Join-Distinct Aggregate Estimation over Update Streams

Sumit Ganguly*
IIT Kanpur
sganguly@cse.iitk.ac.in

Minos Garofalakis
Bell Laboratories
minos@bell-labs.com

Amit Kumar*
IIT Delhi
amitk@cse.iitd.ernet.in

Rajeev Rastogi
Bell Laboratories
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. Providing (perhaps approximate) answers to queries over such streams is a crucial requirement for many application environments; examples include large IP network installations where performance data from different parts of the network needs to be continuously collected and analyzed.

The ability to estimate the number of distinct (sub)tuples in the result of a join operation correlating two data streams (i.e., the cardinality of a projection with duplicate elimination over a join) is an important requirement for several data-analysis scenarios. For instance, to enable real-time traffic analysis and load balancing, a network-monitoring application may need to estimate the number of distinct `(source, destination)` IP-address pairs occurring in the stream of IP packets observed by router R_1, where the `source` address is also seen in packets routed through a different router R_2. Earlier work has presented solutions to the individual problems of distinct counting and join-size estimation (without duplicate elimination) over streams. These solutions, however, are fundamentally different and extending or combining them to handle our more complex "`Join-Distinct`" estimation problem is far from obvious. In this paper, we propose the *first* space-efficient algorithmic solution to the general `Join-Distinct` estimation problem over continuous data streams (our techniques can actually handle general *update streams* comprising tuple deletions as well as insertions). Our estimators are probabilistic in nature and rely on novel algorithms for building and combining a new class of hash-based synopses (termed *"*`JD` *sketches"*) for individual update streams. We demonstrate that our algorithms can provide low error, high-confidence `Join-Distinct` estimates using only small space and small processing time per update. In fact, we present lower bounds showing that the space usage of our estimators is within small factors of the best possible for the `Join-Distinct` problem. 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 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, 6, 11]. 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. 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 [11]. 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 join or multi-join aggregates [1, 6], norm and quantile estimation [2, 14, 16], or histogram and wavelet computation [13, 12]. Estimating the number of distinct (sub)tuples in the result of an equi-join operation correlating two update streams (i.e., the cardinality of a projection with duplicate elimination over a join) is one of the fundamental queries of interest for several data-analysis scenarios. As an example, a network-management application monitoring active IP-sessions may wish to correlate the active sessions at routers R_1 and R_2 by posing a query such as: "What is the number of distinct `(source, destination)` IP-address pairs seen in pack-

[*] Work done while the authors were with Bell Labs.

[1] An item modification is simply seen as a deletion directly followed by an insertion of the modified item.

PODS 2005 June 13-15, 2005, Baltimore, Maryland.
Copyright 2005 ACM 1-59593-062-0/05/06 . . . $5.00.

ets routed through R_1 such that the `source` address is also seen in packets routed by R_2?". Such a query would be used, for example, when trying to determine the load (i.e., number of distinct source-destination sessions) imposed on a core router R_1 by the set of customers connected to a specific router R_2 at the edge of the network. The result is simply the number of distinct tuples in the output of the project-join query $\pi_{\texttt{sour}_1,\texttt{dest}_1}(R_1(\texttt{sour}_1,\ \texttt{dest}_1) \bowtie_{\texttt{sour}_1=\texttt{sour}_2} R_2(\texttt{sour}_2,\ \texttt{dest}_2))$, where $R_i(\texttt{sour}_i, \texttt{dest}_i)$ denotes the multiset of source-destination address pairs observed in the packet stream through router R_i. The ability to provide effective estimates for such "`Join-Distinct`" query aggregates over the observed IP-session data streams 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. `Join-Distinct` queries are also an integral part of query languages for relational database systems (e.g., the `DISTINCT` clause in the SQL standard [20]). Thus, one-pass synopses for effectively estimating `Join-Distinct` aggregates can be extremely useful, e.g., in the optimization of such queries over Terabyte relational databases [10].

Prior Work. Estimating the number of distinct values in one pass over a data set is a very basic problem with several practical applications (e.g., query optimization); as a result, several solutions have been proposed for the simple distinct-count estimation problem over data streams. In their influential paper, Flajolet and Martin [7] propose an effective randomized estimator for distinct-value counting that relies on a hash-based synopsis data structure. 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 Flajolet-Martin estimator that relies solely on simple, linear hash functions. Several estimators based on uniform random sampling have also been proposed for distinct-element counting [4, 15]; however, such sampling-based approaches are known to be inaccurate and substantial negative results have been shown by Charikar et al. [4] 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. [10, 11] have proposed specialized sampling schemes specifically designed for distinct-element counting; their sampling schemes rely on hashing ideas (similar to [7]) to obtain a random sample *of the distinct elements* that is then used for estimation. Bar-Yossef et al. [3] also propose improved distinct-count estimators that combine new techniques and ideas from [2, 7, 11].

In our recent work [9], we have proposed novel, small-space synopsis data structures, termed 2-level hash sketches, and associated estimation algorithms for estimating the cardinality of *general set expressions* (including operators like set intersection or difference) over update streams. (Distinct-element counting can be seen as cardinality estimation for simple set unions.) Our results demonstrate that our 2-level hash sketch estimators require *near-optimal* space for approximating set-expression cardinalities and, unlike earlier approaches, they can handle general update streams without ever requiring access to past stream items. (In contrast, sampling-based solutions like [10, 11] may very well need to rescan and resample past stream items when deletions cause the maintained sample to be depleted; this is clearly an unrealistic requirement in a data-streaming environment.)

The problem of computing *non-distinct* aggregates over join or multi-join stream queries has also received a fair amount of attention recently. Alon et al. [1, 2] have proposed provably-accurate probabilistic techniques for tracking self-join and binary-join sizes (i.e., the total number of tuples in the join result) over update streams. Dobra et al. [6] extend their techniques and theoretical results to multi-join queries and general aggregate functions. The estimation algorithms described in [1, 2, 6] are fundamentally different from the above-cited techniques for distinct-count estimation, and rely on synopses obtained through *pseudo-random, linear projections* of the underlying frequency-distribution vectors. Thus, a direct combination of the two families of techniques to possibly obtain a solution to our `Join-Distinct` estimation problem appears to be very difficult, if not impossible.

Our Contributions. In this paper, we propose the *first* space-efficient algorithmic solution to the general `Join-Distinct` cardinality estimation problem over continuous update streams. Our proposed estimators are probabilistic in nature and rely on novel algorithms for building and combining a new class of hash-based synopses, termed *"`JD` sketches"*, for individual update streams. We present novel estimation algorithms that use our `JD` sketch synopses to provide low error, high-confidence `Join-Distinct` estimates using only small space and small processing time per update. We also present lower bounds showing that the space usage of our estimators is within small factors of the best possible for any solution to our `Join-Distinct` estimation problem. More concretely, our key contributions can be summarized as follows.

• **New JD Sketch Synopsis Structures for Update Streams.** We formally introduce the `JD` sketch synopsis data structure and describe its maintenance over a continuous stream of updates (rendering a multi-set of data tuples). In a nutshell, our `JD` synopses make use of our earlier 2-level hash sketch structures while imposing an additional level of hashing that is needed in order to effectively project and count on the attributes in the distinct-count clause. Our `JD` sketch synopses are built independently on each of the individual streams to be joined, and never require rescanning or resampling of past stream items, regardless of the deletions in the stream: at any point in time, our synopsis structure is guaranteed to be *identical* to that obtained if the deleted items had never occurred in the stream.

• **Novel, Streaming Join-Distinct Estimation Algorithms.** Based on our `JD` sketch synopses, we propose novel probabilistic algorithms for estimating `Join-Distinct` cardinalities over update streams. A key element of our solution is a new technique for intelligently *composing* independently-built `JD` sketches (on different streams) to obtain an estimate for the cardinality of their `Join-Distinct` result. We present a detailed analysis of the space requirements for our estimators, as well as a lower bound result showing that the space requirements of our estimators is actually within small constant and log factors of the optimal. We also discuss how our techniques can be extended to handle other aggregates (e.g., predicate selectivities) over the results of `Join-Distinct` queries. Once again, ours is the first approach to solve these difficult estimation problems in the data- or update-streaming model. Furthermore, even though we present our estimators in a single-site setting, our solutions also naturally extend to more general *distributed-streams* models [11].

- **Experimental Results Validating our Approach.** We present preliminary results from an experimental study with different synthetic data sets that verify the effectiveness of our `JD` 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 `Join-Distinct` cardinality queries over continuous streaming data.

2. PRELIMINARIES

In this section, we discuss the basic elements of our update-stream processing architecture, state our `Join-Distinct` estimation problem, and introduce some key concepts and ideas, including our recently-proposed 2-level hash sketch synopses [9] which play an integral role in our `Join-Distinct` synopses and estimation procedures.

2.1 Stream Model and Problem Formulation

The key elements of our update-stream processing architecture for `Join-Distinct` cardinality estimation are depicted in Figure 1; similar architectures for processing data streams have been described elsewhere (see, for example, [6, 9, 13]).

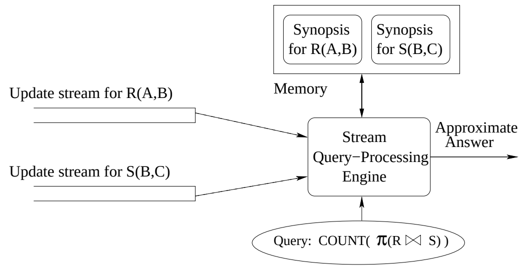


Figure 1: Update-Stream Processing Architecture.

