Processing Set Expressions over Continuous Update Streams

Sumit Ganguly
Bell Laboratories
Lucent Technologies
Murray Hill, NJ 07974
sganguly@bell-labs.com

Minos Garofalakis
Bell Laboratories
Lucent Technologies
Murray Hill, NJ 07974
minos@bell-labs.com

Rajeev Rastogi
Bell Laboratories
Lucent Technologies
Murray Hill, NJ 07974
rastogi@bell-labs.com

ABSTRACT

There is growing interest in algorithms for processing and querying continuous data streams (i.e., data that is seen only once in a fixed order) with limited memory resources. In its most general form, a data stream is actually an *update* stream, i.e., comprising data-item deletions as well as insertions. Such massive update streams arise naturally in several application domains (e.g., monitoring of large IP network installations, or processing of retail-chain transactions).

Estimating the cardinality of set expressions defined over several (perhaps, distributed) update streams is perhaps one of the most fundamental query classes of interest; as an example, such a query may ask "what is the number of distinct IP source addresses seen in passing packets from both router R_1 and R_2 but not router R_3?". Earlier work has only addressed very restricted forms of this problem, focusing solely on the special case of *insert-only* streams and specific operators (e.g., union). In this paper, we propose the *first* space-efficient algorithmic solution for estimating the cardinality of full-fledged set expressions over general update streams. Our estimation algorithms are probabilistic in nature and rely on a novel, hash-based synopsis data structure, termed *"2-level hash sketch"*. We demonstrate how our 2-level hash sketch synopses can be used to provide low-error, high-confidence estimates for the cardinality of set expressions (including operators such as set union, intersection, and difference) over continuous update streams, using only small space and small processing time per update. Furthermore, our estimators never require rescanning or resampling of past stream items, regardless of the number of deletions in the stream. We also present lower bounds for the problem, demonstrating that the space usage of our estimation algorithms is within small factors of the optimal. Preliminary experimental results verify the effectiveness of our approach.

1. INTRODUCTION

Query-processing algorithms for conventional Database Management Systems (DBMS) rely on (possibly) several passes over a collection of *static* data sets in order to produce an accurate answer to a user query. For several emerging application domains, however, updates to the data arrive on a continuous basis, and the query processor needs to be able to produce answers to user queries based solely on the observed stream of data and without the benefit of several passes over a static data image. As a result, there has been a flurry of recent work on designing effective query-processing algorithms that work over continuous *data streams* to produce results online while guaranteeing (1) small memory footprints, and (2) low processing times per stream item [1, 10, 13, 15]. Such algorithms typically rely on summarizing the data stream(s) involved in concise *synopses* that can be used to provide *approximate answers* to user queries along with some reasonable guarantees on the quality of the approximation.

In their most general form, real-life data streams are actually *update streams*; that is, the stream is a sequence of updates to data items, comprising data-item deletions as well as insertions [1]. Such continuous update streams arise naturally, for example, in the network installations of large Internet service providers, where detailed usage information (SNMP/RMON packet-flow data, active VPN circuits, etc.) from different parts of the underlying network needs to be continuously collected and analyzed for interesting trends. Other application domains giving rise to continuous and massive update streams include retail-chain transaction processing (e.g., purchase and sale records), ATM and credit-card operations, logging Web-server usage records, and so on. The processing of such streams follows, in general, a *distributed* model where each stream (or, part of a stream) is observed and summarized by its respective party (e.g., the element-management system of an individual IP router) and the resulting synopses are then collected (e.g., periodically) at a central site, where queries over the entire collection of streams can be processed [15]. This model is used, for example, in Lucent's Interprenet and Cisco's NetFlow products for IP network monitoring.

Clearly, there are several forms of queries that users or applications may wish to pose (online) over such continuous update streams; examples include joins or multi-joins [1, 10], norm computations [2, 19], or quantile estimation [16]. Perhaps one of the most fundamental queries of interest is estimating the result cardinalities of set expressions defined

[1]Item modifications are simply seen as a deletion directly followed by an insertion of the modified item.

over several update streams. As an example, an application monitoring active IP-sessions may wish to correlate the IP-session sources seen at routers R_1, R_2, and R_3 by posing a query such as: "estimate the number of distinct IP addresses seen at both R_1 and R_2 but not R_3". This is simply the number of distinct elements (i.e., set cardinality) for the (multi-)set $(\texttt{source}(R_1) \cap \texttt{source}(R_2)) - \texttt{source}(R_3)$, where $\texttt{source}(R_i)$ is the multi-set of IP source addresses seen at router R_i. The ability to effectively estimate the cardinality of such set expressions over the observed streams of updates for IP-session data in the underlying network can be crucial in quickly detecting possible denial-of-service attacks, network routing or load-balancing problems, potential reliability concerns (catastrophic points-of-failure), and so on. Set expressions are also an integral part of query languages for relational database systems; for example, the SQL standard supports set operators like `UNION`, `INTERSECT`, and `EXCEPT` (i.e., difference) in queries over tables with compatible schemas [22]. Thus, one-pass synopses for effectively estimating set-expression cardinalities can be extremely useful, e.g., in the optimization of such queries over Terabyte relational databases.

Prior Work. Estimating the cardinality of *set union* (i.e., number of distinct elements) over (one or more) element streams is a very basic problem with several practical applications (e.g., query optimization); as a result, several solutions have been proposed in the literature for the set-union estimation problem. In their influential paper, Flajolet and Martin [12] propose a randomized estimator for distinct-element counting that relies on a hash-based synopsis data structure; to this date, the Flajolet-Martin (FM) technique remains one of the most effective approaches for this estimation problem. The analysis of Flajolet and Martin makes the (unrealistic) assumption of an explicit family of hash functions exhibiting ideal random properties; in a later paper, Alon et al. [2] present a more realistic analysis of the FM estimator that relies solely on simple, linear hash functions. Several estimators based on uniform random sampling have also been proposed for distinct-element counting [6, 17]; however, such sampling-based approaches are known to be inaccurate and substantial negative results have been shown by Charikar et al. [6] stating that accurate estimation of the number of distinct values (to within a small constant factor with constant probability) requires nearly the entire data set to be sampled! More recently, Gibbons et al. [14, 15] have proposed specialized sampling schemes specifically designed for distinct-element counting; their sampling schemes rely on hashing ideas (similar to [12]) to obtain a random sample *of the distinct elements* in the input streams that is then used for estimation. Finally, Bar-Yossef et al. [4] propose improved distinct-count estimators that combine new techniques and ideas from [2, 12, 15].

All these earlier papers on set-union estimation either ignore the possibility of deletions in the input stream(s) or fail to deal with deletions in a completely satisfactory manner. For example, sampling-based solutions like [14, 15] may very well require rescanning and resampling of past stream items when deletions cause the maintained sample to be depleted; this is clearly an unrealistic requirement in a data-streaming environment. More importantly, none of the above-mentioned papers addresses the problem of dealing with general set expressions (including operators like set intersection or difference), which is obviously significantly more complex than simple set union.

The method of *Minwise Independent Permutations (MIPs)* [5, 8, 18] is, to the best of our knowledge, the only known technique that can accurately estimate the result cardinalities of set operators other than union (e.g., intersection) over an insertion stream rendering a multi-set of data items. Furthermore, extending the basic technique to deal with set expressions is relatively straightforward (e.g., see [7]). Unfortunately, MIPs are also ill-equipped for dealing with general *update* streams. Deletions can easily deplete the MIP synopsis, rendering it useless for the purposes of set-expression estimation unless we are able to rescan past stream items; again, however, this is not a realistic option in a data-stream setting.

Our Contributions. In this paper, we present the *first* space-efficient algorithmic solution for the full-fledged problem of estimating set-expression cardinalities over general update streams. Our proposed estimators are probabilistic in nature and rely on a novel, hash-based synopsis data structure, termed *"2-level hash sketch"*. We present novel estimation algorithms that use our 2-level hash sketch stream synopses to provide low-error, high-confidence estimates for the cardinality of general set expressions (including set union, intersection, and difference operators) over continuous update streams, using only small space and small processing time per update. We also present lower bounds demonstrating that the space usage of our basic estimators is within small factors of the best possible for any (randomized) solution. Furthermore, our estimators never require rescanning or resampling of past stream items, regardless of the number of deletions in the stream: at any point in time, our 2-level hash sketch summary is guaranteed to be *identical* to that obtained if the deleted items had never occurred in the stream! More concretely, the key contributions of our work are summarized as follows.

• **Novel 2-level Hash Sketch Synopses and Basic Set-Operator Estimators over Update Streams.** We formally introduce the 2-level hash sketch synopsis data structure and describe its maintenance over a continuous stream of updates (rendering a multi-set of data elements). Briefly, 2-level hash sketches extend the hash-based synopses of Flajolet and Martin [12] in a non-trivial manner that renders them (a) robust to item deletions in the stream, and (b) useful for estimating the cardinalities of set difference and intersection (in addition to set union). We then present novel algorithms for (probabilistically) estimating the cardinalities of the three basic set operations (union, difference, and intersection) over 2-level hash sketches. To simplify the analysis of our basic estimators, we initially assume ideal, fully-independent hash mappings for 2-level hash sketch construction; we then demonstrate how our analysis carries over to the (more realistic) limited-independence case. We also prove a lower bound for all randomized approximation algorithms, showing that the space requirements of our estimators is actually within small polynomial and log factors of the optimal.

• **Extension to General Set-Expression Estimation over Update Streams.** We generalize our basic-operator estimators (and their analysis) to derive an accurate, small-space (probabilistic) estimation algorithm for the cardinality of general set expressions over a collection of continuous update streams. Once again, ours is the first approach to solve this estimation problem for arbitrary update streams, while guaranteeing that no access to past stream items will ever be needed. Furthermore, even though we present our estimators in a single-site setting, our solution also naturally extends to the more general *"distributed-streams model with stored coins"* of Gibbons and Tirthapura [15].

