. Specific data processing problems in networking databases have been studied such as constructing traffic matrices [16]. The database architecture necessary for processing multiple configuration and data files arising in networks has been discussed in [3, 15]. In the specific case of sensor network data streams, a system architecture has been presented in [31]. At least two different philosophies seem to exist for dealing with network traffic data streams: one is to appropriately sample to decrease them to manageable size and to collate them in massive data warehouses [15], and the other is to deploy a database infrastructure where querying and summarization can be pushed to network elements and processing is distributed (for example [10, 22]). However, we are not aware of any prior work using the Hamming norm in this context.

4 Preliminaries

4.1 Data Stream Model

We assume a very general, abstracted model of data streams where our input arrives as a stream of data values, (i, d_k). This indicates that we should add the integer d_k to the count for item i. Clearly, we can accommodate subtractions by allowing d_k to be negative. Update operations have the effect that each tuple (i, d_k) causes $a_i \leftarrow a_i + d_k$. The accumulation of all these pieces of information defines the implicit vector, $\boldsymbol{a}$ such that $a_i = l$ means that over all tuples for item i the total of the d_k's is l.

An important factor is any restrictions on how the information in the stream arrives. For the most part, we expect the data to arrive in no particular order, since it is unrealistic to expect it to be sorted. Another question is whether every attribute value will be seen at most once, or whether there can be multiple such data items spread out arbitrarily within the stream. Here, we assume the most general case, that the data arrives unordered and the same value can appear multiple times within the stream. This is termed the *unordered, unaggregated model* (cash register) in [23].

The processing of massive data streams requires the use of a more restricted model of computation. In this model, we must process a stream of data, with the demand that each item in the stream must be processed completely and then discarded before the next is received. It is not possible to backtrack on the stream, so once a data item has been seen it cannot be retrieved unless it is explicitly stored in the working space. For a method in this model to be useful in practice, it must therefore use an amount of working space much smaller than the total size of the data, and also process each item rapidly. There has been a great deal of interest in processing data streams recently, see for example [4, 31].

Example. We consider the following stream of IP flows as <source,dest> pairs: <10.0.0.59,10.0.0.21>, <10.0.0.105,10.0.0.109>, <10.0.0.59,10.0.0.21>, <10.0.0.105,10.0.0.17>, <10.0.0.59,10.0.0.105>, <10.0.0.252,10.0.0.253>

In this example, we see that the same source address appears many times throughout the stream, and the same pair can appear more than once. In a realistic setting, IP address pairs can come in arbitrary order (and are drawn from a space of $(2^{32})^2$ possible pairs).

4.2 Stable Distributions

A vital part of our solution is the use of what are known as stable distributions. Various statistical distributions are said to be *stable*, with a stability parameter p. Distributions with stability parameter p have the following property: If random variables $X_1, X_2 \ldots X_l$ have stable distributions with stability parameter p, then $a_1X_1 + a_2X_2 \ldots a_lX_l$ is distributed as $(\sum_i |a_i|^p)^{1/p} X_0$, where X_0 is also a random variable with p stable distribution. This property will let us use the stable distributions to compute l_p norms. For example, the Gaussian distribution is stable with stability parameter 2, and the Cauchy distribution is stable with parameter 1. See for example [35] for more details of stable distributions.

5 Our Hamming Norm Computation

We first show the algorithm for computing a sketch to approximate the Hamming norm of a single stream. We then show how this can be easily extended to compute norms of combinations (union and differences) of streams.

5.1 The Hamming Norm of a Stream

We can now state our main theorem about computing this norm.

Theorem 5.1 *We can compute a* sketch, $sk(\boldsymbol{a})$ *of a stream* $\boldsymbol{a}$ *that requires space* $O(1/\epsilon^2 \cdot \log 1/\delta)$. *This allows approximation of the Hamming norm within a factor of* $1 \pm \epsilon$ *of the true answer with probability* $1 - \delta$. *Processing each new item and computing the Hamming norm of* $\boldsymbol{a}$ *both take time linear in the size of the sketch, i.e.* $O(1)$ *for fixed* ϵ *and* δ.