Each input stream renders a multi-set of relational tuples (`R(A,B)` or `S(B,C)`) as a continuous stream of updates. Note that, in general, even though A, B, and C are denoted as single attributes in our model, they can in fact denote *sets of attributes* in the underlying relational schema. Furthermore, without loss of generality, we assume that each attribute $X \in \{A, B, C\}$ takes values from the integer domain $[M_X] = \{0, \ldots, M_X - 1\}$. Each streaming update (say, for input `R(A,B)`) is a pair of the form $< (a, b), \pm v >$, where $(a, b) \in [M_A] \times [M_B]$ denotes the specific tuple of `R(A,B)` whose frequency changes, and $\pm v$ is the net change in the frequency of (a, b) in `R(A,B)`, i.e., "$+v$" ("$-v$") denotes v insertions (resp., deletions) of tuple (a, b). (Note that handling deletions substantially enriches our streaming model; for instance, it allows us to deal with estimation over *sliding windows* of the streams by simply issuing implicit delete operations for expired stream items no longer in the window of interest.) We also let N denote an upper bound on the total number of data tuples (i.e., the net sum of tuple frequencies) in either `R(A,B)` or `S(B,C)`. In contrast to conventional DBMS processing, our stream processor is allowed to see the update tuples for each relational input *only once*, 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.

Our focus in this paper is on the difficult problem of estimating the number of *distinct* (A, C) (sub)tuples in the result of the data-stream join `R(A,B)`$\bowtie_B$ `S(B,C)`. More formally, we are interested in approximating the result of the query `Q`$= |\pi_{A,C}($`R(A,B)` $\bowtie$ `S(B,C)`$)|$ or, using SQL,[2]

```
Q =  SELECT COUNT DISTINCT (A, C)
     FROM R(A,B), S(B,C)
     WHERE R.B = S.B
```

(We use $|X|$ to denote the set cardinality, i.e., number of distinct elements with positive net frequency, in the multiset X.) Note that, in general, the attribute sets A, B, and C in `Q` are not necessarily disjoint or non-empty. For example, the target attributes A and C may in fact contain the join attribute B, and either A or C can be empty (i.e., a *one-sided* projection). To simplify the exposition, we develop our estimation algorithms assuming both A and C are non-empty and disjoint from B, i.e., $A, C \neq \phi$ and $A \cap B = B \cap C = \phi$. Then, in Section 5 we discuss how our techniques can handle other forms of `Join-Distinct` aggregate estimation. In a slight abuse of notation, we will use A as a shorthand for the set of distinct A-values seen in `R(A,B)`, and $|A|$ to denote the corresponding set cardinality, i.e., $|A| = |\pi_A($`R(A,B)`$)|$. (B, C and $|B|$, $|C|$ are used similarly, with B being the distinct B-values seen in either `R(A,B)` or `S(B,C)`, i.e., the union $\pi_B($`R(A,B)`$) \cup \pi_B($`S(B,C)`$)$.)

Our stream-processing engine is allowed a bounded 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. The key constraints imposed on such synopses are that: (1) they are much smaller than the number of elements in the streams (e.g., their size is polylogarithmic in $|$`R(A,B)`$|$, $|$`S(B,C)`$|$); and, (2) they can be easily maintained, during a single pass over the streaming update tuples in the (arbitrary) order of their arrival. At any point in time, our estimation algorithms can combine the maintained synopses to produce an estimate for `Q`.

Even for the simpler case of insert-only streams, communication-complexity arguments can be applied to show that the *exact computation* of `Q` requires at least $\Omega(M_B)$ space[3], even for randomized algorithms [18, 17] (the problem is at least as hard as the SET-DISJOINTNESS problem for B values). Instead, our focus is to *approximate* the count `Q` to within a small relative error, with high confidence. Thus, we seek to obtain a (randomized) (ϵ, δ)-approximation scheme [2, 11], that computes an estimate $\hat{\texttt{Q}}$ of `Q` such that $\mathbf{Pr}\left[|\hat{\texttt{Q}} - \texttt{Q}| \leq \epsilon\texttt{Q}\right] \geq 1 - \delta$.

2.2 The 2-level Hash Sketch Stream Synopsis

Flajolet and Martin [7] were the first to propose the use of hash-based techniques for estimating the number of distinct elements (i.e., $|A|$) over an (insert-only) data stream

[2] It is interesting to note here that, yet another possible application of our techniques is to estimate the number of *distinct vertex pairs at distance* 2 over the stream of edges of a massive graph (e.g., Internet-connectivity or Web-linkage data).

[3] 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* $g(n) = O(f(n))$ [5].

A. Briefly, assuming that the elements of A range over the data domain $[M]$, the Flajolet-Martin (FM) algorithm relies on a family of hash functions $\mathcal{H}$ that map incoming elements uniformly and independently over the collection of binary strings in $[M]$. 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 $\mathbf{Pr}[\textsc{lsb}(h(i)) = l] = \frac{1}{2^{l+1}}$.[4] 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 bitmap of size $\Theta(\log M)$. This bitmap is initialized to all zeros and, for each incoming value i in the input (multi-)set A, the bit located at position $\textsc{lsb}(h(i))$ is turned on. The key idea behind the FM technique is that, 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 to map to location l in each synopsis; thus, we expect $|A|/2$ values to map to bit 0, $|A|/4$ to map to bit 1, and so on. Therefore, the location of the leftmost zero (say λ) in a bitmap synopsis is a good indicator of $\log |A|$, or, $2^{\lambda} \approx |A|$. Of course, to boost accuracy and confidence, the FM algorithm employs averaging over several independent instances (i.e., choices of the mapping hash-function $h \in \mathcal{H}$ and corresponding bitmap synopses).

In our earlier work [9], we have proposed a generalization of the basic FM bitmap hash synopsis, termed 2-level hash sketch, that enables accurate, small-space cardinality estimation for *arbitrary set expressions* (e.g., including set difference, intersection, and union operators) defined over a collection of general *update streams* (ranging over the domain $[M]$). 2-level hash sketch synopses rely on a family of (first-level) hash functions $\mathcal{H}$ that uniformly randomize input values over the data domain $[M]$; then, for each domain partition created by first-level hashing, a small (logarithmic-size) *count signature* is maintained for the corresponding multi-set of stream elements.

More specifically, a 2-level hash sketch uses one randomly-chosen first-level hash function $h \in \mathcal{H}$ that, as in the FM algorithm, is combined with the $\textsc{lsb}$ operator to map the domain elements in $[M]$ onto a logarithmic range $\{0, \ldots, \Theta(\log M)\}$ of first-level buckets with exponentially-decreasing probabilities. Then, for the collection of elements mapping to a given first-level bucket, a count signature comprising an array of $\log M + 1$ element counters is maintained. This count-signature array consists of two parts: (a) one *total element count*, which tracks the net total number of elements that map onto the bucket; and, (b) $\log M$ *bit-location counts*, which track, for each $l = 1, \ldots, \log M$, the net total number of elements e with $\textsc{bit}_l(e) = 1$ that map onto the bucket (where, $\textsc{bit}_l(e)$ denotes the value of the l^{th} bit in the binary representation of $e \in [M]$). Conceptually, a 2-level hash sketch for a streaming multi-set A can be seen as a two-dimensional array $\mathcal{S}_A$ of size $\Theta(\log M) \times (\log M + 1) = \Theta(\log^2 M)$, where each entry $\mathcal{S}_A[k, l]$ is a data-element counter of size $O(\log N)$ corresponding to the l^{th} count-signature location of the k^{th} first-level hash bucket. By convention, given a bucket k, $\mathcal{S}_A[k, 0]$ denotes the total element count, whereas the bit-location counts are located at $\mathcal{S}_A[k, 1], \ldots, \mathcal{S}_A[k, \log M]$. The structure of our 2-level hash sketch synopses is pictorially depicted in Figure 2.

Maintenance. The algorithm for maintaining a 2-level hash sketch synopsis $\mathcal{S}_A$ over a stream of updates to a multi-set A is fairly simple. The sketch structure is first initialized to all zeros and, for each incoming update $< e, \pm v >$, the element counters at the appropriate locations of the $\mathcal{S}_A$ sketch are updated; that is, we simply set $\mathcal{S}_A[\textsc{lsb}(h(e)), 0] := \mathcal{S}_A[\textsc{lsb}(h(e)), 0] \pm v$ to update the total element count in e's bucket and, for each $l = 1, \ldots, \log M$ *such that* $\textsc{bit}_l(e) = 1$, we set $\mathcal{S}_A[\textsc{lsb}(h(e)), l] := \mathcal{S}_A[\textsc{lsb}(h(e)), l] \pm v$ to update the corresponding bit-location counts. 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* to a sketch that never sees the deleted items in the stream.

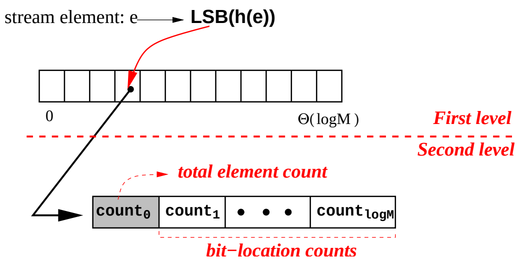


Figure 2: The 2-level Hash Sketch Synopsis Structure.

Estimation. Despite their apparent simplicity, 2-level hash sketches turn out to be a powerful stream-synopsis structure. As shown in our earlier work [9], they can serve as the basis for probabilistic algorithms that accurately estimate the cardinality of general set expressions over rapid update streams while using only small (in fact, *provably near-optimal*) space. Briefly, all these set-expression estimators rely on constructing several (say, s_1) independent instances of *parallel* 2-level hash sketch synopses (using the same first-level hash functions) over the input streams, and using them to infer a low-error, high-probability cardinality estimate [9]. The following theorem summarizes the quality guarantees for the 2-level hash sketch set-intersection estimator in [9] (termed IntersectionEstimator in this paper); similar results have been shown for other set operators (like, set union and difference), as well as general set expressions [9].