• **Experimental Results Validating our Methodology.** We present preliminary results from an experimental study with different synthetic data sets that verify the effectiveness of our 2-level hash sketch synopses and estimation algorithms. The results substantiate our theoretical claims, demonstrating the ability of our techniques to provide space-efficient and accurate estimates for set-expression cardinality queries over continuous streaming data.

2. PRELIMINARIES

In this section, we discuss the basic elements of our update-stream processing architecture and introduce some key concepts and notation for our estimation algorithms. We also describe the hash-based Flajolet-Martin (FM) distinct-value count estimator in more detail, as it will provide the basis for our 2-level hash sketch synopses (introduced in Section 3).

2.1 Update-Stream Processing Model

The key elements of our update-stream processing architecture for set-expression estimation are depicted in Figure 1; similar architectures for processing data streams have been described elsewhere (see, for example, [10]).

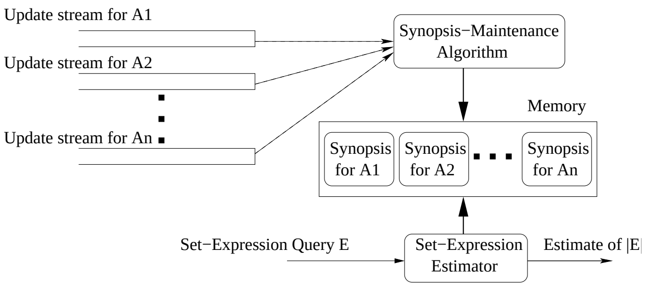


Figure 1: Update-Stream Processing Architecture.

Without loss of generality, we assume that each input stream renders a multi-set A_i of elements from the integer domain $[M] = \{0, \ldots, M-1\}$ as a continuous stream of updates. (To simplify the exposition, we also assume that M is a power of 2.) Each such update is a triple of the form $< i, e, \pm v >$, where i identifies the multi-set A_i being updated, $e \in [M]$ denotes the specific data element whose frequency changes, and $\pm v$ is the net change in the frequency of e in A_i, i.e., "$+v$" ("$-v$") denotes v insertions (resp., deletions) of e. We assume that all deletions in our update streams are legal; that is, an update $< i, e, -v >$ can only be issued if the net frequency of e in A_i is at least v. We also let N denote an upper bound on the total number of data elements (i.e., the sum of element frequencies) in any multi-set A_i. In contrast to conventional DBMS processing, our stream processor is allowed to see the update tuples for each A_i *only once* and in the fixed order of arrival as they stream in from their respective source(s). Backtracking over an update stream and explicit access to past update tuples are impossible.

Given a set expression E over the multi-set streams A_i, we use $|E|$ to denote the number of distinct elements in E whose net frequency is greater than zero; for example, $|A_1 \cup A_2|$ is the number of distinct elements in the union of streams A_1 and A_2. Our stream-processing engine is allowed a certain amount of memory, typically significantly smaller than the total size of its input(s). This memory is used to maintain a concise *synopsis* for each update stream A_i. The key constraints imposed on such synopses are that: (1) they are much smaller than the number of elements in A_i (e.g., their size is logarithmic or polylogarithmic in $|A_i|$); and, (2) they can be easily maintained, during a single pass over the update tuples for A_i, in the (arbitrary) order of their arrival. At any point in time, given an arbitrary set expression E over the A_i's, our set-expression cardinality estimator can combine the maintained collection of synopses to produce an estimate for $|E|$.

Even for the simpler case of insert-only streams, communication complexity arguments can be applied to show that the *exact computation* of set-expression cardinalities requires at least $\Omega(M)$ space[2], even for randomized algorithms [21, 20]. Instead, our focus is to *approximate* the quantity $X = |E|$ to within a small relative error, with high confidence. Thus, we seek to obtain a (randomized) (ϵ, δ)-approximation scheme [2, 15], that computes an estimate $\hat{X}$ of X such that $\mathbf{Pr}\left[|\hat{X} - X| \leq \epsilon X\right] \geq 1 - \delta$.

2.2 The Flajolet-Martin Distinct-Count Estimator

The Flajolet-Martin (FM) technique [12] for estimating the number of distinct elements (i.e., set-union cardinality) over a stream of insertions relies on a family of hash functions $\mathcal{H}$ that map incoming data values uniformly and independently over the collection of binary strings in the input data domain $[M]$. It is then easy to see that, if $h \in \mathcal{H}$ and $\textsc{lsb}(s)$ denotes the position of the *least-significant* 1 *bit* in the binary string s, then for any $i \in [M]$, $\textsc{lsb}(h(i)) \in \{0, \ldots, \log M - 1\}$ and $\mathbf{Pr}\left[\textsc{lsb}(h(i)) = l\right] = \frac{1}{2^{l+1}}$.[3] The basic hash synopsis maintained by an instance of the FM algorithm (i.e., a specific choice of hash function $h \in \mathcal{H}$) is simply a bit-vector of size $\Theta(\log M)$. This bit-vector is initialized to all zeros and, for each incoming value i in the input multi-set A_i, the bit located at position $\textsc{lsb}(h(i))$ is turned on. Of course, to boost accuracy and confidence, the FM algorithm employs averaging over several independent instances (i.e., r independent choices of the mapping hash-function $h \in \mathcal{H}$ and corresponding synopses). The overall FM algorithm is depicted in Figure 2.

Intuitively, the FM algorithm works since, by the properties of the hash functions in $\mathcal{H}$, we expect a fraction of $\frac{1}{2^{l+1}}$ of the distinct values in A_i to map to location l in each synopsis; thus, we expect $|A_i|/2$ values to map to bit 0, $|A_i|/4$ to map to bit 1, and so on. Therefore, the location of the leftmost zero in a bit-vector synopsis is a good indicator of

[2]The asymptotic notation $f(n) = \Omega(g(n))$ is equivalent to $g(n) = O(f(n))$. Similarly, the notation $f(n) = \Theta(g(n))$ means that functions $f(n)$ and $g(n)$ are asymptotically equal (to within constant factors); in other words, $f(n) = O(g(n))$ *and* $g(n) = O(f(n))$ [9].

[3]All log's in this paper denote base-2 logarithms.

```
procedure EstimateDistinctFM( $S$, $\{h_1(), \ldots, h_r()\}$ )
Input: Stream $S$ of data items (i.e., insertions) in the domain
    $[M] = \{0, \ldots, M-1\}$, family of randomizing hash functions
    $h_i$ $(i = 1, \ldots, r)$.
Output: Estimate $R$ of the number of distinct values in $S$.
begin
1.  for $i := 1$ to $r$ do
2.     bitSketch$_i$[] $:= [0, \ldots, 0]$     // bit-string of length $\Theta(\log M)$
3.  for each $j \in S$ do
4.     for $i := 1$ to $r$ do bitSketch$_i[h_i(j)] := 1$
1.  for $i := 1$ to $r$ do
2.     for $m := \log M - 1$ downto 0 do
3.        if bitSketch$_i[m] = 0$ then leftmostZero $:= m$
4.     sum := sum + leftmostZero
5.  endfor
6.  $R := 1.2928 \times 2^{\text{sum}/r}$
7.  return( $R$ )
end
```

Figure 2: The Flajolet-Martin Distinct-Count Estimation Procedure.

$\log |A_i|$. In fact, Flajolet and Martin proved that the estimation procedure depicted in Figure 2 is guaranteed to return an *unbiased* estimate for $|A_i|$ (i.e., the expected value of the returned quantity R is $E[R] = |A_i|$).

The analysis of Flajolet and Martin actually assumes the existence of an explicit family of hash functions $\mathcal{H}$ exhibiting some ideal random properties (namely, fully-independent value mappings) [12]; unfortunately, such hash functions are impossible to compute in small space. Alon et al. [2] present a more realistic analysis of a very similar scheme (based again on bit-vector hash synopses) that relies solely on linear hash functions (guaranteeing only *pairwise* independence). Such hash functions can be computed using only a seed of size $O(\log M)$ and, as shown in [2], produce synopses that guarantee a distinct-value estimate that is within a constant multiplicative factor with constant probability.

3. PROCESSING SET OPERATORS OVER UPDATE STREAMS

In this section, we describe the key ideas underlying our proposed solution for processing set expressions over continuous update streams. We begin by defining our basic synopsis data structures (termed 2-level hash sketches) and the algorithm for maintaining a 2-level hash sketch over a streams of updates (insertions/deletions) to an input multiset. We also describe some procedures for testing (with high probability) certain elementary properties over our 2-level hash sketch synopses that are used as basic primitives in our set-operator routines. We then present our estimation algorithms for processing the three basic set operations (set union, set difference, and set intersection) over 2-level hash sketch synopses. Our algorithm for union can utilize a simple extension of the FM hashing data structure, so it does not actually require the full power of our 2-level hash sketch synopses. (The case of set union is not the focal point of this paper, as the union sub-problem has already been extensively treated in the literature; however, for the sake of homogeneity, we do present a novel algorithm in the context of our sketches and describe its analysis.) The role of 2-level hash sketches becomes critical in our algorithms for estimating set difference and intersection; to the best of our knowledge, ours is the *first* approach to provide low-error, high-confidence probabilistic estimates for these two set operators for general update streams with arbitrary deletions (without ever requiring resampling or rescanning of the stream).

To simplify the exposition in this section, we first present and analyze our estimation schemes assuming ideal randomizing hash functions that guarantee fully-independent value mappings. Then, in Section 3.6, we demonstrate some key statistical lemmas that enable all our results to carry over to the more realistic limited-independence case. More specifically, we show that our analysis can be carried out assuming only $O(\log \frac{1}{\epsilon})$-wise independence, where ϵ denotes the relative-error guarantee provided by our techniques. (An $O(\log \frac{1}{\epsilon})$-wise independent randomizing hash function over $[M]$ can be implemented using only $O(\log \frac{1}{\epsilon} \log M)$ space with standard techniques [3, 18].) Finally, Section 3.7 presents a lower bound on the space usage of any randomized set-operator cardinality estimation algorithm showing that our estimators are within small factors of the best possible solution.