The proof of this claim requires several steps. We first show how l_p norms can be used to approximate Hamming norms, and then show how this can be implemented within the space bounds.

Reduction to l_p norms

Theorem 5.2 *The Hamming norm* $|\boldsymbol{a}|_H$ *can be approximated by finding the* l_p *norm of the vector* $\boldsymbol{a}$ *for sufficiently small* p $(0 < p \leq \epsilon / \log U)$ *provided we have an upper bound* (U) *on the size of each entry in the vector, so* $\forall i. |a_i| < U$.

Proof. We provide another mathematical definition for the Hamming norm which is crucial for our algorithms. We want to find $|\{i|a_i \neq 0\}|$. Observe that $|a_i|^0 = 1$ if $a_i \neq 0$; we can define $|a_i|^0 = 0$ for $a_i = 0$. Thus, the Hamming norm of a vector $\boldsymbol{a}$ is given by $\sum_i |a_i|^0$. This is similar to the definition of the l_p norm of a vector given in Section 3.1. We must define $l_0 = \sum_i |a_i|^0 = |\boldsymbol{a}|_H$. Hence the l_0 norm as defined here is our Hamming norm, and we refer to our sketches as l_0 sketches.

We show that the l_0 norm of a vector can be well-approximated by $(l_p)^p$ if we take $p > 0$ small enough. We consider $\sum_i |a_i|^p = (l_p)^p$ for a small value of p $(p > 0)$. If, for all i we have that $|a_i| \leq U$ for some upper bound U, then

$$|\boldsymbol{a}|_H = \sum_i |a_i|^0 \leq \sum_i |a_i|^p \leq \sum_i U^p |a_i|^0$$

$$\leq U^p \sum_i |a_i|^0 \leq (1+\epsilon) \sum_i |a_i|^0 = (1+\epsilon)|\boldsymbol{a}|_H$$

We use the fact that a_i is an integer and $\forall i |a_i| \leq U$. The last inequality follows from $U^p \leq (1+\epsilon)$ if we set $p \leq \log(1+\epsilon)/\log U \approx \epsilon/\log(U)$. ∎

Creating the l_0 sketch We define an l_0 *sketch* vector $sk(\boldsymbol{a})$ with dimension m (m will be specified shortly) as follows: $sk(\boldsymbol{a})$ is the dot product $\boldsymbol{x^T} \cdot \boldsymbol{a}$ (where $\boldsymbol{x}$ is a matrix of values $x_{i,j}$), so

$$sk(\boldsymbol{a})_j = \sum_{i=1}^{n} x_{i,j} a_i$$

Each $x_{i,j}$ is drawn independently from a random stable distribution with parameter p, with p as small as possible. Here $1 \leq i \leq n$, and $1 \leq j \leq m$; n is the dimension of the underlying vector $\boldsymbol{a}$, and m is the dimension of the sketch vector. According to Section 4.2, each entry of $sk(a)$ is distributed as $(\sum_i |a_i|^p)^{1/p} X$, where X is random variable chosen from a p-stable distribution. We will use $sk(a)$ as our approximation of the l_0 norm. In particular, we use this sketch to find $\sum_i |a_i|^p$ for $0 < p \leq \epsilon/\log U$, from which we can approximate the Hamming norm up to a $(1+\epsilon)$ factor. By construction, we can use any $sk(\boldsymbol{a})_j$ to estimate the $(l_p)^p = \sum_i |a_i|^p$. We combine these values to get a good estimator for the $(l_p)^p$ by taking the median of all entries $|sk(\boldsymbol{a})_j|^p$.

Lemma 5.1

$$(1-\epsilon)^p \operatorname{median}_j |sk(\boldsymbol{a})_j|^p \leq \operatorname{median} |X_0|^p (\textstyle\sum_i |a_i|^p) \leq (1+\epsilon)^p \operatorname{median}_j |sk(\boldsymbol{a})_j|^p$$

with probability $1-\delta$ if $m = O(1/\epsilon^2 \cdot \log 1/\delta)$, where X_0 is a random variable with p-stable distribution.