Theorem 2.1. *([9]) : Algorithm* IntersectionEstimator *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* $\Theta(s_1 \log^2 M \log N)$, *where* $s_1 = \Theta\left(\frac{\log(1/\delta)|A \cup B|}{\epsilon^2 |A \cap B|}\right)$. ∎

3. JOIN-DISTINCT SYNOPSIS STRUCTURE AND ESTIMATION ALGORITHM

In this section, we define our basic `Join-Distinct` synopses (termed `JD` sketches) and describe the algorithm for maintaining `JD` sketches over a stream of updates. We then present our `Join-Distinct` estimation algorithm that, abstractly, combines information from individual `JD` sketch synopses to produce a theoretically-sound, guaranteed-error estimate of the `Join-Distinct` cardinality.

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

3.1 Our Stream Synopsis: JD Sketches

Briefly, our proposed JD sketch synopsis data structure for update stream R(A,B) uses hashing on attribute(s) A (similar to the basic FM distinct-count estimator) and, for each hash bucket of A, a family of 2-level hash sketches is deployed as a concise synopsis of the B values corresponding to tuples mapped to this A-bucket. More specifically, a JD sketch synopsis $\mathcal{X}_{A,B}$ for stream R(A,B) relies on a hash function h_A selected at random from an appropriate family of randomizing hash functions $\mathcal{H}_A$ that uniformly randomize values over the domain $[M_A]$ of A. As in the FM algorithm (and 2-level hash sketches), this hash function h_A is used in conjunction with the LSB operator to map A-values onto a logarithmic number of hash buckets $\{0, \ldots, \Theta(\log M_A)\}$ with exponentially-decreasing probabilities. Each such bucket $\mathcal{X}_{A,B}[i]$ is an array of s_1 independent 2-level hash sketches built on the (multi-)set of B values for (A, B) tuples whose A component maps to bucket i. Let $\mathcal{X}_{A,B}[i, j]$ $(1 \leq j \leq s_1)$ denote the j^{th} 2-level hash sketch on B for the i^{th} A bucket – a crucial point in the JD sketch definition is that the B hash functions (h_B) used by the j^{th} 2-level hash sketch in $\mathcal{X}_{A,B}$ are *identical* across all A buckets; in other words, $\mathcal{X}_{A,B}[i_1, j]$ and $\mathcal{X}_{A,B}[i_2, j]$ use exactly the same hash functions on B for any i_1, i_2 in $\{0, \ldots, \Theta(\log M_A)\}$.

Conceptually, a JD sketch $\mathcal{X}_{A,B}$ for the update stream R(A,B) can be seen as a four-dimensional array of total size $\Theta(\log M_A)\times$ $s_1\times$ $\Theta(\log M_B)\times$ $(\log M_B + 1) = s_1\cdot$ $\Theta(\log M_A$ $\log^2 M_B)$, where each entry $\mathcal{X}_{A,B}[i, j, k, l]$ is a counter of size $O(\log N)$. Our JD sketch structure is pictorially depicted in Figure 3. Note that, a JD sketch $\mathcal{X}_{A,B}$ essentially employs two distinct levels of hashing, with a first-level hash function applied to the *projected attribute(s)* A, and s_1 second-level hash functions applied to the *join attribute(s)* B of the input stream (by the 2-level hash sketches in the corresponding A-bucket).

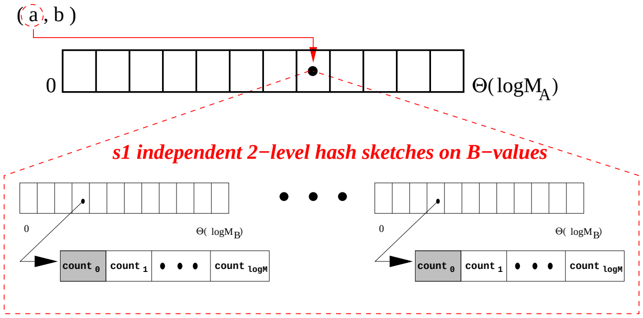


Figure 3: Our JD Sketch Synopsis for Stream R(A,B).

Maintenance. The maintenance algorithm for a JD sketch synopsis built over the R(A,B) stream follows easily along the lines of our 2-level hash sketch maintenance procedure. Once again, all counters in our data structure are initialized to zeros and, for each incoming update $< (a, b), \pm v >$ (where $(a, b) \in [M_A] \times [M_B]$), the a value is hashed using $h_A()$ to locate an appropriate A-bucket and all the 2-level hash sketches in that bucket are then updated using the $< b, \pm v >$ tuple; that is, each of the s_1 2-level hash sketches $\mathcal{X}_{A,B}[h_A(a), j]$ $(j = 1, \ldots, s_1)$ is updated with $< b, \pm v >$ using the 2-level hash sketch maintenance algorithm described in Section 2.2. Thus, the per-tuple maintenance cost for a JD sketch is $\Theta(s_1 \log M_B)$. It is easy to see that, as in the case of 2-level hash sketches, our JD sketch synopses are also impervious to deletions in the update stream.

3.2 The Join-Distinct Estimator

We now proceed to describe our Join-Distinct estimation algorithm for update streams. In a nutshell, our estimator constructs several independent pairs of *parallel* JD sketch synopses $(\mathcal{X}_{A,B}, \mathcal{X}_{C,B})$ for the input update streams R(A,B) and S(B,C) (respectively). The key here is that, for the $\mathcal{X}_{C,B}$ sketch, C plays the same role as attribute A in $\mathcal{X}_{A,B}$ (i.e., it is used to determine a first-level bucket in the JD sketch). Furthermore, both $\mathcal{X}_{A,B}$ and $\mathcal{X}_{C,B}$ use exactly the same hash functions for B in corresponding 2-level hash sketches for any A or C bucket; that is, the B-hash functions for $\mathcal{X}_{A,B}[*, j]$ and $\mathcal{X}_{C,B}[*, j]$ are identical for each $j = 1, \ldots, s_1$ (here, "$*$" denotes any first-level bucket in either of the two JD sketches). Then, at estimation time, each such pair of parallel JD sketches is *composed* in a novel manner to build an "FM-like" synopsis for the number of distinct (A, C) pairs in the join result. As we will see, this composition step is novel and non-trivial, and relies on the use of new, *composable* families of hash functions ($h_A()$ and $h_C()$) for the first level of our JD sketch synopses. We demonstrate how such hash-function families can be built and composed, and analyze their randomization properties. We begin by describing the basic JD sketch composition step in detail, and then formally describe our Join-Distinct estimation algorithm; the analysis of the space requirements and approximation guarantees for our estimator are discussed in Section 4.

Composing a Parallel JD Sketch Pair. Consider a pair of parallel JD sketch synopses $(\mathcal{X}_{A,B}, \mathcal{X}_{C,B})$ over R(A,B) and S(B,C). The goal of our JD sketch composition step is to combine information from $\mathcal{X}_{A,B}$ and $\mathcal{X}_{C,B}$ to produce an "FM-like" bitmap synopsis for the number of distinct (A, C) value pairs. The key here, of course, is that we should build this bitmap using *only* (A, C) *pairs in the result of the join* R⋈S. Thus, our composition step should use $\mathcal{X}_{A,B}$ and $\mathcal{X}_{C,B}$ to determine (with high probability (w.h.p.)) the (A, C)-value pairs that belong in the join result, and map such pairs to the cells of a bitmap $\mathcal{Y}_{A,C}$ of logarithmic size (i.e., $O(\log M_A + \log M_C)$) with exponentially-decreasing probabilities.

Our JD sketch composition algorithm, termed Compose, is depicted in Figure 4. Briefly, our algorithm considers all possible combinations of bucket indices $k \neq l$ from the two input parallel JD sketches(the $k \neq l$ restriction comes from a technical condition on the composition of the two first-level hash functions on A and C). For each such bucket pair $\mathcal{X}_{A,B}[k]$ and $\mathcal{X}_{C,B}[l]$, the composition algorithm employs the corresponding 2-level hash sketch synopses on B to estimate the size of the intersection for the sets of B values mapped to those first-level buckets (procedure IntersectionEstimator in Step 4). If that size is estimated to be greater than zero (i.e., the two buckets share at least one common join-attribute value), then the bit at location $\min\{k, l\}$ of the output (A, C)-bitmap $\mathcal{Y}_{A,C}$ is set to one (Step 5). Of course, since our JD sketches do not store the entire set of B values for each first-level bucket but, rather, a concise collection of independent 2-level hash sketch synopses, the decision of whether two first-level buckets join is necessar-

```
procedure Compose( $\mathcal{X}_{A,B}, \mathcal{X}_{C,B}$ )
Input: Pair of parallel JD sketch synopses for update
       streams R(A,B), S(B,C).
Output: Bitmap-sketch $\mathcal{Y}_{A,C}$ on $(A,C)$ value pairs in R⋈ S
begin
  $\mathcal{Y}_{A,C}$ := [ 0, ··· , 0 ]        // initialize
  for each bucket $k$ of $\mathcal{X}_{A,B}$ do
      for each bucket $l \neq k$ of $\mathcal{X}_{C,B}$ do
          if IntersectionEstimator($\mathcal{X}_{A,B}[k]$, $\mathcal{X}_{C,B}[l]$) $\geq 1$ then
              $\mathcal{Y}_{A,C}[\min\{k,l\}] := 1$
  return($\mathcal{Y}_{A,C}$)
end
```

Figure 4: JD Sketch Composition Algorithm.

ily approximate and probabilistic (based on the 2-level hash sketch intersection estimate). Our analysis in Section 4 will clearly demonstrate the effect of this approximation on the error guarantees and space requirements of our `Join-Distinct` estimation algorithm.