3.1 Our Stream Synopsis: 2-level Hash Sketches

Our proposed synopsis data structure, termed 2-level hash sketch, is a generalization of the basic bit-vector hash synopsis proposed by Flajolet and Martin for distinct-value estimation [12]. 2-level hash sketch synopses rely on two distinct, independent families (i.e., levels) of hash functions $\mathcal{H}$ and $\mathcal{G}$. The first-level hash functions $h \in \mathcal{H}$ are randomizing hash functions that map $[M]$ uniformly onto a range $[M^k]$ (i.e., $h : [M] \rightarrow [M^k]$), where k is a small integer constant (e.g., $k = 2$) used to guarantee that the h mapping over the elements of $[M]$ is injective with high probability. On the other hand, second-level hash functions $g \in \mathcal{G}$ randomize the domain values in $[M]$ uniformly over the binary domain $[2] = \{0, 1\}$ (i.e., $g : [M] \rightarrow [2]$).

A 2-level hash sketch uses one randomly-chosen first-level hash function $h \in \mathcal{H}$ and s independently-chosen second-level hash functions $g_1, \ldots, g_s \in \mathcal{G}$, where s is a parameter of the 2-level hash sketch. As in the Flajolet-Martin algorithm (Figure 2), the first-level hash function h is used in conjunction with the LSB operator to map the domain values in $[M]$ onto a logarithmic range $\{0, \ldots, \Theta(\log M)\}$ of first-level buckets with exponentially decreasing probabilities. Then, each of the s second-level hash functions g_i $(i = 1, \ldots, s)$ is applied to the collection of elements mapping to a given first-level bucket to further map each element to one of two second-level buckets (i.e., 0 or 1) and the corresponding element counter. Conceptually, a 2-level hash sketch for a streaming multi-set A can be seen as a three-dimensional array $\mathcal{X}_A$ of size $\Theta(\log M) \times s \times 2$, where each entry $\mathcal{X}_A[i_1, i_2, i_3]$ is a data-element counter of size $O(\log N)$. The structure of our 2-level hash sketch synopses is pictorially depicted in Figure 3.

Maintenance. The algorithm for maintaining a 2-level hash sketch synopsis $\mathcal{X}_{A_i}$ over a stream of updates to a multi-set A_i is fairly simple. The sketch structure is first initialized to all zeros and, for each incoming update $< i, e, \pm v >$, the element counters at the appropriate locations of the $\mathcal{X}_{A_i}$ sketch are updated; that is, for each $j = 1, \ldots, s$, we simply set $\mathcal{X}_{A_i}[\text{LSB}(h(e)), j, g_j(e)] := \mathcal{X}_{A_i}[\text{LSB}(h(e)), j, g_j(e)] \pm v$. Note here that our 2-level hash sketch synopses are essentially impervious to delete operations; in other words, the sketch obtained at the end of an update stream is *identical*

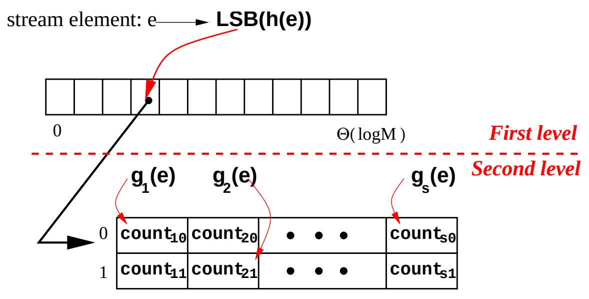


Figure 3: Our 2-level Hash Sketch Synopses.

to a sketch that never sees the deleted items in the stream.

We now proceed to describe our algorithms for processing basic set operators over update streams using our 2-level hash sketch synopses and their analysis. As mentioned earlier, to simplify the exposition, our analysis initially assumes ideal, fully-independent first-level hash functions. (For second-level hash mappings, simple pair-wise independence is sufficient for our analysis (Lemma 3.1).)

3.2 Elementary Property Checks

Our basic set-operator estimators rely on checking certain elementary properties for the collection of elements that map to a given first-level bucket in a 2-level hash sketch synopsis. We now describe the procedures for performing these elementary property checks. Briefly, the key idea here is to make use of the second-level information maintained for the first-level bucket in question; of course, given the space limitations for our 2-level hash sketches, the implications made are necessarily probabilistic (with high confidence for sufficiently large s; see Lemma 3.1).

Our three procedures for checking elementary 2-level hash sketch properties (termed SingletonBucket, IdenticalSingletonBucket, and SingletonUnionBucket) are depicted in Figure 4. Procedure SingletonBucket($\mathcal{X}$, i) returns **true** iff the collection of distinct elements mapping to the i^{th} first-level bucket is a *singleton* (i.e., contains only one distinct element). Given two 2-level hash sketches $\mathcal{X}_A$ and $\mathcal{X}_B$ (for update streams A and B) built using the same first- and second-level hash functions, procedure IdenticalSingletonBucket($\mathcal{X}_A$, $\mathcal{X}_B$, i) returns **true** iff the i^{th} first-level buckets for both $\mathcal{X}_A$ and $\mathcal{X}_B$ (1) are singletons, and (2) contain the exact same distinct value from $[M]$. Finally, procedure SingletonUnionBucket($\mathcal{X}_A$, $\mathcal{X}_B$, i) returns **true** iff the set union of the elements from A and B mapping to the i^{th} first-level bucket of $\mathcal{X}_A$ and $\mathcal{X}_B$ is a singleton.

The key intuition behind our three property checkers is that they employ the randomizing properties of the second-level binary hash signatures in a 2-level hash sketch to derive a high-confidence answer about properties of the element collection in the corresponding first-level bucket. As the following lemma demonstrates, our procedures are guaranteed to draw a valid conclusion with a confidence of at least $1-\delta$ as long as the number of (independent) second-level hash functions s is $\Theta(\log\frac{1}{\delta})$ and each such function g_i is (at least) pair-wise independent.

LEMMA 3.1. *Procedures* SingletonBucket, IdenticalSingletonBucket, *and* SingletonUnionBucket *return the correct value with*

```
procedure SingletonBucket( $\mathcal{X}$, $i$ )
Input: 2-level hash sketch $\mathcal{X}$, first-level bucket index $i$.
Output: true iff $i^{th}$ bucket is a singleton.
begin
1.  if ($\mathcal{X}[i,1,0] + \mathcal{X}[i,1,1] = 0$) return false // bucket is empty
2.  unique := true;  $j := 1$
3.  while ( unique and $j \le s$ ) do
4.     if ( $\mathcal{X}[i,j,0] > 0$ and $\mathcal{X}[i,j,1] > 0$ ) then
5.        unique := false   // at least two elements in this bucket
6.     $j := j + 1$
7.  endwhile
8.  return( unique )
end
```

```
procedure IdenticalSingletonBucket( $\mathcal{X}_A$, $\mathcal{X}_B$, $i$ )
Input: 2-level hash sketches $\mathcal{X}_A$, $\mathcal{X}_B$, first-level bucket index $i$.
Output: true iff $i^{th}$ buckets in $\mathcal{X}_A$ and $\mathcal{X}_B$ contain the same
   singleton value.
begin
1.  if (not SingletonBucket($\mathcal{X}_A$, $i$)) or (not SingletonBucket($\mathcal{X}_B$, $i$))
    then
2.     return( false)
3.  same := true;  $j := 1$
4.  while ( same and $j \le s$ ) do
5.     if ( $(\mathcal{X}_A[i,j,0] > 0) \neq (\mathcal{X}_B[i,j,0] > 0)$ or
            $(\mathcal{X}_A[i,j,1] > 0) \neq (\mathcal{X}_B[i,j,1] > 0)$ ) then
6.        same := false   // buckets differ in at least one element
7.     $j := j + 1$
8.  endwhile
9.  return( same )
end
```

```
procedure SingletonUnionBucket( $\mathcal{X}_A$, $\mathcal{X}_B$, $i$ )
Input: 2-level hash sketches $\mathcal{X}_A$, $\mathcal{X}_B$, first-level bucket index $i$.
Output: true iff the union of $i^{th}$ buckets in $\mathcal{X}_A$ and $\mathcal{X}_B$ is a
   singleton.
begin
1.  if ( (SingletonBucket($\mathcal{X}_A$, $i$) and ($\mathcal{X}_B[i,1,0] + \mathcal{X}_B[i,1,1] = 0$))
    or (SingletonBucket($\mathcal{X}_B$, $i$) and ($\mathcal{X}_A[i,1,0] + \mathcal{X}_A[i,1,1] = 0$)) )
    then
2.        return( true )  // one singleton and one empty bucket
3.  else return( IdenticalSingletonBucket($\mathcal{X}_A$, $\mathcal{X}_B$, $i$) )
end
```

Figure 4: Elementary Property Check Procedures for 2-level Hash Sketches.

probability at least $1-\delta$ *if the number of second-level hash functions is* $s = \Theta(\log\frac{1}{\delta})$ *and each* g_i *is* m*-wise independent with* $m \ge 2$. ∎