The proof of this theorem follows from the results in [28]. Theorems 5.2 and Lemma 5.1 tell us that this technique (for a small enough value of p) will estimate the Hamming norm with guaranteed quality and fidelity. ∎

Practical aspects for streaming data We now list the additional modifications to this procedure to produce a streaming algorithm with low space requirements.

1. **Maintaining the sketch under updates** The l_0 sketch is initially the zero vector, since this is the sketch of an empty stream. We can then build the sketch progressively as each item in the data stream is received. Our update procedure on receiving tuple (i, d_k) is as follows: we add d_k times $x_{i,j}$ to each entry $sk(\boldsymbol{a})_j$ ($1 \leq j \leq m$) in the sketch. That is,

   $$\forall 1 \leq j \leq m : sk(\boldsymbol{a})_j \leftarrow sk(\boldsymbol{a})_j + d_k x_{i,j}$$

   Clearly, this procedure ensures that at any point, the sketch is indeed the dot product of the vector $\boldsymbol{a}$ with $\boldsymbol{x}$, as required.

2. **Reducing Space Requirements** To complete the proof of Theorem 5.1 we need to show that this technique can be implemented in small space. So we do not wish to precompute and store all the values $x_{i,j}$. To do so would consume much more space than simply recording each a_i from which the number of distinct items could be easily found. Instead, we will generate $x_{i,j}$ when it is needed. Note that $x_{i,j}$ may be needed several times during the course of the algorithm, and must take the same value each time. We can achieve this by using pseudo-random generators for the generation of the values from the stable distributions. In other words, we will use standard techniques to generate a sequence of pseudo-random numbers from a seed value (see for example [36]). We will use i to seed a pseudo-random number generator $random()$. We will then use the stream of pseudo-random numbers generated by $random()$ to generate a sequence of p-stable distributed random variables $x_{i,1}, x_{i,2}, \ldots x_{i,m}$. This ensures that $x_{i,j}$ takes the same value each time it is used, since we use the same seed i each time, but that it still has the properties of being drawn from a p-stable distribution. Results in [28] assure us that we can use random number generators in place of a true source of randomness with out any fear of loss of quality of the results.

3. **Generating values from Stable Distributions** We need to be able to generate values from a stable distribution with a very small stability parameter p. This can be done using standard methods such as those described in [35]. These take uniform random numbers r_1, r_2 drawn from the range $[0 \ldots 1]$ and output a value drawn from a stable distribution with parameter p. This is much like the Box-Muller procedure for drawing values from the Normal distribution. We denote this transform as a (deterministic) function $stable(r_1, r_2, p)$ computable in constant time. This function is defined as follows: first, we compute $\theta = \pi(r_1 - \frac{1}{2})$. Then

   $$stable(1/2{+}\theta/\pi, r_2, p) = \frac{\sin p\theta}{\cos^{1/p}\theta} \left(\frac{\cos(\theta(1-p))}{-\ln r_2} \right)^{\frac{1-p}{p}}$$

   To use the result of Lemma 5.1, we need to find $\operatorname{median} |X_0|^p$, the median of absolute values from a stable distribution with parameter p. We can do this in advance using numeric methods and then scale by this constant factor to find the desired result, denoted $scalefactor(p)$ in our algorithm.

This then gives the algorithm for maintaining a sketch for computing the Hamming norm: see Figure 2. There are two basic parts: the sketch is updated based on every item encountered in the stream; the approximation of the l_0 norm is found by returning the median value of the absolute values of the sketch vector, scaled appropriately. The algorithm in Figure 2 implements the method we have described, and concludes the proof of Theorem 5.1. ∎

```
initialise sk[1...m] = 0.0
for all tuples (i, dk) do
  initialise random with i
  for j = 1 to m do
    r1 = random()
    r2 = random()
    sk[j] = sk[j] + dk * stable(r1, r2, p)
for j = 1 to m do
  sk[j] = absolute(sk[j])^p
return median_j(sk[j]) * scalefactor(p)
```

Figure 2: Algorithm to approximate the Hamming norm

5.2 Computing Norms of Multiple Streams

With relatively little modification, the above method can be used to find the union or difference of two or more streams.