Our `JD` sketch composition algorithm actually implements a *composite hash function* $h_{A,C}()$ over (A,C)-value pairs that combines the first-level hash functions $h_A()$ and $h_C()$ from sketches $\mathcal{X}_{A,B}$ and $\mathcal{X}_{C,B}$, respectively. We now examine this composite hash function and its properties in more detail. Our ability to use the final (A,C) bitmap synopsis $\mathcal{Y}_{A,C}$ output by algorithm Compose to estimate the number of distinct (A,C)-value pairs in `R(A,B)⋈S(B,C)` depends crucially on designing a composite hash function $h_{A,C}()$ (based on the individual functions $h_A()$, $h_C()$) that guarantees certain randomizing properties similar to those of the hash functions used in the Flajolet-Martin or 2-level hash sketch estimators. More specifically, our composite hash function $h_{A,C}()$ needs to (a) allow us to easily map (A,C)-value pairs onto a logarithmic range with exponentially-decreasing probabilities; and, (b) guarantee a certain level of *independence* between distinct tuples in the (A,C)-value pair domain. The key problem here, of course, is that, since the tuples from `R(A,B)` and `S(B,C)` are seen in arbitrary order and are individually summarized in the $\mathcal{X}_{A,B}$ and $\mathcal{X}_{C,B}$ synopses, our composite hash-function construction can only use the hash values $\textsc{lsb}(h_A())$ and $\textsc{lsb}(h_C())$ maintained in the individual `JD` sketches. This limitation makes the problem nontrivial, since it essentially rules out standard pseudo-random hash-function constructions, e.g., using finite-field polynomial arithmetic over $[M_A] \times [M_C]$ [2, 9]. To the best of our knowledge, our hash-function composition problem is novel and has not been previously studied in the pseudo-random generator literature. In the following theorem, we establish the existence of such *composable* hash-function pairs $(h_A(), h_C())$ and demonstrate that, for such functions, the composition procedure in algorithm Compose indeed guarantees the required properties for the resulting composite hash function (i.e., exponentially-decreasing mapping probabilities and pairwise independence for (A,C)-value pairs).

Theorem 3.1. *The hash functions $(h_A(), h_C())$ used to build a parallel* `JD` *sketch pair $(\mathcal{X}_{A,B}, \mathcal{X}_{C,B})$ can be constructed so that the hash-function composition procedure in algorithm* Compose*: (1) guarantees that (A,C)-value pairs are mapped onto a logarithmic range $\Theta(\log\max\{M_A, M_C\})$ with exponentially-decreasing probabilities (in particular, the mapping probability for the j^{th} bucket is $p_j = \Theta(4^{-(j+1)})$); and, (2) gives a composite hash function $h_{A,C}()$ that guarantees* pairwise independence *in the domain of (A,C)-value pairs.* ∎

Proof: Let $M = \max\{M_A, M_C\}$, and construct the individual hash functions $h_A()$ and $h_C()$ as linear hash functions over the field $\mathrm{GF}(2^{2\log M})$ comprising boolean-coefficient polynomials of degree at most $2\log M - 1$ (or, equivalently, $2\log M$-bit integers) [19]. [5] More specifically, given input arguments $a, c \in M$ (i.e., field elements of $\mathrm{GF}(2^{\log M})$), define $h_A(a) = r + s \cdot \alpha \cdot a$ and $h_C(c) = t + s \cdot c$, where r, s, t are randomly-chosen field elements of $\mathrm{GF}(2^{2\log M})$ (i.e., $2\log M$-bit integers) such that $s \neq 0$ and $r \neq t$, and α denotes the $2\log M$-bit constant with a single 1-bit at location $\log M$. Note that the α constant corresponds to the polynomial $x^{\log M}$ in $\mathrm{GF}(2^{2\log M})$ and, in GF-polynomial arithmetic [19], the operation $\alpha \cdot a$ essentially left-shifts the argument a by $\log M$ bits. Clearly, both $h_A()$ and $h_C()$ guarantee pairwise independence over M (see, e.g., [2]). Let $h_{A,C}()$ denote the composite hash function over (A,C)-value pairs defined by algorithm Compose. Recall that the first-level buckets of the `R(A,B)` and `S(B,C)` `JD` sketches basically record the locations of the *least-significant* 1*-bits* ($\textsc{lsb}$'s) of $h_A()$ and $h_C()$ values, respectively. It is then easy to see that, given an (A,C)-value pair, say (a,c), with $k = \textsc{lsb}(h_A(a))$ and $l = \textsc{lsb}(h_C(c))$ with $k \neq l$ (and, of course, assuming that the corresponding sets of B-values join), the Compose algorithm will map that pair to bucket $j = \textsc{lsb}(h_{A,C}(a,c)) = \min\{k,l\} = \min\{\textsc{lsb}(h_A(a)), \textsc{lsb}(h_C(c))\}$ of $\mathcal{Y}_{A,C}$. And, since $k \neq l$, it is easy to see that $j = \textsc{lsb}(h_{A,C}(a,c)) = \textsc{lsb}(h_A(a) + h_C(c))$; in other words, the composite hash function computed by Compose is essentially $h_{A,C}(a,c) = h_A(a) + h_C(c) = (r+t) + s \cdot (\alpha \cdot a + c)$. (Recall that in GF-field arithmetic "+" is basically an XOR operation.)

To estimate the mapping probabilities for $\textsc{lsb}(h_{A,C}(a,c))$ over the $\Theta(\log M)$ range, observe that, by our construction:

$$\begin{aligned}\mathbf{Pr}\left[\textsc{lsb}(h_{A,C}(a,c)) = j\right] = \; & \mathbf{Pr}\left[l_A(a) = j \text{ and } l_C(c) > j\right] \\ & + \mathbf{Pr}\left[l_C(c) = j \text{ and } l_A(a) > j\right],\end{aligned}$$

where $l_A(a) = \textsc{lsb}(h_A(a))$ (with $l_C(c)$ defined similarly). Using the pairwise independence of $h_A()$ and $h_C()$ and the fact that $\mathbf{Pr}\left[l_A(a) = i\right] = \mathbf{Pr}\left[l_C(c) = i\right] = \frac{1}{2^{i+1}}$, the above expression gives:

$$\mathbf{Pr}\left[\textsc{lsb}(h_{A,C}(a,c)) = j\right] \;=\; 2 \cdot \frac{1}{2^{j+1}} \sum_{i \geq j+2} \frac{1}{2^i} \;=\; \frac{2}{4^{j+1}}.$$

Thus, our construnction gives exponentially-decreasing mapping probabilities, as needed.

We now give a counting argument to demonstrate that our composite hash function $h_{A,C}()$ indeed guarantees *pairwise independence* over (A,C)-value pairs. Let $K = M^2$ denote the number of elements in the $\mathrm{GF}(2^{2\log M})$ field. Given an input (a,c) pair and a hash function value v, the number of ways in which the (a,c) pair maps to v (i.e., $h_{A,C}(a,c) = v$) is given by the number of choices for the triple of parameters (r,s,t) for which the equation

$$h_{A,C}(a,c) = v \text{ or, equivalently, } (r+t) + s \cdot (\alpha \cdot a + c) = v$$

is satisfied in $\mathrm{GF}(2^{2\log M})$. It is easy to see that, for each of the $K(K-1)$ possible (random) choices for the (r,s) pair (remember that $s \neq 0$), the above equation essentially specifies a unique solution for t in $\mathrm{GF}(2^{2\log M})$, namely, $t = v - s \cdot (\alpha \cdot a + c) - r$. In other words, out of the (approximately) K^3 possible total choices for (r,s,t) triples over $\mathrm{GF}(2^{2\log M})$,

[5] Without loss of generality, we assume M to be a power of 2 in what follows.

only (approximately) K^2 choices will satisfy the equation $h_{A,C}(a,c) = v$; therefore, given (a,c) and v, we have

$$\mathbf{Pr}\left[h_{A,C}(a,c) = v\right] \;\approx\; \frac{K^2}{K^3} \;=\; \frac{1}{K}. \qquad (1)$$

Now, suppose we are given two distinct input pairs $(a_1, c_1) \neq (a_2, c_2)$ and hash values v_1, v_2. We again want to count the number of ways in which the two equations $h_{A,C}(a_1, c_1) = v_1$ and $h_{A,C}(a_2, c_2) = v_2$ are satisfied over the possible choices for the (r, s, t) triples. We can write this system of equations in matrix form as

$$\begin{bmatrix} 1 & \alpha \cdot a_1 + c_1 \\ 1 & \alpha \cdot a_2 + c_2 \end{bmatrix} \begin{bmatrix} r+t \\ s \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}.$$

The fact that $(a_1, c_1) \neq (a_2, c_2)$ also implies that $\alpha \cdot a_1 + c_1 \neq \alpha \cdot a_2 + c_2$ (since they correspond to different $2 \log M$-bit numbers as explained earlier); thus, the determinant of the above system of equations is non-zero, which implies a unique solution for $(r + t, s)$. So, for each of the K possible choices for r, the system of equations $h_{A,C}(a_1, c_1) = v_1$ and $h_{A,C}(a_2, c_2) = v_2$ uniquely determine s and t; in other words, over the (approximately) K^3 choices for (r, s, t) triples over $\mathrm{GF}(2^{2 \log M})$, we have

$$\mathbf{Pr}\left[h_{A,C}(a_1,c_1) = v_1 \text{ and } h_{A,C}(a_2,c_2) = v_2\right] \approx \frac{K}{K^3} = \frac{1}{K^2}$$
$$= \mathbf{Pr}\left[h_{A,C}(a_1,c_1) = v_1\right] \cdot \mathbf{Pr}\left[h_{A,C}(a_2,c_2) = v_2\right]$$

(based on Equation (1)). This proves the pairwise independence property for the composite hash function $h_{A,C}()$, completing our proof argument. ∎

The Join-Distinct Estimator. The pseudo-code of our algorithm for producing an (ϵ, δ) probabilistic estimate for the `Join-Distinct` problem (termed JDEstimator) is depicted in Figure 5. Briefly, our estimator employs an input collection of s_2 independent `JD` sketch synopsis pairs built in parallel over the input update streams `R(A,B)` and `S(B,C)`. (The values for the s_1, s_2 parameters are determined by our analysis in Section 4.) Each such parallel `JD` sketch pair $(\mathcal{X}^i_{A,B}, \mathcal{X}^i_{C,B})$ is first composed (using algorithm Compose) to produce a bitmap synopsis $\mathcal{Y}^i_{A,C}$ over the (A, C)-value pairs in the join result (Steps 1-2). Then, the resulting s_2 $\mathcal{Y}^i_{A,C}$ bitmaps are examined level-by-level in parallel, searching for the *highest bucket level* ("index") at which the number of bitmaps that satisfy the condition: "$\mathcal{Y}^i_{A,C}[\text{index}] = 1$", lies between $s_2 \cdot (1 - 2\epsilon)\frac{\epsilon}{8}$ and $s_2 \cdot (1 + \epsilon)\epsilon$ (Steps 4-11). (These lower and upper bound values are again based on our analysis.) The final estimate returned is equal to the fraction of $\mathcal{Y}^i_{A,C}$ synopses satisfying the condition at level "index" (i.e., "count/s_2") scaled by the inverse of the mapping probability for that level $p_{\text{index}} = 2 \cdot 4^{-(\text{index}+1)}$ (by Theorem 3.1). Note that, if no such level is found, our algorithm only returns a "failure" indicator; our analysis shows that this event can only happen with low probability.