Proof (sketch): Consider, as an example, the SingletonBucket procedure. In order to check whether a first-level bucket contains just a single distinct value, the procedure essentially checks each one of the s second level bucket pairs to see if there is at least one pair with both counts positive; if such a pair cannot be found, then the procedure concludes that the bucket is a singleton. Assume that the procedure returns an erroneous conclusion. Clearly, if the bucket is truly a singleton, the SingletonBucket procedure will always return the correct answer; so, the only possible error occurs when the bucket contains at least two distinct values but the SingletonBucket procedure concludes that the bucket is a singleton. The only way this can happen is if, *for each one* of the s second-level bucket pairs, the corresponding hash function maps the two distinct element values to the same binary value (0 or 1). By pair-wise independence, this happens with probability $\frac{1}{2}$ for a given second-level hash function g_j; thus, by the independence of the g_j's the proba-

```
procedure SetUnionEstimator( $\{\mathcal{X}_A^i, \mathcal{X}_B^i\}$ ($i = 1, \ldots, r$), $\epsilon$)
Input: $r$ independent 2-level hash sketch pairs $\{\mathcal{X}_A^i, \mathcal{X}_B^i\}$ for
   streams $A$ and $B$, relative accuracy parameter $\epsilon$.
Output: Estimate for $|A \cup B|$.
begin
1.  $f := (1+\epsilon)r/8$
2.  index := 0
3.  while ( true ) do
4.     count := 0
5.     for $i := 1$ to $r$ do
6.        if ( $\mathcal{X}_A^i$[ index, 1, 0] + $\mathcal{X}_A^i$[ index, 1, 1] > 0 ) or
             ( $\mathcal{X}_B^i$[ index, 1, 0] + $\mathcal{X}_B^i$[ index, 1, 1] > 0 ) then
7.           count := count +1
8.     endfor
9.     if ( count $\le f$ ) then break   // first index with count $\le f$
10.    else index := index +1
11. endwhile
12. $\hat{p}$ := count / $r$ ;  $R := 2^{\text{index}+1}$
13. return( $\frac{\log(1-\hat{p})}{\log(1-1/R)}$ )
end
```

Figure 5: Set-Union Cardinality Estimator.

bility of an erroneous singleton conclusion is upper-bounded by $(\frac{1}{2})^s \le \delta$ for $s = \Theta(\log \frac{1}{\delta})$. ∎

3.3 The Set-Union Estimator

Given two multi-sets of elements A and B in the form of continuous update streams, the set-union cardinality $|A \cup B|$ is the number of distinct elements with positive net frequency in either A or B. We present an (ϵ, δ)-estimator for the set-union cardinality $|A \cup B|$ based on maintained 2-level hash sketch synopses $\mathcal{X}_A$ and $\mathcal{X}_B$ for streams A and B, respectively. Our set-union algorithm does not actually require the full power of 2-level hash sketches, since it does not need to make use of any second-level hash structures. Thus, our union estimator can work by simply maintaining a single counter (of size $O(\log N)$) for each of the $\Theta(\log M)$ first-level hash buckets (Figure 3). [4] This means that, in the simpler case of set union, we can use a simple extension of the basic FM hash data structure.

Our algorithm for producing an (ϵ, δ)-estimator for the union of two update streams A and B (termed SetUnionEstimator) is shown in Figure 5. Briefly, our estimator examines an input collection of r independent 2-level hash sketch synopses built over A and B in parallel (each copy using independently-chosen first- and second-level hash functions from $\mathcal{H}$ and $\mathcal{G}$) in order to determine the smallest first-level bucket index at which only a constant fraction $\le (1+\epsilon)/8$ of the sketch buckets turns out to be non-empty for the union $A \cup B$ (Steps 4-10). (Note that the non-empty **if**-condition in Step 6 can be checked by simply maintaining a single element counter at the corresponding first-level bucket; we present the algorithm assuming full 2-level hash sketch synopses for uniformity.) As our analysis shows, the observed fraction $\hat{p}$ (Step 12) of non-empty first-level hash buckets can be used to provide an estimate for the probability of observing a non-empty bucket at this level of the sketch which, in turn, allows us to give a robust estimate for $|A \cup B|$ (Step 13).

Analysis. We first demonstrate that, when the number of independent input sketches is $r = \Theta(\frac{\log(1/\delta)}{\epsilon^2})$, algorithm SetUnionEstimator terminates with a bucket index for which the non-empty bucket count satisfies $\frac{7(1-\epsilon)r}{128} \le \text{count} \le \frac{(1+\epsilon)r}{8}$ with probability at least $1-\delta$. Consider a specific level j of first-level hash buckets and let p_j denote the probability that bucket j is non-empty for $A \cup B$, i.e., bucket j is not empty in either $\mathcal{X}_A$ or $\mathcal{X}_B$. By independence, this probability is exactly $p_j = 1 - (1 - 1/R_j)^u$, where $u = |A \cup B|$ and $R_j = 2^{j+1}$. A simple application of the binomial expansion gives us that $u/R_j - (1/2)(u/R_j)^2 \le p_j \le u/R_j$.

Now, fix j to be a positive integer such that $1/16 < u/R_j \le 1/8$; for this value j of the bucket index, the above bounds for the probability $p_j = p$ give $7/128 < p \le 1/8$. Note that the ratio count/r at this level j is essentially an average over r independent observations of the 0/1 random variable corresponding to $p_j = p$. By Chernoff bounds, the estimate $\hat{p}_j = \hat{p} = \text{count}/r$ at level j satisfies $|\hat{p} - p| \le \epsilon p$ with probability at least $1 - \delta$ as long as $rp \ge \frac{2\log(1/\delta)}{\epsilon^2}$, or (since $p > 7/128$) $r \ge \frac{256\log(1/\delta)}{7\epsilon^2}$. Consequently, with this value j of the bucket index and $r = \Theta(\frac{\log(1/\delta)}{\epsilon^2})$, our SetUnionEstimator procedure finds $\hat{p} \in (1 \pm \epsilon)p$ which implies that $\frac{7(1-\epsilon)r}{128} \le \text{count} \le \frac{(1+\epsilon)r}{8}$ with probability at least $1-\delta$ (since $7/128 < p \le 1/8$ at this level j).

Thus, with probability $\ge 1 - \delta$, SetUnionEstimator finds a level j such that count $\le \frac{(1+\epsilon)r}{8}$ and the ratio $\hat{p}_j = \text{count}/r$ satisfies $|\hat{p}_j - p_j| \le \epsilon p_j$. The following lemma then demonstrates that, for $p_j \le 1/4$, we can directly substitute the estimate $\hat{p_j}$ in the equation $p_j = 1 - (1 - 1/R_j)^u$ and solve for u without any significant change in the relative accuracy guarantees. (A similar lemma is proven in [4], even though their proposed estimation technique is quite different from ours.)

Lemma 3.2. *Let $f(x) = \log(1-x)/\log(1-1/R)$. If $|y - x| \le \frac{\epsilon}{2}x$ for some $\epsilon < 1$ and $x \le 1/4$, then $|f(y) - f(x)| \le \epsilon f(x)$.* ∎

Proof: By Taylor Series, there is a value in $w \in (x, y)$ such that $\ln(1-y) = \ln(1-x) - (y-x)/(1-w)$ (ln denotes the natural logarithm function). Thus, we have:

$$(-\ln(1-1/R))|f(y) - f(x)| \le \frac{|y-x|}{1 - \max\{x,y\}} \le \frac{\frac{\epsilon}{2}x}{1-(1+\frac{\epsilon}{2})x}.$$

Now, since $x \le 1/4$ and $\epsilon < 1$, we have $(1+\frac{\epsilon}{2})x < 3/8$ which gives: $(-\ln(1-1/R))|f(y) - f(x)| \le \epsilon x \le -\epsilon \ln(1-x)$. Since, for any x, $\ln x = \log x \cdot \ln 2$, the result follows. ∎

We summarize the results of the above analysis in the following theorem.

Theorem 3.3. *Procedure SetUnionEstimator returns an (ϵ, δ)-estimate for the size of the set union $|A \cup B|$ of two update streams A and B using 2-level hash sketch synopses with a total storage requirement of $\Theta(\frac{\log(1/\delta)}{\epsilon^2}\log M \log N)$.* ∎

3.4 The Set-Difference Estimator

Given two multi-sets A and B presented as a continuous stream of updates, the cardinality of the set difference of A and B (i.e., $|A - B|$) is defined as the number of distinct element values whose net frequency is positive in A and zero in B. In this section, we present an (ϵ, δ)-approximation scheme for estimating the set-difference operator over two update streams A and B based on their maintained 2-level

[4] Of course, for the general set-expression estimation problem considered in this paper, we need to build full 2-level hash sketch synopses for each update stream, since we do not know a-priori how a stream will be used in an incoming expression (Figure 1).

```
procedure SetDifferenceEstimator( $\{\mathcal{X}_A^i, \mathcal{X}_B^i\}$ $(i = 1, \ldots, r)$, $\hat{u}$, $\epsilon$ )
Input: $r$ independent 2-level hash sketch pairs $\{\mathcal{X}_A^i, \mathcal{X}_B^i\}$ for
  streams $A$ and $B$, set-union cardinality estimate $\hat{u}$, relative
  accuracy parameter $\epsilon$.
Output: Estimate for $|A - B|$.
begin
1.  sum := count := 0
2.  for $i$ := 1 to $r$ do
3.     atomicEstimate := AtomicDiffEstimator( $\mathcal{X}_A^i$, $\mathcal{X}_B^i$, $\hat{u}$)
4.     if ( atomicEstimate $\neq$ noEstimate) then
5.        sum := sum + atomicEstimate;  count := count +1
6.     endif
7.  endfor
8.  return( sum $\times$ $\hat{u}$ / count )
end

procedure AtomicDiffEstimator( $\mathcal{X}_A^i$, $\mathcal{X}_B^i$, $\hat{u}$, $\epsilon$ )
begin
1.  index := $\lceil \log(\frac{\beta \cdot \hat{u}}{1-\epsilon}) \rceil$        // $\beta$ is constant > 1 (see analysis)
2.  if ( not SingletonUnionBucket($\mathcal{X}_A^i$, $\mathcal{X}_B^i$, index) ) then
3.     return( noEstimate)
4.  estimate := 0
5.  if ( SingletonBucket($\mathcal{X}_A^i$, index) and
         ($\mathcal{X}_B^i$[ index, 1, 0] + $\mathcal{X}_B^i$[index, 1, 1] = 0) ) then
6.        estimate := 1          // found witness of $A - B$
7.  return( estimate )
end
```

Figure 6: Set-Difference Cardinality Estimator.

hash sketch synopses $\mathcal{X}_A$ and $\mathcal{X}_B$. Our set-difference algorithm assumes the existence of an $(\epsilon', \Theta(\delta))$-estimate $\hat{u}$ that approximates the cardinality of the union $u = |A \cup B|$ to within a relative error of $\epsilon' = \epsilon/3$ with probability at least $1 - \Theta(\delta)$. ($\hat{u}$ can be obtained using the $\mathcal{X}_A$ and $\mathcal{X}_B$ synopses, and the procedure described in Section 3.3.)