Theorem 5.3 *The Hamming difference, $|\boldsymbol{a} - \boldsymbol{b}|_H$ can be computed using sketches of size $O(1/\epsilon^2 \log 1/\delta)$. The difference is then approximated within a factor of $1 \pm \epsilon$ with probability $1 - \delta$.*

Proof. The Hamming distance between two streams can be computed using just the Hamming norm, since it is equal to $|\{i|a_i \neq b_i\}| = |\{i|(a_i - b_i) \neq 0\}| = |\boldsymbol{a} - \boldsymbol{b}|_H$. Hence we need to find $sk(\boldsymbol{a} - \boldsymbol{b})$ As seen before, the sketch $sk(\boldsymbol{a})$ is the result of the dot product of the induced vector $\boldsymbol{a}$ with the matrix of values $x_{i,j}$. So $sk(\boldsymbol{a} - \boldsymbol{b}) = sk(\boldsymbol{a}) - sk(\boldsymbol{b})$ follows immediately from the fact that the dot product is a linear function. Therefore, to find the approximate Hamming distance between two streams we have the following simple procedure: (1) Find sketches for each stream individually as before (2) Compute a new sketch as $sk(\boldsymbol{a}) - sk(\boldsymbol{b})$. (3) Find the Hamming norm of this compound sketch as described above, by finding the median value. ∎

Theorem 5.4 *The Hamming norm of the union of multiple streams, $|\boldsymbol{a} + \boldsymbol{b} + \ldots|_H$ can be computed using sketches of size $O(1/\epsilon^2 \log 1/\delta)$. The norm is then approximated within a factor of $1 \pm \epsilon$ with probability $1 - \delta$.*

This also results from the linearity of the dot product function, in the same way to the above proof. It follows that the union of multiple streams $\boldsymbol{a}, \boldsymbol{b} \ldots = \boldsymbol{a} + \boldsymbol{b} + \ldots$ and so a sketch for this can be computed as $sk(\boldsymbol{a} + \boldsymbol{b} + \ldots) = sk(\boldsymbol{a}) + sk(\boldsymbol{b}) + \ldots$. This in particular allows the computation of the number of distinct elements over several separate streams, without overcounting any elements common to two or more of the streams.

6 Experiments

We implemented our method of using stable distributions to approximate the Hamming norm of a stream and Hamming norm of two or more streams. We also implemented Probabilistic Counting as described in Section 3.2 for approximating the number of distinct elements, since this is the method that comes closest to being able to compute the Hamming norm of a sequence. This allows us to test how well our methods perform for the two main motivating applications: counting the number of distinct items in a stream, and comparing streams of network data.

6.1 Implementation Issues

For computing with stable distributions, we implemented the method of [35] to generate stable distributions with arbitrary stability parameters. The transformation $stable(r_1, r_2, p)$ takes two values drawn from a uniform $[0, 1]$ distribution (r_1, r_2) and outputs a value drawn from a stable distribution with parameter p.

Ideally, we set the stability parameter p of the stable distribution to be as low as possible, to approximate as well as possible the actual Hamming norm. However, as p gets closer to zero, the values generated from stable distributions get significantly larger, gradually exceeding the range of floating point representation. Through experimentation, we found the smallest value of p that did not generate floating point overflow was 0.02. Hence we set p to this value, and empirically found the median of the stable distribution as generated by this procedure, which is $1.425 = scalefactor(0.02)$. This median is used in the scaling of the result to get the approximation of the Hamming norm. Note that using $p = 0.02$ means that, even if **every** distinct element occurs a million times, then the contribution by every distinct element to the Hamming norm will be $(10^6)^{0.02} = 1.318$, so this gives a worst case overestimate of 32%. This could be a large source of error, although even this level of error is likely to be acceptable for many applications. In fact we shall see that most of our experiments show an error of much less than 10%.