4. ANALYSIS

In this section, we present the analysis of the approximation guarantees and space requirements of our `Join-Distinct` estimation algorithm (Figure 5). We begin by introducing some additional notation and terminology that we will use in our analysis. We then proceed to demonstrate the main theoretical result in this paper that establishes the (worst-case) space requirements of our `Join-Distinct` estimator for guaranteeing small relative error with high probability. Finally, we present a lower bound on the space usage of any (possibly, randomized) estimation algorithm for the `Join-Distinct` problem, showing that our estimator is within small factors of the best possible solution.

```
procedure JDEstimator( $(\mathcal{X}^i_{A,B}, \mathcal{X}^i_{C,B})$, $i = 1, \ldots, s_2$, $\epsilon$ )
Input: $s_2$ independent pairs of parallel JD sketch synopses
        $(\mathcal{X}^i_{A,B}, \mathcal{X}^i_{C,B})$ for the update streams R(A,B), S(B,C).
Output: $(\epsilon, \delta)$-estimate for $|\pi_{A,C}($R(A,B) $\bowtie$ S(B,C)$)|$.
begin
1.  for $i := 1$ to $s_2$ do
2.      $\mathcal{Y}^i_{A,C}$ := Compose($\mathcal{X}^i_{A,B}, \mathcal{X}^i_{C,B}$)
3.  let $B$ denote the highest bucket index in the $\mathcal{Y}_{A,C}$ synopses
4.  for index := $B$ downto 0 do
5.      count := 0
6.      for $i := 1$ to $s_2$ do
7.          if $\mathcal{Y}^i_{A,C}$[index] = 1 then count := count +1
8.      endfor
9.      if $\left( (1-2\epsilon)\frac{\epsilon}{8} \le \frac{\text{count}}{s_2} \le (1+\epsilon)\epsilon \right)$ then
10.         return( count/($p_{\text{index}} \cdot s_2$) )
11. endfor
12. return( fail )
end
```

Figure 5: Join-Distinct Estimation Algorithm.

4.1 Notation

Let S_p denote a random sample of distinct (A, C)-value pairs drawn from the cartesian product $[M_A] \times [M_C]$ of the underlying value domains, where each possible pair is selected for inclusion in the sample with probability p. (Note that the collection of (A, C)-value pairs mapped to level i of each bitmap synopsis $\mathcal{Y}_{A,C}$ generated in Steps 1-2 of our `Join-Distinct` estimator can essentially be seen as such a random sample S_{p_i} with selection probability $p_i = \Theta(4^{-(i+1)})$.) We also define two additional random variables U_p and T_p over the sample S_p of (A, C)-value pairs as follows:

$$U_p = |\{b : \exists (a,c) \in S_p \text{ such that } [(a,b) \in \texttt{R(A,B)} \text{ OR } (b,c) \in \texttt{S(B,C)}]\}|,$$
$$T_p = |\{b : \exists (a,c) \in S_p \text{ such that } [(a,b) \in \texttt{R(A,B)} \text{ AND } (b,c) \in \texttt{S(B,C)}]\}|.$$

In other words, the U_p random variable counts the size of the total B-neighborhood of the S_p sample; that is, the number of distinct B-values that are "connected" (through `R(A,B)` or `S(B,C)`) to *either* the A *or* the C component of an (A, C)-value pair in S_p. T_p, on the other hand, captures the size of the B-support of the S_p sample in the join result `R(A,B)`⋈`S(B,C)`; that is, the number of distinct B-values that appear with *both* the A *and* the C component of an (A, C)-value pair in S_p.

Figure 6 depicts an example instantiation for the sample S_p and the above-defined random variates using the simple analogy of a 3-partite graph over $[M_A] \times [M_B] \times [M_C]$ as a representation for the join `R(A,B)`⋈`S(B,C)`. Note that an (a, b) ((b, c)) edge in the figure represents the existence of at least one such data tuple in the `R(A,B)` (resp., `S(B,C)`) stream. We define the B-*degree* of a value x in A (or, C) as the number of distinct B-values that are "connected" to x in the input stream, and let deg(A), deg(C) denote the *average* B-*degree* of values in A and C, respectively; for example, in

Figure 6, $\deg(A) = \frac{10}{6} = 1.67$.

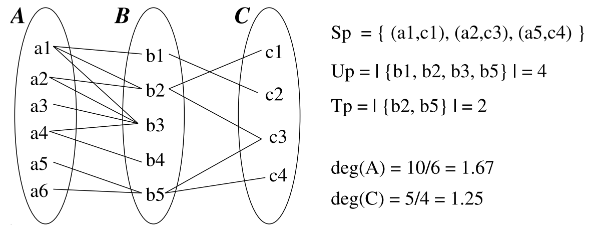


Figure 6: Example `R(A,B)`$\bowtie$ `S(B,C)` Instance.

4.2 Space-Usage Analysis

We now present the analysis of the worst-case space requirements of our probabilistic `Join-Distinct` estimator (as a function of the associated (ϵ, δ) estimation guarantees). In order to simplify the exposition, we abstract away many of the constants in our bounds using $\Theta()$-notation – the derivation of the exact constants for our analysis is deferred to the full version of this paper [8]. We should, once again, emphasize that our analysis only considers the *worst-case* space usage for our algorithm; as our experimental results demonstrate, the average-case behavior of our schemes is consistently much better than what our pessimistic, worst-case bounds would predict (Section 6).

Fix a specific $\mathcal{Y}_{A,C}$ bitmap synopsis for (A, C)-value pairs produced by algorithm JDEstimator, and consider a particular bucket level l in this bitmap. As we already mentioned in Section 3, our estimator makes use of the 2-level hash sketches synopses in the original `JD` sketch buckets for `R(A,B)` and `S(B,C)` that map to bucket $\mathcal{Y}_{A,C}[l]$ in order to decide whether there exists a common B-value that would give rise to an (A, C)-value pair in this bucket in the `Join-Distinct` result. In other words, our algorithm sets $\mathcal{Y}_{A,C}[l] = 1$ only if our 2-level hash sketch set-intersection estimator (IntersectionEstimator in Figure 4) is able to accurately detect that the intersection of B-values in the corresponding buckets of the two `JD` sketches being composed is non-empty (given, of course, that this is indeed the case). Since our intersection estimator (based on concise 2-level hash sketches) is necessarily probabilistic, we need to ensure that the detection of such non-empty B-value intersections for $\mathcal{Y}_{A,C}[l]$ is done with high probability. Observe that the set of (A, C)-value pairs mapped to $\mathcal{Y}_{A,C}[l]$ is essentially a random sample S_{p_l} of $[M_A] \times [M_C]$ with selection probability $p_l = \Theta(4^{-(l+1)})$. Furthermore, by our earlier results on set-intersection estimation using 2-level hash sketches (Theorem 2.1), we know that we can detect a non-empty intersection with high probability ($\geq 1-\delta$) as long as the number s_1 of 2-level hash sketch synopses maintained is at least $\Theta(\log(\frac{1}{\delta}) \cdot |\text{union}|/|\text{intersection}|)$. (Detecting the condition $|\text{intersection}| \geq 1$ can be done using any constant relative error $\epsilon < 1/2$ in Theorem 2.1.) Thus, using the notation introduced earlier in this section and the fact that our algorithm maintains s_1 2-level hash sketches at each `JD` sketch level, the conditional probability that our estimator detects (w.h.p.) a non-empty B intersection at level l of $\mathcal{Y}_{A,C}$ (given that it exists) can be expressed as:

$$\mathbf{Pr}\left[\text{detect at } l\right] = \mathbf{Pr}\left[\frac{U_{p_l}}{T_{p_l}} \leq \Theta(s_1/\log(\frac{1}{\delta})) \mid T_{p_l} \geq 1\right].$$

Let $e(p_l)$ denote the conditional expectation $\mathbf{E}\big[\frac{U_{p_l}}{T_{p_l}} \mid T_{p_l} \geq 1\big]$, and assume that the number of per-level 2-level hash sketches is $s_1 = \Theta(e(p_l) \cdot \log(\frac{1}{\delta})/\epsilon)$. Then, a simple application of Markov's inequality [21] gives that the above probability is $\mathbf{Pr}\left[\text{detect at } l\right] \geq 1 - \Theta(\epsilon)$.