Our algorithm for estimating set difference over two update streams A and B (termed SetDifferenceEstimator) is depicted in Figure 6. Briefly, our algorithm uses averaging over r independent copies of 2-level hash sketch synopses built over A and B; each of the r copies using independently-chosen first- and second-level hash functions from $\mathcal{H}$ and $\mathcal{G}$. For each corresponding pair of 2-level hash sketches for A and B (which, of course, use the same hash functions), our basic difference-estimation procedure (termed AtomicDiffEstimator) is called to return an atomic estimate. The key idea in AtomicDiffEstimator is to try to discover a singleton first-level bucket in the pair $\mathcal{X}_A^i$ and $\mathcal{X}_B^i$ that contains a *"witness"* element for $A - B$. This is accomplished by selecting a first-level bucket at a level located slightly higher than $\log |A \cup B|$ (Steps 1-2), so that we actually find a singleton bucket for $A \cup B$ with constant probability; if the bucket is not a singleton, then we cannot use this pair of sketches in our set-difference estimation and a **noEstimate** flag is returned. Otherwise, we check to see whether the bucket contains a witness element for $A - B$ using the SingletonBucket procedure for the $\mathcal{X}_A^i$ bucket and a simple test to see if the $\mathcal{X}_B^i$ bucket is empty (Step 5). AtomicDiffEstimator returns an atomic estimate of 1 if it finds a witness singleton and 0 otherwise. SetDifferenceEstimator then simply averages all the valid (i.e., 0 or 1) atomic estimates and scales the result by the union estimate $\hat{u}$ to compute the final estimate for the set difference $|A - B|$.

Analysis. Consider the first-level bucket "index" chosen in Step 3 of our AtomicDiffEstimator procedure, and let $R = 2^{\text{index}+1}$. Note that, by our selection of $\hat{u}$, R is at least $\beta|A \cup B|$ with high probability. In this bucket, our procedure tries to discover a witness value for $A - B$ by checking the following condition.

Set-Difference Witness Condition: Bucket "index" is a non-empty singleton for A and empty for B, provided that bucket "index" is a singleton bucket for $A \cup B$.

Let p denote the (conditional) probability that the Set-Difference Witness Condition is true. Then, we can write:

$$p = \frac{\mathbf{Pr}\left[\text{"index" singleton for } A \text{ and empty for } B\right]}{\mathbf{Pr}\left[\text{"index" singleton for } A \cup B\ \right]}$$
$$= \frac{\frac{|A-B|}{R}\left(1 - \frac{1}{R}\right)^{|A\cup B|-1}}{\frac{|A\cup B|}{R}\left(1 - \frac{1}{R}\right)^{|A\cup B|-1}} = \frac{|A-B|}{|A \cup B|}.$$

To see this, note that the probability of any given element mapping to bucket "index" is $1/R$, so (by independence) the probability of a given element being the *single* element mapping to that bucket is exactly $\frac{1}{R}(1-1/R)^{|A\cup B|-1}$. Now, the number of elements that can give a singleton bucket for A and an empty bucket for B is exactly $|A - B|$, giving the numerator $\frac{|A-B|}{R}(1 - 1/R)^{|A \cup B|-1}$ (a similar argument applies for the denominator), and the derivation follows.

Our technique relies on using the 0/1 atomic estimates returned from the AtomicDiffEstimator procedure as independent "observations" of p and averaging them to obtain an estimate $\hat{p}$ of p. Since (as shown above) $p = |A-B|/|A \cup B|$, our final estimate for the set-difference size $|A-B|$ is $\hat{d} = \hat{p} \cdot \hat{u}$ (Step 8). Let r denote the number of independent 2-level hash sketch synopses maintained and r' be the number of independent observations of p used to obtain $\hat{p}$. Clearly, $r' \leq r$ since for some of our sketches the first-level "index" bucket is not a singleton and a **noEstimate** flag is returned; however, we can lower-bound the probability that a valid observation of p is obtained as follows:

$$\mathbf{Pr}\left[0/1 \text{ observation}\right] = \mathbf{Pr}\left[\text{"index" singleton for } A \cup B\right]$$
$$= \frac{|A \cup B|}{R}\left(1 - \frac{1}{R}\right)^{|A \cup B|-1}$$
$$> \frac{|A \cup B|}{R}\left(1 - \frac{|A \cup B|}{R}\right) > \frac{\beta - 1}{\beta^2},$$

where the first inequality follows from Bernoulli's inequality and the second inequality comes from the fact that $|A \cup B|/R < 1/\beta$. Then, we can simply apply Chernoff bounds to show that, with probability at least $1 - \Theta(\delta)$, for any constant $\epsilon_1 < 1$, the number of valid observations r' is going to be at least $(1 - \epsilon_1)\frac{\beta-1}{\beta^2}r$ as long as $r \geq \Theta(\frac{\log(1/\delta)\beta^2}{\epsilon_1^2(\beta-1)})$.

In order to produce an (ϵ, δ)-estimate $\hat{d}$ for the set difference $|A - B|$, we ensure that our $\hat{p}$ average determined in procedure SetDifferenceEstimator is an $(\epsilon/3, \Theta(\delta))$-estimate for p; that is:

$$\mathbf{Pr}\left[|\hat{p} - p| \leq \frac{\epsilon p}{3}\right] \geq 1 - \Theta(\delta).$$

By Chernoff bounds, the above inequality holds if $r'p \geq \Theta(\frac{\log(1/\delta)}{\epsilon^2})$ or, equivalently, $r' \geq \Theta(\frac{\log(1/\delta)|A \cup B|}{\epsilon^2 |A-B|})$. Assuming this condition is satisfied, we have:

$$|\hat{d} - |A - B|| = |\hat{p}\hat{u} - pu| \in |p(1 \pm \epsilon/3)u(1 \pm \epsilon/3)| \subseteq pu(1 \pm \epsilon),$$

since $\epsilon \leq 1$. Thus, we obtain an (ϵ, δ)-estimate for $|A - B|$ provided that the number of valid p observations r' satisfies $r' \geq \Theta(\frac{\log(1/\delta)|A \cup B|}{\epsilon^2 |A-B|})$ or the total number of independent 2-level hash sketches maintained is $r \geq \Theta(\frac{\log(1/\delta)\beta^2|A\cup B|}{\epsilon^2 \min\{\epsilon_1^2, 1-\epsilon_1\}(\beta-1)|A-B|})$ (since, as discussed above, $r' \geq (1-\epsilon_1)\frac{\beta-1}{\beta^2}r$ with high probability if $r \geq \Theta(\frac{\log(1/\delta)\beta^2}{\epsilon_1^2(\beta-1)})$). The optimal values for the constants ϵ_1 and β (i.e., the values minimizing the required number of independent sketch copies) can be easily determined from the above expression as $\epsilon_1 = (\sqrt{5}-1)/2$ and $\beta = 2$. Based on the above analysis, we can state the following theorem.

THEOREM 3.4. *Procedure* SetDifferenceEstimator *returns an (ϵ, δ)-estimate for the size of the set difference $|A - B|$ of two update streams A and B using 2-level hash sketch synopses with a total storage requirement of*

$$\Theta\left(\frac{\log(1/\delta)|A \cup B|}{\epsilon^2|A-B|}\log M \log N \log(\frac{\log(1/\delta)M}{\epsilon^2\delta})\right).$$

∎

Proof: Follows easily from the above analysis. The $\Theta(\log M \log N \log(\frac{\log M}{\delta}))$ term denotes the size of each 2-level hash sketch synopsis maintained by our algorithm. Note that, in order to guarantee a confidence of $1-\delta$ for the final estimate, each of the possible basic property checks done over the chosen level for each of our r maintained 2-level hash sketch synopses has to have a probability of failure $\leq \Theta(\frac{\delta}{r})$ (by the union bound). Since $r = \Theta(\frac{\log(1/\delta)|A\cup B|}{\epsilon^2|A-B|}) \leq \Theta(\frac{\log(1/\delta)M}{\epsilon^2})$, the number of second-level hash buckets (and counters) required is $s = \Theta(\log(\frac{\log(1/\delta)M}{\epsilon^2\delta}))$. ∎

3.5 The Set-Intersection Estimator

Given two continuous update streams A and B, the cardinality of the set intersection of streams A and B (i.e., $|A \cap B|$) is defined as the number of distinct data elements whose net frequency is positive in both A and B. The structure of our set-intersection estimator (termed SetIntersectionEstimator) for A and B based on their 2-level hash sketch synopses $\mathcal{X}_A$ and $\mathcal{X}_B$ is basically identical to that of the SetDifferenceEstimator procedure depicted in Figure 6. The only difference is that, since we are now looking for "witness" elements for the intersection $A \cap B$, the **if**-condition in Step 5 of procedure AtomicDiffEstimator is changed to: "(SingletonBucket($\mathcal{X}_A^i$, index) **and** SingletonBucket($\mathcal{X}_B^i$, index))", to obtain the corresponding atomic set-intersection estimation algorithm AtomicIntersectEstimator. The following theorem can then be shown using an analysis similar to that of Section 3.4.