Experimental Environment Experiments were run on a Sun Enterprise Server on one of its UltraSparc 400MHz processors. To test our methods, we used a mixture of synthetic data generated by random statistical distributions, and real data from network monitoring tools. For this, we obtained 26Mb of streaming NetFlow data [33], from an AT&T network. We performed a series of experiments, firstly to compare the accuracy of using sketches against existing methods for counting the number of distinct elements. We started by comparing our approach with the probabilistic counting algorithm for the insertions-only case (i.e., no deletions). We then investigated the problem for streams where both insertions and deletions were allowed. Next we ran experiments on the more general situations presented by Network data with streams whose entries in the implicit vectors are allowed to be negative. As mentioned earlier, probabilistic counting techniques can fail dramatically when presented with this situation. Finally, we ran experiments for computing the Hamming distance between network data streams and on the union of multiple data streams. We used only the stable distributions method, since this is the only method which is able to solve this problem using very small working space in the

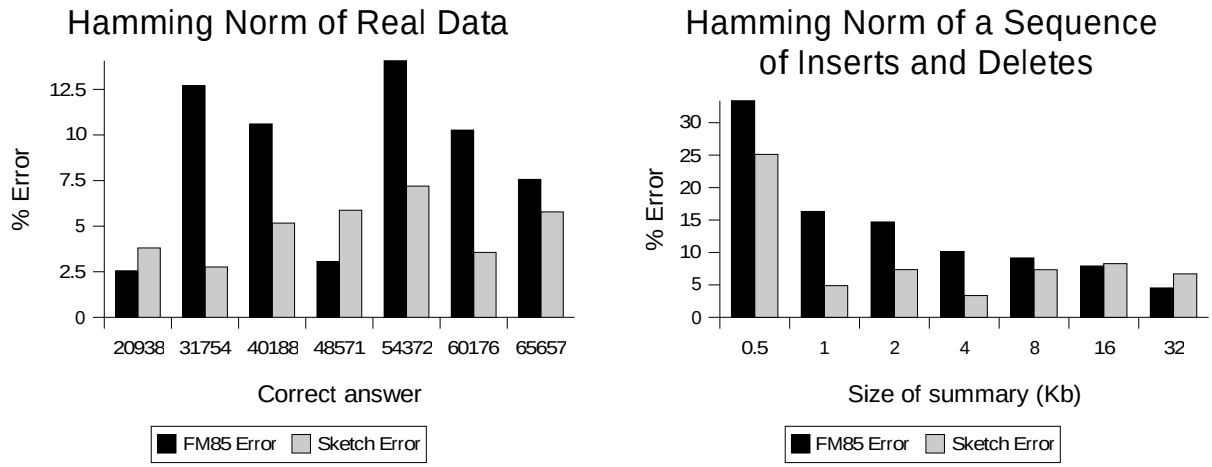

Figure 3: Testing performance on finding the Hamming Norm

data streams model.

For our experiments, the main measurement that we gathered is how close the approximation was to the correct value. This was done by using exact methods to find the correct answer ($exact$), and then comparing this to the approximation ($approx$). Then a percentage error was calculated simply as $(\max(exact, approx)/\min(exact, approx) - 1) \times 100\%$.

Timing Issues Under our initial implementation, the time cost of using stable distributions against using probabilistic counting was quite high — a factor of about six or seven times, although still only a few milliseconds per item. The time can be much reduced at the cost of some extra space usage. This is due to the fact that the majority of the processing time is in creating the values of the stable distribution using a transformation from uniform random variables. By creating a look-up table for computing a stable distribution with a fixed stability parameter we avoid this processing time. This table is indexed by values from a uniform distribution, and a value interpolated from the values in the table. This is analogous to printed statistical tables which allow finding values from, for example, the Normal Distribution. Clearly there is a compromise between the table size and fidelity of the interpolated values to the true stable distribution. The extra space cost could be shared, if there are multiple concurrent computations to find the Hamming norm of several different data streams (for example, multiple attributes in a database table). This approach would also be suitable for use in embedded monitoring systems, since the method only requires simple arithmetic operations and a small amount of writable memory. A compromise solution is to use a cache to store the random variables corresponding to some recently encountered attribute values. If there is locality in the attribute values then cache misses will be small.

6.2 Results of Hamming Norm Experiments

Hamming Norm of Network Data We first examined finding the Hamming norm of the sequence of IP addresses, to find out how many distinct hosts were active. We used exact methods to be able to find the error ratio. We increased the number of items examined from the stream, and looked at between 100,000 and 700,000 items in 100,000 increments. The results presented on the left of Figure 3 show that there were between 20,000 and 70,000 distinct IP addresses in the stream. Both probabilistic counting and sketches were used, given a workspace of 8Kb. Again, using sketches is highly competitive with probabilistic counting, and is on the whole more reliable, with an expected error of close to 5% against probabilistic counting, which is nearer to 10%.