Let $m = |\pi_{A,C}(\texttt{R(A,B)} \bowtie \texttt{S(B,C)})|$, i.e., the true size of our `Join-Distinct` query, and let $\hat{m}$ denote the estimate for m returned by our JDEstimator algorithm. Define the probability p as $p = \epsilon/m$, and let l be the unique level in the $\mathcal{Y}_{A,C}$ bitmap synopsis such that $\frac{p}{4} < p_l \leq p$ (remember that the per-level probabilities decrease by powers of 4). Using the union bound and the pairwise-independence properties of the mappings in $\mathcal{Y}_{A,C}$ in conjunction with the Inclusion/Exclusion Principle, the probability $\mathbf{Pr}\left[T_{p_l} \geq 1\right]$ can be bounded as follows:

$$mp_l(1 - \frac{mp_l}{2}) \;\leq mp_l - \binom{m}{2} p_l^2 \;\leq\; \mathbf{Pr}\left[T_{p_l} \geq 1\right] \;\leq\; mp_l. \tag{2}$$

Now, consider the probability $r_l = \mathbf{Pr}\left[\mathcal{Y}_{A,C}[l] = 1\right]$. Since our 2-level hash sketch set-intersection estimator guarantees *no false positives* (i.e., will not detect a non-empty intersection unless it actually exists), it is easy to see that $r_l \leq \mathbf{Pr}\left[T_{p_l} \geq 1\right] \leq mp_l \leq \epsilon$ (since $p_l \leq p = \epsilon/m$); furthermore, we have:

$$\begin{aligned} r_l = \mathbf{Pr}\left[\mathcal{Y}_{A,C}[l] = 1\right] \;&\geq\; \mathbf{Pr}\left[T_{p_l} \geq 1\right] \cdot \mathbf{Pr}\left[\text{detect at } l\right] \\ &\geq\; (1 - \Theta(\epsilon))mp_l(1 - \frac{mp_l}{2}) \\ &\geq\; (1 - \Theta(\epsilon))\frac{\epsilon}{8} \end{aligned}$$

where the last inequality follows from our earlier analysis, our choice of level l, Equation (2), and the fact that $\epsilon < 1$.

Given the above bounds on the r_l probability, a simple application of the Chernoff bound [21] shows that, using $s_2 = \Theta(\frac{\log(1/\delta)}{\epsilon^3})$ independent copies of the $\mathcal{Y}_{A,C}$ bitmap synopsis, the fraction $\hat{r_l}$ of the level-l buckets that satisfy the condition "$\mathcal{Y}_{A,C}[l] = 1$" (i.e., the $\frac{\text{count}}{s_2}$ ratio in algorithm JDEstimator) is guaranteed to be in the range $(1 \pm \epsilon)r_l$. [6] Using our bounds on r_l, this implies that, with high probability, we have $(1 - \Theta(\epsilon))^2 mp_l(1 - \frac{mp_l}{2}) \leq \hat{r_l} \leq (1 + \epsilon)mp_l$, which, of course, implies that the observed ratio estimate $\frac{\hat{r_l}}{p_l}$ (returned in Step 10 of algorithm JDEstimator) satisfies

$$(1 - \Theta(\epsilon))^3 m \;\leq\; \frac{\hat{r_l}}{p_l} \;\leq\; (1 + \epsilon)m.$$

(Note that, since $\Theta(\epsilon) < 1$, we have $(1-\Theta(\epsilon))^3 > (1-3\Theta(\epsilon))$; thus, we get a $\Theta(\epsilon)$ approximation by simply using $\epsilon' = \epsilon/3$.)

As the above analysis demonstrates, if our estimation algorithm selects the (unique) level l such that $p_l \in (\frac{\epsilon}{4m}, \frac{\epsilon}{m}]$ (i.e., "index" $= l$ in Figure 5), then the estimate $\frac{\hat{r_l}}{p_l}$ returned

[6] Note that the space requirements for our distinct-count estimation procedure over the "composed" $\mathcal{Y}_{A,C}$ bitmaps are worse than those of known (ϵ, δ)-approximation schemes for simple distinct-count estimation (e.g., [9]). This is primarily due to the fact that our composed hash functions can guarantee *only pairwise independence*.

(Step 10 of JDEstimator) will be within a small relative error of the true Join-Distinct count m with high probability. We now demonstrate that, with an appropriately small value of ϵ, our algorithm is in fact *guaranteed to terminate at level l* or $l+1$ with high probability. Consider any level $k \geq l+2$. Note that, the mapping probability for level k is $p_k \leq \frac{p}{16} = \frac{\epsilon}{16m}$. Following the steps of the analysis for level l, it is easy to see that, even for this level $k \geq l+2$, the fraction $\hat{r_k}$ of bitmaps satisfying "$\mathcal{Y}_{A,C}[k] = 1$" will be $\hat{r_k} \leq (1+\epsilon)mp_k \leq (1+\epsilon)\frac{\epsilon}{16}$, with high probability. Thus, since $\hat{r_l} \geq (1-\Theta(\epsilon))^2\frac{\epsilon}{8}$ (w.h.p.), our algorithm, with high probability, (1) terminates at either level l or $l+1$; and, (2) does not terminate at any level $k \geq l+2$, *as long as* $(1-\Theta(\epsilon))^2 > \frac{1+\epsilon}{2}$ (that is, some "separation" exists between levels l and $l+2$). It is easy to see that this condition can easily be satisfied with an appropriately small value of ϵ (the exact constant factors can be found in the full paper [8]). The error guarantees for our final estimate will hold regardless of which of the two levels (l or $l+1$) our algorithm chooses.

Remember that our analysis relies on maintaining $s_1 = \Theta(e(p_l) \cdot \log(\frac{1}{\delta})/\epsilon)$ 2-level hash sketch summaries at each level of our JD sketches, where $e(p_l)$ denotes the conditional expectation $\mathbf{E}\big[\frac{U_{p_l}}{T_{p_l}} \mid T_{p_l} \geq 1\big]$. As a final step in our analysis, the following lemma demonstrates that, for any level l such that $p_l \leq p = \frac{\epsilon}{m}$ (like the levels considered in our analysis above), the conditional expectation $e(p) = \mathbf{E}\big[\frac{U_p}{T_p} \mid T_p \geq 1\big]$ (i.e., using p instead of p_l) is a good approximation to $e(p_l)$; the detailed proof is deferred to the full paper [8]. This result allows us to use $e(p)$ instead of the "level-specific" $e(p_l)$ term in the remainder of our discussion.

LEMMA 4.1. *Assume $m \geq 1$, $\epsilon < \frac{1}{4}$, and $0 < q \leq p = \frac{\epsilon}{m}$. Then, $\mathbf{E}\big[\frac{U_q}{T_q} \mid T_q \geq 1\big] \leq (1+5\epsilon) \cdot \mathbf{E}\big[\frac{U_p}{T_p} \mid T_p \geq 1\big]$.* ∎

The following theorem summarizes the results of our analysis; we use $M = \max\{M_A, M_B, M_C\}$ to simplify the statement of our worst-case space bounds.

THEOREM 4.2. *Let m denote the result size of an input* Join-Distinct *query on two update streams* R(A,B) *and* S(B,C), *and let $M = \max\{M_A, M_B, M_C\}$, $e(p) = \mathbf{E}\big[\frac{U_p}{T_p} \mid T_p \geq 1\big]$, where $p = \frac{\epsilon}{m}$. Algorithm* JDEstimator *returns and (ϵ, δ)-estimate for m using* JD *sketch synopses with a total storage requirement of $\Theta(s_1 s_2 \log^3 M \log N)$ and per-tuple update time of $\Theta(s_1 \log M)$, where: $s_1 = \Theta\left(\frac{e(p)\log(1/\delta)}{\epsilon}\right)$, and $s_2 = \Theta\left(\frac{\log(1/\delta)}{\epsilon^3}\right)$.* ∎

The conditional-expectation term $e(p)$ basically captures the expected ratio of the total B-neighborhood to the corresponding B-support for the join operation over randomly-chosen (A, C) pairs. Intuitively, large $e(p)$ values imply that the underlying join query has relatively small support over $A \times C$ and, thus, our probabilistic estimation techniques will require more space in order to be able to detect supporting tuples for the join.

4.3 Further Discussion and Lower Bounds

Ignoring the conditional-expectation term $e(p)$, Theorem 4.2 essentially states that our Join-Distinct estimation problem over update streams is solvable in small (i.e., polylogarithmic) space; unfortunately, the existence of the $e(p)$ term in our space bounds does not allow us to make such a claim. It is, therefore, interesting to ask at this point how good our randomized estimator really is – is it perhaps possible to design a new estimation procedures that significantly improves on the space requirements stated in Theorem 4.2? In this section, we further discuss the role of the conditional-expectation term in the bounds of Theorem 4.2, and we answer the above question in the negative by demonstrating space lower bounds for Join-Distinct estimators working in the streaming model.

Recall that the conditional expectation $e(p)$ basically captures the expectation of the union-over-intersection ratio of B values for a randomly-chosen subset of (A, C)-value pairs in our input streams (given that this intersection is non-empty). Since the $e(p)$ term is not very intuitive, we provide, in the following lemma, a *relaxation* (i.e., an upper bound) for the conditional expectation $e(p)$ in terms of "easier" properties of the underlying data streams.

LEMMA 4.3. *Let $p = \frac{\epsilon}{m}$. Then, $e(p) = \mathbf{E}\big[\frac{U_p}{T_p} \mid T_p \geq 1\big] \leq \frac{\epsilon|A||C|(\deg(A)+\deg(C))}{m}$.* ∎

Lemma 4.3 essentially states that, with limited space, our estimator is guaranteed to provide robust estimates for Join-Distinct cardinalities that are sufficiently large compared to the product of the total number of distinct (A, C)-value pairs seen in the input streams and the average B-degree of A and C values. We should note here that the upper bound on $e(p)$ in Lemma 4.3 is fairly loose – it essentially assumes that B-values are not shared across different values in A or C (and, once again, our analysis only looks at worst-case bounds for our estimators). The following theorem proves a lower bound for all approximation algorithms for our Join-Distinct estimation problem over streams, which (based on Lemma 4.3) shows that the space requirements of our JDEstimator algorithm cannot be significantly improved (i.e., it is within small constant and log factors of the optimal). Our proof makes use of information-theoretic arguments and Yao's lower-bounding lemma, and can be found in Appendix A.