THEOREM 3.5. *Procedure* SetIntersectionEstimator *returns an (ϵ, δ)-estimate for the size of the set intersection $|A \cap B|$ of two update streams A and B using 2-level hash sketch synopses with a total storage requirement of*

$$\Theta\left(\frac{\log(1/\delta)|A \cup B|}{\epsilon^2|A\cap B|}\log M \log N \log(\frac{\log(1/\delta)M}{\epsilon^2\delta})\right).$$

∎

3.6 Extension to Limited Independence

Thus far, the analysis of our set-operation estimators has made the (unrealistic) assumption that the first-level hash functions used in our 2-level hash sketch synopses guarantee fully (i.e., M-wise) independent value mappings. (Second-level hash mappings only require pair-wise independence (Lemma 3.1).) In this section, we present a series of statistical lemmas that allow the analysis of our (ϵ, δ) set-operation estimators to be extended to the much more realistic setting of t-wise independent first-level hashing, where $t = \Theta(\log\frac{1}{\epsilon})$. Note that maintaining these first-level hash functions implies an additive storage cost of $O(\log\frac{1}{\epsilon}\log M)$ per 2-level hash sketch for storing an appropriate seed (e.g., [3, 18]). This cost can be factored in the equations of Theorems 3.3-3.5 by simply adding a $\log\frac{1}{\epsilon}$ multiplicative factor.

The only place in our analysis where the assumption of full independence is used is in deriving the closed-form expression for the probability of the conditions checked through our 2-level hash sketch synopses (e.g., the "Set-Difference Witness Condition" for set difference, or the non-empty bucket condition for set union). Our results below demonstrate that these (fully-independent) probabilities are actually estimated to within small relative error if only t-wise independence is assumed with $t = \Theta(\log\frac{1}{\epsilon})$. Assume that we have fixed a first-level bucket j and let $1/R = 1/2^{j+1}$ denote the probability that an element in $[M]$ maps to bucket j. We use i-subscripted small letters (e.g., x_i, y_i) to denote the Boolean random variables for the simple event "the i^{th} distinct value in a stream maps to bucket j". Throughout this section, we use $\mathbf{Pr}[]$ ($\mathbf{Pr_t}[]$) to denote the probability function under full (resp., t-wise) independence of these Boolean random variables. (We omit the proofs of these statistical results since they are fairly long and do not offer much in terms of understanding; similar results for limited independence variables have appeared elsewhere, e.g., [18].)

LEMMA 3.6. *Let $X = \sum_{i=1}^m x_i$ be the sum of m Boolean random variables such that $E[x_i] = 1/R$, for $1 \leq i \leq m$. Then, $|\mathbf{Pr_t}[X \geq 1] - \mathbf{Pr}[X \geq 1]| \leq 2\binom{m}{t}(1/R)^t$.* ∎

COROLLARY 3.7. *Under the assumptions of Lemma 3.6, and if $t \geq \max\{3, \frac{\log(2/\epsilon)}{\log(R/m)}\}$ then,*

$$|\mathbf{Pr_t}[X \geq 1] - \mathbf{Pr}[X \geq 1]| \leq \epsilon \mathbf{Pr}[X \geq 1] \quad \text{and}$$
$$|\mathbf{Pr_t}[X = 0] - \mathbf{Pr}[X = 0]| \leq \epsilon \mathbf{Pr}[X = 0].$$

∎

Note that, with x_i's corresponding to the distinct elements of the union $A \cup B$, the condition $X \geq 1$ in Corollary 3.7 is essentially the set-union condition in Step 6 of procedure SetUnionEstimator (Figure 5) that checks for a non-empty bucket for $A \cup B$. Thus, Corollary 3.7 shows that the non-empty bucket fraction $\hat{p}$ assuming full independence (in the analysis of Section 3.3) is approximated to within a relative error of ϵ if only $\Theta(\log(1/\epsilon))$-independent hash functions are used. The corresponding result for the set-difference and set-intersection witness conditions is slightly more complicated and relies on the following statistical lemma (we omit the detailed constants to simplify the exposition).

LEMMA 3.8. *Let $X = \sum_{S_1} x_i$ and $Y = \sum_{S_2} y_j$ denote the sums of disjoint sets S_1, S_2 of Boolean random variables with $\mathbf{E}[x_i] = \mathbf{E}[y_i] = 1/R$ for each $x_i \in S_1$, $y_i \in S_2$, and* *let E denote the event $E := (X = 1 \wedge Y = 0 | X + Y = 1)$. If $t \geq \max\{4, \Theta(\log(1/\epsilon)\}$, then $|\mathbf{Pr_t}[E] - \mathbf{Pr}[E]| \leq \epsilon \mathbf{Pr}[E]$.* ▮

Again, it is easy to see that, with $S_1 := A - B$ and $S_2 := B$, the event $E := (X = 1 \wedge Y = 0 | X + Y = 1)$ in Lemma 3.8 is exactly the set-difference witness condition described in the analysis of our SetDifferenceEstimator estimator in Section 3.4. Similarly, with $S_1 := A \cap B$ and $S_2 := (A \cup B) - S_1$, E gives the corresponding condition for set intersection (Section 3.5). Thus, Lemma 3.8 that the witness condition probability estimate $\hat{p}$ assuming full independence (see analysis in Section 3.4) is estimated to within a relative error of ϵ using only $\Theta(\log(1/\epsilon))$-wise independence.

An interesting question, of course, is how this additional level of approximation affects the storage requirements of our estimators. We now demonstrate that the effect is bounded by a small constant factor. Consider the case of set difference and let $\hat{p}_t$ denote the probability of a set-difference witness under t-wise independence with $t = \Theta(\log(3/\epsilon))$. Our final estimate is $\hat{p}_t\hat{u}$ and we would like to guarantee that it is within $(1 \pm \epsilon)pu$. A simple application of the triangle inequality gives:

$$|\hat{p}_t\hat{u} - pu| \;\leq\; |\hat{p}_t\hat{u} - \hat{p}\hat{u}| + |\hat{p}\hat{u} - pu| \;\leq\; \frac{\epsilon}{3}\hat{p}\hat{u} + |\hat{p}\hat{u} - pu|$$

and, assuming that $\hat{p}\hat{u}$ (the estimate under full independence) approximates pu to within a relative error of $\epsilon/3$, we have:

$$|\hat{p}_t\hat{u} - pu| \;\leq\; \frac{\epsilon}{3}(1 + \frac{\epsilon}{3})pu + \frac{\epsilon}{3}pu \leq \epsilon pu,$$

for any $\epsilon < 1$. Thus, simply tightening our relative-error requirement to $\epsilon' = \epsilon/3$ (with the corresponding increase in our earlier storage-cost expressions) is sufficient to guarantee a relative error of ϵ for the final set-difference estimate with only $\Theta(\log(3/\epsilon))$-wise independence. Very similar derivations can also be given for our set union and intersection estimators under limited independence.

3.7 Lower Bounds

At this point, it is interesting to ask how good our randomized estimators for set operators really are – is it possible to design new estimation procedures that significantly improve on the space requirements stated in Theorems 3.3–3.5? In this section, we answer this question in the negative by demonstrating space lower bounds for set-operation estimators working in the streaming model.

The space requirements of our set-union estimator actually match those of earlier algorithms for set union over insertion streams (see, for example, [2, 4, 15]; this is, of course, modulo the $O(\log N)$ factor, since our algorithms need to maintain counters for dealing with deletions in the stream. Our SetUnionEstimator space requirements also match (to within log and constant factors) the lower bounds shown by Alon et al. [2] on the space needed by any randomized algorithm for estimating the number of distinct values in a data stream. As evidenced in the space bounds of Theorems 3.4–3.5, estimating set difference and intersection is a significantly more difficult problem than that for union; essentially, our results show that, with limited space, our estimators can only provide robust estimates for differences/intersections that are sufficiently large compared to the corresponding set union (i.e., $|A \cup B|$). (Similar observations have been made for estimators designed for the special case of *insert-only* streams [5, 11].) The following theorem proves a lower bound for all randomized approximation algorithms stating that the space requirements of our SetDifferenceEstimator and SetIntersectionEstimator estimators cannot be significantly improved (their space usage is within small polynomial and log factors of the optimal).

THEOREM 3.9. *Any randomized algorithm that, with high probability, estimates the set cardinality $|A\mathsf{op}B|$ ($\mathsf{op} \in \{-, \cap\}$) to within any constant relative error ϵ must use at least $\Theta(\frac{|A \cup B|}{\epsilon |A\mathsf{op}B|})$ bits.* ▮

Proof: Let $n = |A \cup B|$. Consider first the problem of estimating the set-intersection cardinality $|A \cap B|$. Determining the value of $|A \cap B|$ *exactly* with high probability is at least as hard as the well-known *SET-DISJOINTNESS* problem of (probabilistic) communication complexity, which requires at least $\Theta(n)$ bits of communication (i.e., space) [21, 20]. Assume now that we have a procedure $P(\epsilon)$ that estimates $|A \cap B|$ with high probability to within a relative error of ϵ using less than $o(\frac{n}{\epsilon|A \cap B|})$ bits; then, we will show that this implies an $o(n)$ solution for *SET-DISJOINTNESS*. More specifically, our algorithm for solving the *SET-DISJOINTNESS* problem for A and B is as follows. Pick any constant $\epsilon' < 1$ and run $P(\epsilon')$ to determine a high-probability estimate $\hat{t}$ of $|A \cap B|$. By our assumptions for $P()$, we know that this run will use only $o(\frac{n}{|A \cap B|})$ bits (remember that ϵ' is a constant) and, with high probability, $(1 - \epsilon')|A \cap B| < \hat{t} < (1 + \epsilon')|A \cap B|$. This last inequality also implies that, with high probability,

$$\frac{1 - \epsilon'}{2(1 + \epsilon')|A \cap B|} < \frac{1 - \epsilon'}{2\hat{t}} < \frac{1}{2|A \cap B|}. \qquad (1)$$

Now, run algorithm $P(\epsilon)$ again, this time with $\epsilon = \frac{1-\epsilon'}{2\hat{t}}$; since, $\frac{1-\epsilon'}{2\hat{t}} < \frac{1}{2|A \cap B|}$ it is easy to see that this run will estimate $|A \cap B|$ to within an *additive* error of less that 1/2, so it essentially allows us to estimate $|A \cap B|$ *exactly* (with high probability). Furthermore, the space used by $P(\epsilon)$ is only $o(\frac{n}{\epsilon|A \cap B|}) \leq o(\frac{2n(1+\epsilon')}{1-\epsilon'})$ (by Inequality (1)), which is obviously $o(n)$. Thus, we have a procedure for solving the *SET-DISJOINTNESS* problem using less than $o(n)$ bits; this is clearly a contradiction. The same argument also goes through for set difference, since $A - B$ is simply $A \cap \overline{B}$. ▮

4. PROCESSING SET EXPRESSIONS

In this section, we generalize the estimation techniques for individual set operators presented in Sections 3.3-3.5 to formulate an (ϵ, δ)-estimator for the cardinality of general set expressions over a collection of update streams A_i, $i = 1, \ldots, n$. Such set expressions are of the generic form $E := (((A_1\mathsf{op}_1 A_2)\mathsf{op}_2 A_3) \cdots A_n)$, where the connectives op_j denote the standard set operators, namely, union, intersection, and set difference (as an example, $E := A_4 - (A_3 \cap (A_2 \cup A_1)))$. Our goal is to estimate $|E|$, that is, the number of distinct elements with positive net frequency in the output of E using only a collection of independent small 2-level hash sketch synopses built over the A_i update streams (of course, as in the simple set-operator case, for a given sketch, we use the same first- and second-level hash functions across all A_i's).