This should also be compared against sampling based methods as reported in [19], where the error ratio was frequently in excess of 200%. This shows that for comparable amounts of space usage, the two methods are competitive with each other for counting distinct items. The worst case for the l_0 sketch occurs when the number of distinct elements is very low (high skew), and here exact methods could be used with small additional space requirements.

Streams based on sequences of inserts and deletes Our second experiment tested how the methods worked on more dynamic data, with a mixture of insertions and deletions. It also tested how much they depend on the amount of working space. We created a sequence of insertions and deletions of items, to simulate addition and removal of records from a database table. This was done by inserting an element with one probability, p_1, and removing an element with probability p_2, while ensuring that for each element i, the number of such elements seen was never less than zero. Again, 100,000 transactions were carried out to test the implementation.

We ran a sequence of experiments, varying the amount of working space allocated to the counting programs, from

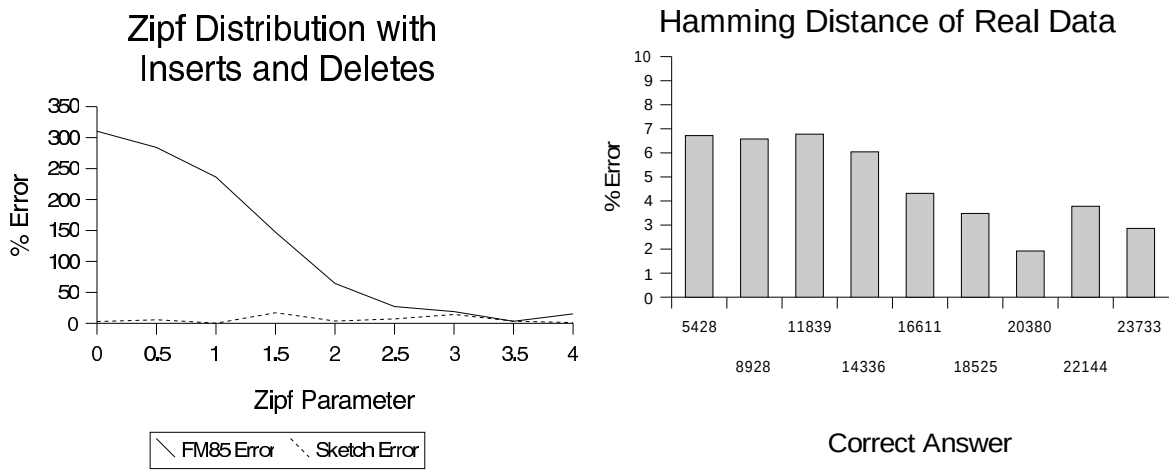

Figure 4: Testing performance for Network Monitoring

0.5Kb, up to 32Kb. The results are shown on the right of Figure 3. The first observation is that the results outdo what we would expect from our theoretical limits from Theorem—5.1. Even with only 1Kb of working space, the sketching procedure using stable distributions was able to compute a very accurate approximation, correct to within a few percentage points. It is important to note that l_0 sketches were able to nearly equal or better the fidelity of probabilistic counting in every case, and also offer additional functionality. Although in this example the quality of the result does not improve as more working space is made available for it, we claim that this is due to the algorithm being fortunate in small space, since these procedures are in their nature strongly probabilistic. Certainly, with a workspace of only a few kilobytes, we can be sure of a result which is highly likely to be within a few percentage points of the correct answer. This is more than good enough for most of the applications we have already mentioned.