THEOREM 4.4. *Let Φ be any scheme for building a summary of a database relation, and let $\mathcal{A}$ be any (randomized or deterministic) scheme such that $\mathcal{A}(\Phi($*R(A,B)*$), \Phi($*S(B,C)*$))$ provides a constant-factor approximation of $|\pi_{A,C}($*R(A,B) $\bowtie$ S(B,C)*$)|$ with high probability. Then, given an a-priori lower bound β on the* Join-Distinct *result size, where $(\deg(A) + \deg(C))\sqrt{\beta} \leq 2N$, the size of the summary Φ must be at least $\frac{|A||C|(\deg(A)+\deg(C))}{2\beta}$.* ∎

5. EXTENSIONS

Handling Other Join-Distinct Aggregates. Our discussion thus far has focused on the query Q$= |\pi_{A,C}($R(A,B) $\bowtie$ S(B,C)$)|$ (where $A, C \neq \phi$ and $A \cap B = B \cap C = \phi$). We now discuss how our proposed Join-Distinct estimation techniques can be adapted to deal with other forms of Join-Distinct COUNT queries conforming to the general query pattern described in Section 2.1.

For example, consider the case of a one-sided projection query Q'$= |\pi_{A,B}($R(A,B) $\bowtie$ S(B,C)$)|$, where we seek to estimate the number of distinct R(A,B) tuples joining with at least one tuple from S(B,C)(i.e., the number of distinct tuples in a stream *semi-join*). Our JDEstimator algorithm can readily handle the estimation of Q' – the key idea is to simply replace attribute C by B in the JD sketch construction and estimation steps already described for Q. Thus, for Q', the JD sketch synopsis built on the S(B,C) side actually uses a first-level hash function on attribute B in addition to the per-bucket 2-level hash sketch collections (also built on B); then, when the JD sketch composition process (Figure 4) is applied (at estimation time), the output is a set of FM-like bitmap synopses $\mathcal{Y}_{A,B}$ on (A, B)-value pairs that can be used to produce an estimate for Q'. Similarly consider the case of a "full-projection" query Q''$= |\pi_{A,B,C}($R(A,B) $\bowtie$ S(B,C)$)|$, that simply outputs the number of distinct (A, B, C) tuples in the join result. Handling Q'' simply involves replacing A (C) by (A, B) (resp., (B, C)) in the JD sketch construction and JDEstimator algorithms for Q.

Handling other forms of Join-Distinct aggregates (e.g., predicate selectivities over the result of a Join-Distinct query) is slightly more complicated. In a nutshell, the key idea is to augment the first-level hash tables in our JD sketch synopses with count signatures for the corresponding projected-attribute values. Then, after the JD sketch composition step, these count signatures can be employed (in a manner similar to [9]) to identify *singleton* projected-attribute values. Such singletons essentially form a *distinct sample* of the projected-attribute values in the Join-Distinct result and, thus, can be used to estimate distinct-values predicate selectivities (as in [10]). The results of our analysis for our estimators can also be readily extended to cover the aforementioned different forms of Join-Distinct estimation problems; due to space constraints, the details are deferred to the full paper [8].

An Alternative, $\Theta(|B|)$-Space Join-Distinct Estimator. In developing our Join-Distinct estimation algorithms, we have thus far insisted on polylogarithmic-space synopsis structures. Such a restriction makes sense, for example, when joining on attributes with very large numbers of distinct values (e.g., (source, destination) IP-address pairs). When this is not the case, and using $\Theta(|B|)$ space is a viable option for estimating Q$= |\pi_{A,C}($R(A,B) $\bowtie$ S(B,C)$)|$, we now briefly describe a different, simpler Join-Distinct estimation algorithm.

In a nutshell, our alternative algorithm again relies on our idea of using *composable hash functions* (Theorem 3.1) to compose a bit-vector sketch on (A, C) from hash sketches built individually on R(A,B) and S(B,C); however, the synopsis structure used is different from that of JDEstimator. More specifically, we make use of a $\Theta(|B|)$ bit-vector indicating the existence of a particular B value in an input stream; for each non-empty B-bucket, we maintain a collection of independent FM synopses (using counters instead of bits) that summarize the collection of distinct A (C) values for tuples in R(A,B) (resp., S(B,C)) containing this specific B-value. (These FM synopses are built using composable hash functions $h_A()$ and $h_C()$, as in Section 3.2.) At estimation time, the A and C FM synopses for each B-value that appears in *both* R(A,B) and S(B,C) (note that, since we are using $\Theta(|B|)$ space, this co-occurrence test is now *exact*) are composed to produce an (A, C)-bitmap sketch for that B-value. Then, all such (A, C)-bitmaps are unioned (by simple bitwise OR-ing) to give FM bit-vectors on (A, C) for the result of R$\bowtie$ S, that can be directly used for estimating Q. It can be shown [8], that our alternative Join-Distinct estimator can produce an (ϵ, δ)-estimate for Q using $\Theta(|B| \log(\frac{1}{\delta})/\epsilon^3)$ space, and can also be easily extended to handle other forms of Join-Distinct aggregates. Note that this new estimator has a "hard" space requirement of at least $\Omega(|B|)$ and, unfortunately, there is no obvious way to extend it so that it works with less than $O(|B|)$ space.

6. EXPERIMENTAL STUDY

In this section, we describe the results obtained from a preliminary experimental study of the algorithmic techniques developed in this paper. 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 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 Join-Distinct aggregates over continuous streaming data.

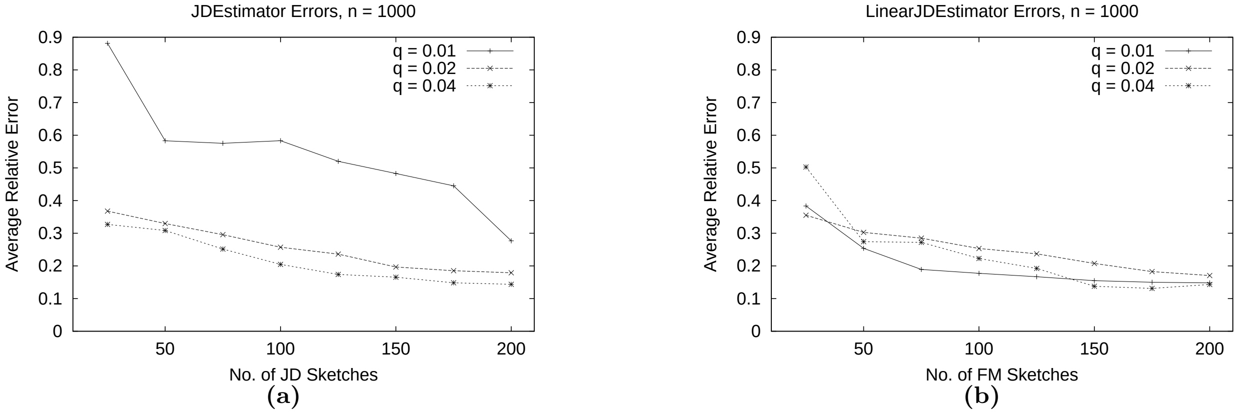
6.1 Testbed

Methodology. Our experiments study the *accuracy* of the two probabilistic Join-Distinct cardinality estimation techniques proposed in this paper, namely our JDEstimator algorithm (based on JD sketches) and our alternative $\Theta(|B|)$-space estimation algorithm (termed LinearJDEstimator) discussed in Section 5. We tested our two estimators over different synthetic data streams, employing the conventional *absolute relative error metric* as the primary metric for gauging approximation accuracy; that is, given two streams R(A,B) and S(B,C) and an estimate $\hat{J}$ of their Join-Distinct cardinality J, we define the error of the estimate as the ratio $\frac{|\hat{J}-J|}{|J|}$. Our experiments measure the errors of our estimation algorithms as a function of the space made available for building synopses of the two input data streams. (Note, once again, that ours is the *first* solution to the Join-Distinct estimation problem in the streaming context.) To account for the randomness in our techniques, all numbers reported below represent averages over 10 runs of our algorithms with different random seeds.

Data Sets. Given that our synopsis data structures for both JDEstimator and LinearJDEstimator are impervious to deletions in the stream, our synthetic data generator focuses solely on *insert-only* data streams. Furthermore, since our analysis of JDEstimator shows that its accuracy depends on the ratio U_p/T_p of neighborhood size to join support (Section 4), we employ a controlled data-generation process that allows us to effectively vary this ratio.

More specifically, our synthetic data sets are generated using a *random-graph model*. A random graph is specified using two parameters: the number of nodes n, and edge probability q. Given these two parameters, we construct a bipartite graph on $n \times n$ vertices, where an edge between each pair of vertices (i, j) is added independently with probability q. The edges (i, j) of the final bipartite graph are essentially the (two-attribute) tuples of an input stream R(A,B). The second stream S(B,C) is also generated similarly, using the same set of $n \times n$ graph nodes and the same edge probability q. It is not difficult to see that the size of a Join-Distinct query over R(A,B) and S(B,C) is equal to the number of pairs of (A, C) vertices in the combined tri-partite graph which have a path of length two between them.

Our random-graph model allows us to effectively control



(a) **(b)**

Figure 7: Average relative estimation error numbers for JDEstimator (a) and LinearJDEstimator (b), over random data graphs with $n = 1,000$ and $q \in \{0.01, 0.02, 0.04\}$.

the U_p/T_p ratio. Indeed, if we look at a random tuple $(a, c) \in A \times C$, the expected size of the U_p neighborhood (i.e., the set of B-vertices adjacent with either a or c) is approximately $2nq$. Furthermore, the size of the join support T_p of (a, c) (i.e., the number of B-vertices adjacent with both a and c) is about nq^2. So, for a given edge probability q, the U_p/T_p ratio is of the order $2/q$; thus, by varying q, we effectively vary U_p/T_p accordingly.