Briefly, our general set-expression estimator follows along the lines of our set-difference and set-intersection algorithms. As in the SetDifferenceEstimator and SetUnionEstimator procedures, we assume a robust estimate $\hat{u}$ for the union cardinality $u = |\cup_i A_i|$, where i ranges over the streams participating in our input set expression E, and uses $\hat{u}$ to select an appropriate first-level bucket index $j = \lceil \log(\frac{\beta \cdot \hat{u}}{1-\epsilon}) \rceil$, where β is a constant > 1. (This estimate $\hat{u}$ can be obtained from the synopses using our SetUnionEstimator procedure.) Our set-expression estimation algorithm for E starts by discarding all parallel 2-level hash sketch collections $\{\mathcal{X}_{A_1}, \mathcal{X}_{A_2}, \dots\}$ for which bucket j is not a singleton bucket for $\cup_i A_i$. (An easy generalization of our elementary check procedures in Section 3.2 can be used to determine this fact with high confidence.) Then, E is mapped to a Boolean expression $B(E)$ over the level-j buckets of the 2-level hash sketch synopses for A_i's; this expression is defined inductively as follows:

$E = A_i$ **:** Define $B(E) := (\mathcal{X}_{A_i}[j,1,0] + \mathcal{X}_{A_i}[j,1,1] > 0)$ (i.e., **true** iff bucket j is non-empty in $\mathcal{X}_{A_i}$).

$E = E_1 \cup E_2$ **:** Define $B(E) := B(E_1) \vee B(E_2)$ (i.e., the disjunction of the sub-expressions $B(E_1)$ and $B(E_2)$).

$E = E_1 \cap E_2$ **:** Define $B(E) := B(E_1) \wedge B(E_2)$ (i.e., the conjunction of the sub-expressions $B(E_1)$ and $B(E_2)$).

$E = E_1 - E_2$ **:** Define $B(E) := B(E_1) \wedge \overline{B(E_2)}$ (i.e., must satisfy $B(E_1)$ and not satisfy $B(E_2)$).

It is easy to see that, with the above methodology, our Boolean condition $B(E)$ for set expression E essentially corresponds to an "E Witness Condition" at the selected bucket index j, as defined below.

E **Witness Condition:** Bucket j is a non-empty singleton for the set expression E defined over $A_1, \dots, A_n$, provided that bucket j is a singleton bucket for $\cup_{i=1}^n A_i$.

As in our development for the SetDifferenceEstimator estimator, letting p_E denote the (conditional) probability that the E Witness Condition is true and $R = 2^{j+1}$, we have:

$$p_E = \frac{\mathbf{Pr}\left[\text{bucket } j \text{ non-empty singleton for } E\right]}{\mathbf{Pr}\left[\text{bucket } j \text{ singleton for } U = \cup_i A_i\ \right]}$$
$$= \frac{\frac{|E|}{R}\left(1 - \frac{1}{R}\right)^{|U|-1}}{\frac{|U|}{R}\left(1 - \frac{1}{R}\right)^{|U|-1}} = \frac{|E|}{|U|}.$$

An analysis similar to that in Section 3.4 can then be employed to demonstrate the following theorem.

Theorem 4.1. *The set-expression estimator described above returns an (ϵ, δ)-estimate for the cardinality of a set-expression $|E|$ over a collection of update streams $A_1, \dots, A_n$ using 2-level hash sketch synopses with a total storage requirement of*

$$\Theta\left(\frac{n \log(1/\delta) |\cup_i A_i|}{\epsilon^2 |E|} \log M \log N \log\left(\frac{n \log(1/\delta) M}{\epsilon^2 \delta}\right)\right).$$

∎

We can also easily extend the limited-independence analysis of Section 3.6 to show that our result for set-expression estimation holds under only $\Theta(\log(1/\epsilon))$-wise independent first-level hash functions for our sketches. (Once again, the cost for storing these functions can be factored in by simply adding a $\log(1/\epsilon)$ multiplicative factor in the expression of Theorem 4.1.)

We should note here that the technique presented in this section for dealing with the union operator in the context of larger set expressions is, in fact, different from the SetUnionEstimator procedure described in Section 3.3. Instead, the manner in which our set-expression estimator handles union essentially follows along the general paradigm of our set difference and intersection estimators (with an appropriately-defined "witness" condition – Sections 3.4-3.5). It is easy to see that both techniques basically have the same asymptotic storage requirements (remember that set union does not require second-level hashing). On the other hand, a detailed analysis shows that our more specialized SetUnionEstimator algorithm does have better (i.e., smaller) constants for set-union estimation which is, in general, much easier than the corresponding problem for set difference/intersection. The key benefit of the "witness"-based union algorithm is that (as shown in this section) it allows for a very clean, uniform algorithm for processing general set expressions.

5. EXPERIMENTAL STUDY

In this section, we present the results of a preliminary empirical study of our 2-level hash sketch synopses and set-expression estimators with several synthetic data sets. The objective of this study is to test the effectiveness of our novel stream-synopsis data structures and probabilistic estimation algorithms in practical data-streaming scenarios, and study their average-case behavior over several different problem instances. Our preliminary experimental results substantiate our theoretical claims, demonstrating the ability of our techniques to provide (with only limited space) accurate approximate answers to set-expression cardinality queries over continuous streaming data.

5.1 Testbed and Methodology

Methodology. In our experiments, we study the *accuracy* of the probabilistic set-expression cardinality estimation techniques developed in this paper using 2-level hash sketch synopses constructed over different synthetic data streams. The primary metric used to gauge the accuracy of our estimators is the conventional absolute relative error metric; that is, given an expression E and an estimate $\hat{e} = |E|$ of its cardinality, we define the error of the estimate as the ratio $\frac{|\hat{e} - |E||}{|E|}$. We perform experiments to measure the errors of our cardinality estimators as a function of the space made available for building 2-level hash sketch synopses for the input data streams. This accuracy/space tradeoff is studied over various input expressions, ranging from simple binary set operations (primarily difference and intersection) to more complex set expressions (over three or more streams). (Again, note that our techniques are the *first* to deal with set difference/intersection and set expressions over general update streams; thus, in a sense, comparing against the accurate answer is probably the best measure of effectiveness for our approach.)

To account for the probabilistic nature of our estimation algorithms, we run each experiment between $10 - 15$ times (with different random-seed values). The numbers used in our plots are averages of the observed relative error values after trimming away 30% of the highest relative errors for each experiment. We used this more robust, "trimmed-average" error metric to avoid the effects of outlier estimates (due to the variance of our randomized schemes) on the observed average-case behavior of our estimators.

Synthetic Data Generation. Our 2-level hash sketch synopses are impervious to delete operations, in the sense that a sketch obtained at the end of an update stream is identical to one that never sees the deleted items in the stream. Given this fact, our synthetic data generator produces *insert-only* streams for updating the 2-level hash sketch synopses for our estimation algorithms. Furthermore, since the accuracy of our cardinality estimates for a set expression E crucially depends on the ratio of the underlying set union to $|E|$ (Theorems 3.4, 3.5, 4.1), we generate our data streams in a controlled manner that allows us to vary this cardinality ratio and observe the behavior of our techniques for different settings. (We fix the size of the underlying set union, i.e., $|\cup_{A_i \in E} A_i|$, to $u \approx 2^{18}$ in all our experiments.)

We now describe the data-generation process for a binary set operation, say $A \cap B$, assuming a given target size e for the cardinality $|A \cap B|$. (We vary the value of e from $u/2$ down to $u/2^{10}$ in diminishing powers of 2.) In a first step, we generate 2^{18} 32-bit random unsigned integers and eliminate all duplicates (thus, the actual union size u can be slightly less than 2^{18}). Then, for each generated integer x, we insert x to either (a) both A and B, with probability e/u; or, (b) only A or only B, with equal probability $\frac{1-e/u}{2}$. Thus, at the end of this process, we expect to have approximately $\frac{e}{u}u = e$ elements in $A \cap B$, and about equal numbers of elements in both A and B. It is easy to devise a very similar controlled data-generation scheme for $A - B$.

For set expressions E involving multiple, say n, streams, our controlled data generation is slightly more complicated. Briefly, the main idea is to keep track of all $2^n - 1$ partitions in the Venn diagram of the underlying set union and give "assignment probabilities" to each partition such that the sum of probabilities for all partitions that comprise E is approximately $|E|/u$. (For simplicity, the probabilities are chosen so that all underlying sets have the same expected size.) Generated random integers are then assigned to these partitions as discussed above.