Hamming norm of unrestricted streams We generated a set of synthetic data to test the method's performance on the more general problems presented by Network data. The main purpose of the next experiment was to highlight that existing methods are unable to cope with many data sets. Zipf distributions with a variety of skewness parameters were used. Additionally, when a value was generated, a coin was tossed: with probability $\frac{1}{2}$ the transaction is an insertion of that element, and with probability $\frac{1}{2}$ it is a deletion of that element. The results are presented on the left of Figure 4. When we compare the results of probabilistic counting on this sequence to the Hamming norm of the induced vector, we see massive disparity. The error fraction ranges from 20% to 400% depending on the skew of the distribution, and it is on the uniform distribution on which this procedure performs the worst. On the other hand, using sketches gives a result which is consistently close to the correct answer, and in the same region of error as the previous experiments. Probabilistic counting is only competitive at computing the Hamming norm for distributions of high skew. This is when the number of non-zero entries is low (less than 100), and so it could be computed exactly without difficulty.

Hamming Distance between Network Streams Our second experiment on network data was to investigate finding the Hamming distance (dissimilarity) between two streams. This we did by construction, to observe the effect as the Hamming distance increased. We fixed one stream, then constructed a new stream by merging this stream with a second. With probability p we took item i from the first stream, and with probability $1 - p$ we took item i from the second, for 100,000 items. We then created sketches for both streams and found their difference using sketches of size 8Kb. The results are shown in the right hand chart of Figure 4 as we varied the probability from 0.1 to 0.9. Here, it was not possible to compare to existing approximation methods, since no other method is able to find the Hamming distance between streams.

The performance of sketching shows high fidelity. Here, the answers are all within 7% of the true answer, and the trend is for the quality to improve as the size of the Hamming distance between the streams increases. This is because the worst performance of sketches for this problem is observed when the number of different items is low (when the norm is low). Hence sketches are shown to be good tools for this network monitoring task.

Union of Multiple Data Streams Finally, we tested the stated properties of summing sketches to merge data streams. We again used real network data, and for each experiment we split a stream of 100,000 values into a number of pieces. A sketch for each piece was computed separately, and then these sketches combined, and compared against the result found when a single sketch was computed. The results found confirm the result of Theorem 5.4 completely: perhaps remarkably, no matter how many pieces the stream is split into (2, 5, 10, 20, 50 or 100), the final result is exactly the same. In this case, the norm was approximated as 18745.07, an error of 3.96% from the

real value. We might have been concerned that round off errors from floating point arithmetic could cause discrepancies between the answers, but this turns out not to be the case.

7 Concluding Remarks

We have proposed the computation of the Hamming norm of a stream as a basic operation of interest. Computing the Hamming norm of a single stream is a generalisation of the problem of maintenance of the number of distinct values in a database under insertions and deletions, which is of fundamental interest. The Hamming norm of two (or more) streams gives the size of their union or the number of places where they differ, depending on whether the norm is computed on the sum of the streams or the difference respectively. Both these estimations are of great interest as well. In spite of its importance, no algorithms were previously known for computing the Hamming Norm.

We presented a novel and efficient algorithm for computing the "l_0 sketch" for any data stream such that its Hamming norm can be estimated to an arbitrarily small factor using only the sketches. The sketches are very small in size and can be computed in a distributed manner. They can be added to obtain the sketch of the merged stream or subtracted to obtain the sketch of the "difference stream". Their size and accuracy does not depend upon the data distribution. Our experiments with real network flow data demonstrate the powerful accuracy of our algorithm, which outperformed existing methods (where applicable) significantly.

Our Hamming norm estimation technique is more general than is needed in the applications described. For example, we can allow entries in the implicit vector to become non-integral. Also, our solution will work even if some entries become negative which can occur in certain situations. New applications may arise in the future where these features will find greater use.

The "sketch" we compute for estimating Hamming norms is suitable for indexing. That is, once we have computed the short "sketches" for data streams, we can cluster, perform similarity and other proximity searches on the streams using *only* the sketches. This is a powerful feature which will be desirable in emerging stream databases.

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *JCSS: Journal of Computer and System Sciences*, 58, 1999.

[2] Amazon project. `http://www.cs.cornell.edu/database/amazon/amazon.htm`

[3] S. Babu, L. Subramanian, and J. Widom. A data stream management system for network traffic management. In *Proceedings of Workshop on Network-Related Data Management*, 2001.

[4] S. Babu and J. Widom. Continuous queries over data streams. *ACM Sigmod Record*, 30(3):109–120, 2001.