6.2 Experimental Results

Figure 7 presents some of our preliminary relative-error results for our JDEstimator and LinearJDEstimator algorithms as a function of the number of sketches employed in the underlying synopsis data structures (i.e., `JD` sketches for JDEstimator and FM sketches for LinearJDEstimator). (In the case of JDEstimator, the number of inner 2-level hash sketches is kept fixed at 40.) The numbers shown were obtained over random data graphs generated with $n = 1,000$ and edge probabilities q varying from 0.01 to 0.04. With these parameter settings, the input relation sizes varied between approximately $10,000$ and $40,000$ (distinct) tuples, while the true `Join-Distinct` result cardinality was between approximately $820,000$ (for $q = 0.04$) and $115,000$ (for $q = 0.01$). (Qualitatively similar results were obtained for other parameter settings.)

Our results demonstrate the effectiveness of our probabilistic `Join-Distinct` estimators: in most cases, using only about 100-150 sketches, our algorithms' estimates are within relative errors of about $15 - 20\%$ or lower. The setup with $q = 0.01$ clearly represents a "difficult" case for our JDEstimator algorithm, since it corresponds to a fairly low neighborhood-to-join-support (i.e., U_p/T_p) ratio, making it hard to detect non-empty bucket intersections over the `JD` sketches. Since, as mentioned earlier, our probabilistic intersection estimators guarantee no false negatives, JDEstimator typically ends up underestimating the true `Join-Distinct` cardinality in such difficult setups. As our numbers show, however, our alternative LinearJDEstimator can actually provide an effective alternative even for such "sparse" joins, assuming, of course, that the number of distinct values for the join attribute B remains reasonably small ($|B| = 1,000$ in our example setting).

7. CONCLUSIONS

Estimating the cardinality of `Join-Distinct` expressions over (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 to the general `Join-Distinct` cardinality estimation problem in the data- or update-streaming model. Our proposed estimators rely on novel, hash-based synopsis data structures that can be effectively *composed* to provide low error, high-confidence `Join-Distinct` estimates using only small space and small processing time per update. Preliminary results from an empirical study of our algorithms have substantiated our theoretical claims, showing that our techniques can provide efficient and accurate `Join-Distinct` cardinality estimates over streams.

8. REFERENCES

[1] Noga Alon, Phillip B. Gibbons, Yossi Matias, and Mario Szegedy. "Tracking Join and Self-Join Sizes in Limited Storage". In *Proc. of the 18th ACM Symposium on Principles of Database Systems*, Philadeplphia, Pennsylvania, May 1999.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments". In *Proc. of the 28th Annual ACM Symposium on the Theory of Computing*, pages 20–29, Philadelphia, Pennsylvania, May 1996.

[3] Ziv Bar-Yossef, T.S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. "Counting distinct elements in a data stream". In *Proc. of the 6th Intl. Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM'02)*, Cambridge, Massachusetts, September 2002.

[4] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. "Towards Estimation Error Guarantees for Distinct Values". In *Proc. of the 19th ACM Symposium on Principles of Database Systems*, Dallas, Texas, May 2000.

[5] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *"Introduction to Algorithms"*. MIT Press (The MIT Electrical Engineering and Computer Science Series), 1990.

[6] Alin Dobra, Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. "Processing Complex Aggregate Queries over Data Streams". In *Proc. of the 2002 ACM SIGMOD Intl. Conference on Management of Data*, pages 61–72,

Madison, Wisconsin, June 2002.

[7] Philippe Flajolet and G. Nigel Martin. “Probabilistic Counting Algorithms for Data Base Applications”. *Journal of Computer and Systems Sciences*, 31:182–209, 1985.

[8] Sumit Ganguly, Minos Garofalakis, Amit Kumar, and Rajeev Rastogi. “Join-Distinct Aggregate Estimation over Update Streams”. Bell Labs Technical Memorandum, March 2005.

[9] Sumit Ganguly, Minos Garofalakis, and Rajeev Rastogi. “Tracking Set-Expression Cardinalities over Continuous Update Streams”. *The VLDB Journal*, 13(4):354–369, December 2004. (Special Issue on Data Stream Processing).

[10] Phillip B. Gibbons. “Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports”. In *Proc. of the 27th Intl. Conference on Very Large Data Bases*, Roma, Italy, September 2001.

[11] Phillip B. Gibbons and Srikanta Tirthapura. “Estimating Simple Functions on the Union of Data Streams”. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, Crete Island, Greece, July 2001.

[12] Anna C. Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. “Fast, small-space algorithms for approximate histogram maintenance”. In *Proc. of the 34th Annual ACM Symposium on the Theory of Computing*, Montreal, Quebec, May 2002.

[13] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. “Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries”. In *Proc. of the 27th Intl. Conference on Very Large Data Bases*, Roma, Italy, September 2001.

[14] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. “How to Summarize the Universe: Dynamic Maintenance of Quantiles”. In *Proc. of the 28th Intl. Conference on Very Large Data Bases*, pages 454–465, Hong Kong, China, August 2002.

[15] 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 International Conf. on Very Large Data Bases*, pages 311–322, September 1995.

[16] Piotr Indyk. “Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computation”. In *Proc. of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 189–197, Redondo Beach, California, November 2000.

[17] Bala Kalyanasundaram and Georg Schnitger. “The Probabilistic Communication Complexity of Set Intersection”. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, November 1992.

[18] Eyal Kushilevitz and Noam Nisan. *“Communication Complexity”*. Cambridge University Press, 1997.

[19] Rudolf Lidl and Harald Niederreiter. *“Introduction to Finite Fields and their Applications”*. Cambridge University Press, 1994. (Second Edition).

[20] Jim Melton and Alan R. Simon. *“Understanding the New SQL: A Complete Guide”*. Morgan Kaufmann Publishers, 1993.

[21] Rajeev Motwani and Prabhakar Raghavan. *“Randomized Algorithms”*. Cambridge University Press, 1995.

APPENDIX

A. LOWER-BOUND PROOF

Our proof goes along the ideas of the lower bound shown in [1]. We use Yao's lemma. We exhibit two distributions according to which we select the tuples in relations R and S respectively. Suppose there exits a *deterministic* algorithm Φ which always produces a synopsis of length less than $|A||C|(\deg(A) + \deg(C))/(2\beta)$. Then, we will show that such an algorithm will not be able to output a constant factor (better than 2) estimation of the `Join-Distinct` size for a set of inputs of constant probability measure. Yao's lemma will then imply Theorem 4.4.

Let $m = N - d\sqrt{\beta}$. We first define a distribution D_1 on set of m tuples, each tuple belonging to relation R. Attribute B takes values from the set $\{1, \ldots, t\}$, where we fix the value of t later. We pick a value i from this set uniformly at random. We define m different tuples $r_1, \ldots, r_m$ as follows : $r_k[B] = i$ for $k = 1, \ldots, m$. These tuples are divided into m/d groups, each group being of size d. The tuples r in group k satisfy $r_k[A] = a_k$, where $a_1, a_2, \ldots, a_{m/d}$ are distinct values in A.

We can now define the distribution on the first relation R. R has N tuples. The first m tuples are drawn according to D_1 as described above. The remaining $N - m = d\sqrt{\beta}$ tuples are chosen as follows : $r[B] = 0$ for all these tuples r. As before, we divide these $N - m$ tuples into groups of d each. All tuples r in the same group have the same value of $r[A]$, but tuples in different groups have distinct A-attribute values.

We now define another distribution D_2, which again is a distribution on sets of tuples of size m, each tuple belonging to relation S. We first define a set system $\mathcal{P}$ on $\{1, \ldots, t\}$ as follows. Each set $P \in \mathcal{P}$ has size $t/10$ and if P_1, P_2 are two distinct sets in $\mathcal{P}$, then $|P_1 \cap P_2| \leq t/20$. Further, $\mathcal{P}$ has 2^t sets. The existence of such a set system can be shown by the probabilistic method. We pick a set P from this set system uniformly at random. We define $|P| = t/10$ tuples. The B attributes of these tuples are distinct elements of P. We divide these $t/10$ elements into groups of d elements each. Thus, there are $t/10d$ such groups.

We make $10m/t$ copies of these $t/10$ tuples (thus, there are a total of m tuples). Observe that we have a total of $(t/10d)\cdot(10m/t) = m/d$ groups of these tuples. We associate a unique C attribute with each of these groups.

This gives us a set of m tuples for S. We define another $N - m$ tuples in exactly the same way as we did for relation R. This describes a relation S of size N.

Suppose we are given two relations R and S according to these distributions. The first thing to observe is that the `Join-Distinct` size is either β or $\beta + m^2/td$, depending on whether the value $i \in B$ chosen according to D_1 belongs to the set P chosen in distribution D_2. We set $t = m^2/(d\beta)$. So, the `Join-Distinct` size is either β or 2β. Thus, if we give a wrong answer to the `Join-Distinct` size query, then we are off by a constant factor. Suppose S_1 and S_2 are two relations such that $\Phi(S_1) = \Phi(S_2)$. Then, we claim that our algorithm will give an error on a constant fraction of the values i chosen from the set $\{1, \ldots, t\}$. Indeed, suppose S_1 is defined by the set $P_1 \in \mathcal{P}$ and S_2 is defined by the set $P_2 \in \mathcal{P}$. If $i \notin P_1 \cap P_2$, then the `Join-Distinct` size of R with one of S_1 and S_2 will be β, while it will be 2β with the other one. But, at least $t/20$ values of i satisfy this property. Thus, if S_1 and S_2 are mapped to the same synopsis, then we give errors on a constant fraction of the possible R relations. Now, there are 2^t possible S relations. If Φ always produces a synopsis with at most $t-1$ bits, then most of the possible relations S will have the property that there exists another relation S' such that $\Phi(S) = \Phi(S')$. But then, our algorithm will give an error for a constant fraction of the input. Thus, Φ must use at least $t = m^2/td = m/d \cdot m/d \cdot d \cdot 1/\beta$ bits. This proves the desired result since, clearly, $d = (\deg(A) + \deg(C))/2$ in our instance. ∎