5.2 Experimental Results

We now present some of our preliminary experimental numbers for our probabilistic estimation algorithms. We focus our discussion here on three input set expressions: binary set intersection $A \cap B$, binary set difference $A - B$, and the more complex three-stream expression $(A - B) \cap C$. (We have observed qualitatively similar results for the estimation of other expressions.) We present plots that depict the (average) relative error behavior of our estimators as a function of the number of 2-level hash sketch synopses maintained on each data stream. A rough estimate for the number of bytes used by our synopses is given by multiplying the number of sketches with 32; since we are only considering insert-only streams, this estimate assumes simple bits (instead of counters) at each cell of our 2-level hash sketches. (The number of second-level hash functions used is kept fixed at 32.)

Figure 7(a) depicts the average (percentage) relative-error numbers for our set-intersection cardinality estimator as a function of the number of 2-level hash sketches used, and for three distinct values of the target intersection size $|A \cap B|$. The plots demonstrate the effectiveness and accuracy of our estimation algorithm. Even with as few as $128 - 256$ 2-level hash sketches, the error of our estimates is close to or below 20%, essentially across the range of the target intersection sizes tested. And, of course, increasing the number of sketches can lead to significant further reductions in the observed estimation error which finally drops to $\leq 10\%$ for 512 sketches.

Similar trends can also be observed for set-difference cardinality estimator in Figure 7(b). In this case, errors for smaller target difference sizes (i.e., $|A - B| = 8192$) are higher (about 48%) for small numbers of sketches. Once again, however, when our synopsis space reaches 512 sketches, all errors are in the area of 10% or lower. Note that, as predicted by our theoretical results, the quality of our estimates for a given number of sketches, in general, improves with higher target expression sizes. We do, of course, observe certain crossovers in the plots but they are to be expected given the variance of our randomized estimation techniques.

Finally, Figure 8 depicts the average relative-error plots for our set-expression cardinality estimator with the input expression $(A - B) \cap C$, for three different target expression sizes. The numbers clearly show trends that are very similar to those observed for the simpler binary set intersection/difference experiments. Once again, error numbers are fairly small even for moderate synopsis sizes, eventually tailing off to 20% or lower for 512 sketches. And, in accordance with our theoretical results (Theorem 4.1), larger target expression sizes imply better cardinality estimates (for a given synopsis size).

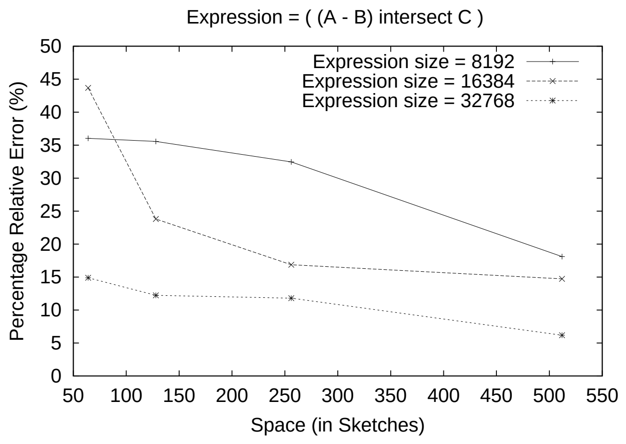
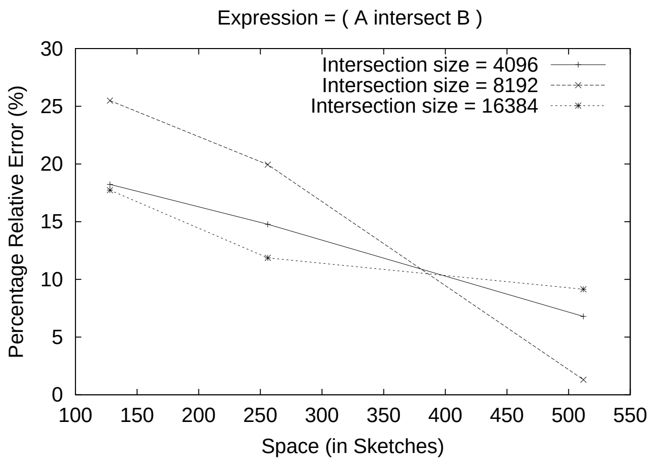


Figure 8: Average Relative Error for Estimating the Set-Expression Cardinality $|(A - B) \cap C|$.

6. CONCLUSIONS

Estimating the cardinality of set expressions defined over several (perhaps, distributed) continuous update streams is a fundamental class of queries that next-generation data-stream processing systems need to effectively support. In this paper, we have proposed the first space-efficient algorithmic solution for estimating the cardinality of full-fledged set expressions over general streams of updates (including item deletions as well as insertions). Our estimators rely



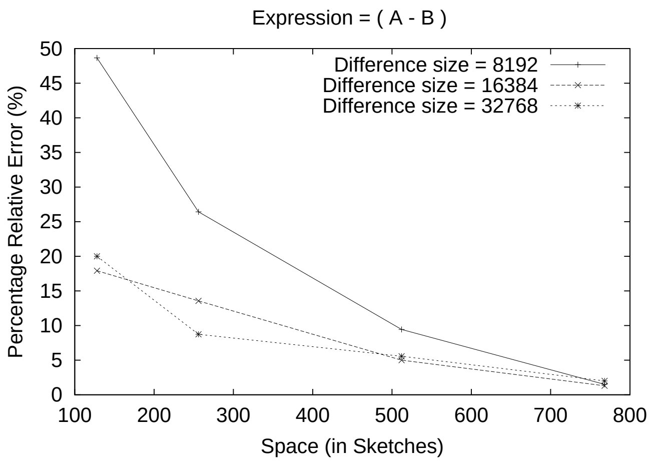


Figure 7: Average Relative Error for Estimating: (a) Set-Intersection Cardinality $|A \cap B|$; (b) Set-Difference Cardinality $|A - B|$.

on a novel, 2-level hash sketch synopsis data structure to provide low-error, high-confidence estimates for the cardinality of set expressions (including operators such as set union, intersection, and difference) over continuous update streams, using only small space and small processing time per update. Furthermore, unlike earlier approaches, our algorithms never require require rescanning or resampling of past stream items, regardless of the number of deletions in the stream. Preliminary results from an empirical study of our estimators have substantiated our theoretical claims, showing that our techniques can provide space-efficient and accurate set-expression cardinality estimates over streaming data.

7. REFERENCES

[1] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. "Tracking Join and Self-Join Sizes in Limited Storage". In *Proc. of the 18th ACM Symp. on Principles of Database Systems*, May 1999.

[2] N. Alon, Y. Matias, and M. Szegedy. "The Space Complexity of Approximating the Frequency Moments". In *Proc. of the 28th Annual ACM Symp. on the Theory of Computing*, May 1996.

[3] N. Alon and J. H. Spencer. *"The Probabilistic Method"*. John Wiley & Sons, Inc., 1992.

[4] Z. Bar-Yossef, T.S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. "Counting distinct elements in a data stream". In *Proc. of the RANDOM'2002 Intl. Workshop*, September 2002.

[5] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. "Min-wise independent permutations". In *Proc. 30th ACM Symp. on the Theory of Computing*, May 1998.

[6] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. "Towards Estimation Error Guarantees for Distinct Values". In *Proc. of the 19th ACM Symp. on Principles of Database Systems*, May 2000.

[7] Z. Chen, F. Korn, N. Koudas, and S. Muthukrishnan. "Selectivity Estimation For Boolean Queries". In *Proc. of the 19th ACM Symp. on Principles of Database Systems*, May 2000.

[8] E. Cohen. "Size-estimation Framework with Applications to Transitive Closure and Reachability". *Journal of Computer and Systems Sciences*, 55(3):441–453, December 1997.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *"Introduction to Algorithms"*. MIT Press (The MIT Electrical Engineering and Computer Science Series), 1990.

[10] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. "Processing Complex Aggregate Queries over Data Streams". In *Proc. of the 2002 ACM SIGMOD Intl. Conf. on Management of Data*, June 2002.

[11] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. "An approximate L^1-difference algorithm for massive data streams". In *Proc. 40th IEEE Symp. on Foundations of Computer Science*, October 1999.

[12] P. Flajolet and G. N. Martin. "Probabilistic Counting Algorithms for Data Base Applications". *Journal of Computer and Systems Sciences*, 31:182–209, 1985.

[13] M. Garofalakis, J. Gehrke, and R. Rastogi. "Querying and Mining Data Streams: You Only Get One Look". Tutorial in *28th Intl. Conf. on Very Large Data Bases*, August 2002.

[14] P. B. Gibbons. "Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports". In *Proc. of the 27th Intl. Conf. on Very Large Data Bases*, September 2001.

[15] P. B. Gibbons and S. Tirthapura. "Estimating Simple Functions on the Union of Data Streams". In *Proceedings of the Thirteenth Annual ACM Symp. on Parallel Algorithms and Architectures*, July 2001.

[16] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. "How to Summarize the Universe: Dynamic Maintenance of Quantiles". In *Proc. of the 28th Intl. Conf. on Very Large Data Bases*, August 2002.

[17] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. "Sampling-based estimation of the number of distinct values of an attribute". In *Proc. 21st Intl. Conf. on Very Large Data Bases*, September 1995.

[18] P. Indyk. "A Small Approximately Min-wise Independent Family of Hash Functions". In *Proc. of the 10th Annual ACM-SIAM Symp. on Discrete Algorithms*, January 1999.

[19] P. Indyk. "Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computation". In *Proc. of the 41st Annual IEEE Symp. on Foundations of Computer Science*, November 2000.

[20] B. Kalyanasundaram and G. Schnitger. "The Probabilistic Communication Complexity of Set Intersection". *SIAM Journal on Discrete Mathematics*, 5(4):545–557, Nov. 1992.

[21] E. Kushilevitz and N. Nisan. *"Communication Complexity"*. Cambridge University Press, 1997.

[22] J. Melton and A. R. Simon. *"Understanding the New SQL: A Complete Guide"*. Morgan Kaufmann Publishers, 1993.