[5] J. Bunge and M. Fitzpatrick. Estimating the number of species: A review. *Journal of the American Statistical Association*, 88(421):364, 1993.

[6] M. Charikar, S. Chaudhuri, R. Motwani, and V. R. Narasayya. Towards estimation error guarantees for distinct values. In *Proceedings of the Nineteenth Symposium on Principles of Database Systems*, pages 268–279, 2000.

[7] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: How much is enough? *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 27(2):436–447, 1998.

[8] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. Ullman, and C. Yang. Finding interesting associations without support pruning. In *Proceedings of 16th International Conference on Data Engineering*, pages 489–500, 2000.

[9] G. Cormode, P. Indyk, N. Koudas, and S. Muthukrishnan. Fast mining of tabular data via approximate distance computations. In *Proceedings of the International Conference on Data Engineering*, 2002.

[10] C. Cranor, L. Gao, T. Johnson, and O. Spatscheck. Gigascope: High performance network monitoring with an SQL interface. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2002.

[11] P. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structures or how to build a data quality browser. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2002.

[12] DIMACS workshop on streaming data analysis and mining; and DIMACS workshop on sublinear algorithms. Center for Discrete Mathematics and Computer Science, `http://dimacs.rutgers.edu/Workshops/`

[13] P. Domingos, G. Hulten, and L. Spencer. Mining time-changing data streams. In *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining*, 2001.

[14] 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.

[15] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford. Netscope: Traffic engineering for IP networks. *IEEE Network Magazine*, pages 11–19, 2000.

[16] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational IP networks: Methodology and experience. *IEEE/ACM Transactions on Networking*, pages 265–279, 2001.

[17] P. Flajolet and G. N. Martin. Probabilistic counting. In *24th Annual Symposium on Foundations of Computer Science*, pages 76–82, 1983.

[18] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for database applications. *Journal of Computer and System Sciences*, 31:182–209, 1985.

[19] P. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *27th International Conference on Very Large Databases*, 2001.

[20] P. Gibbons and Y. Matias. Synopsis structures for massive data sets. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, A, 1999.

[21] P. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings of the 13th ACM Symposium on Parallel Algorithms and Architectures*, 2001.

[22] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. QuickSAND: Quick summary and analysis of network data. Technical Report 2001-43, DIMACS, 2001.

[23] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of 27th International Conference on Very Large Data Bases*, 2001.

[24] M. Grossglauser, N. Koudas, Y. Park, and A. Varriot. Falcon: Fault management via alarm warehousing and mining. In *Proceedings of Workshop on Network-Related Data Management*, 2001.

[25] S. Guha, N. Koudas, and K. Shim. Data streams and histograms. In *Proceedings of Symposium on Theory of Computing*, pages 471–475, 2001.

[26] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proceedings of 41st Annual Symposium on Foundations of Computer Science*, 2000.

[27] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proceedings of the 21st International Conference on Very Large Databases*, pages 311–322, 1995.

[28] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the 40th Symposium on Foundations of Computer Science*, 2000.

[29] P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *Proceedings of the 26th International Conference on Very Large Databases*, pages 363–372, 2000.

[30] W.B. Johnson and J. Lindenstrauss. Extensions of Lipshitz mapping into Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[31] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proceedings of 18th International Conference on Data Engineering*, 2002.

[32] D. Moore, G. M. Voelker, and S. Savage. Inferring Internet denial of service activity. In *Proceedings of the Usenix Security Symposium*, 2001.

[33] Cisco NetFlow. More details at `http://www.cisco.com/warp/public/732/Tech/netflow/`

[34] NOAA. National Oceanic and Atmospheric Administration, U.S. National Weather Service. `http://www.nws.noaa.gov/`.

[35] J. Nolan. Stable distributions. Available from `http://academic2.american.edu/~jpnolan/stable/chap1.ps`

[36] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 1992.

[37] California PATH Smart-AHS. More details at `http://path.berkeley.edu/SMART-AHS/index.html`

[38] N. Thaper, S. Guha, P. Indyk, and N. Koudas. Multidimensional dynamic histograms. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2002.

[39] Unidata. Atmospheric data repository. `http://www.unidata.ucar.edu/`