# Processing Complex Aggregate Queries over Data Streams


Alin Dobra* Cornell University dobra@cs.cornell.edu

Minos Garofalakis Bell Labs, Lucent minos@bell-labs.com

Johannes Gehrke Cornell University johannes@cs.cornell.edu

Rajeev Rastogi Bell Labs, Lucent rastogi@bell-labs.com



## ABSTRACT

Recent years have witnessed an increasing interest in designing algorithms for querying and analyzing streaming data (i.e., data that is seen only once in a fixed order) with only limited memory. Providing (perhaps approximate) answers to queries over such continuous data streams is a crucial requirement for many application environments; examples include large telecom and IP network installations where performance data from different parts of the network needs to be continuously collected and analyzed.

In this paper, we consider the problem of approximately answering general *aggregate* SQL queries over continuous data streams with limited memory. Our method relies on randomizing techniques that compute small "sketch" summaries of the streams that can then be used to provide approximate answers to aggregate queries with provable guarantees on the approximation error. We also demonstrate how existing statistical information on the base data (e.g., histograms) can be used in the proposed framework to improve the quality of the approximation provided by our algorithms. The key idea is to intelligently partition the domain of the underlying attribute(s) and, thus, decompose the sketching problem in a way that provably tightens our guarantees. Results of our experimental study with real-life as well as synthetic data streams indicate that sketches provide significantly more accurate answers compared to histograms for aggregate queries. This is especially true when our domain partitioning methods are employed to further boost the accuracy of the final estimates.


## 1. INTRODUCTION

Traditional Database Management Systems (DBMS) software is built on the concept of *persistent data sets*, that are stored reliably in stable storage and queried/updated several times throughout their lifetime. For several emerging application domains, however, data arrives and needs to be processed on a continuous ($24 \times 7$) basis, without the benefit of several passes over a static, persistent data image. Such *continuous data streams* arise naturally, for example, in the network installations of large Telecom and Internet service providers where detailed usage information (Call-Detail-Records (CDRs), SNMP/RMON packet-flow data, etc.) from different parts of the underlying network needs to be continuously collected and analyzed for interesting trends. Other applications that generate rapid, continuous and large volumes of stream data include transactions in retail chains, ATM and credit card operations in banks, financial tickers, Web server log records, etc. In most such applications, the data stream is actually accumulated and archived in the DBMS of a (perhaps, off-site) data warehouse, often making access to the archived data prohibitively expensive. Further, the ability to make decisions and infer interesting patterns *on-line* (i.e., as the data stream arrives) is crucial for several mission-critical tasks that can have significant dollar value for a large corporation (e.g., telecom fraud detection). As a result, recent years have witnessed an increasing interest in designing data-processing algorithms that work over continuous data streams, i.e., algorithms that provide results to user queries while looking at the relevant data items *only once and in a fixed order* (determined by the stream-arrival pattern).

Two key parameters for query processing over continuous data-streams are (1) the amount of *memory* made available to the on-line algorithm, and (2) the *per-item processing time* required by the query processor. The former constitutes an important constraint on the design of stream processing algorithms, since in a typical streaming environment, only limited memory resources are available to the query-processing algorithms. In these situations, we need algorithms that can summarize the data stream(s) involved in a concise, but reasonably accurate, *synopsis* that can be stored in the allotted (small) amount of memory and can be used to provide *approximate answers* to user queries along with some reasonable guarantees on the quality of the approximation. Such approximate, on-line query answers are particularly well-suited to the exploratory nature of most data-stream processing applications such as, e.g., trend analysis and fraud/anomaly detection in telecom-network data, where the goal is to identify generic, interesting or "out-of-the-ordinary" patterns rather than provide results that are exact to the last decimal.

**Prior Work.** The strong incentive behind data-stream computation has given rise to several recent (theoretical and practical) studies of on-line or one-pass algorithms with limited memory requirements for different problems; examples include quantile and order-statistics computation [16, 21], estimating frequency moments and join sizes [3, 2], data clustering and decision-tree construction [10, 18], estimating correlated aggregates [13], and computing one-dimensional (i.e., single-attribute) histograms and Haar wavelet decompositions [17, 15]. Other related studies have proposed techniques for incrementally maintaining equi-depth histograms [14] and Haar wavelets [22], maintaining samples and simple statistics over sliding windows [8], as well as general, high-level architectures for stream database systems [4].

None of the earlier research efforts has addressed the general problem of processing general, possibly multi-join, aggregate queries over continuous data streams. On the other hand, efficient ap-

*Work done while visiting Bell Labs.




*ACM SIGMOD* '2002 June 4-6, Madison, Wisconsin, USA

proximate multi-join processing has received considerable attention in the context of *approximate query answering*, a very active area of database research in recent years [1, 6, 12, 19, 20, 24]. The vast majority of existing proposals, however, rely on the assumption of a static data set which enables either several passes over the data to construct effective, *multi-dimensional* data synopses (e.g., histograms [20] and Haar wavelets [6, 24]) or intelligent strategies for randomizing the access pattern of the relevant data items [19]. When dealing with continuous data streams, it is crucial that the synopsis structure(s) are constructed directly on the stream, that is, in one pass over the data in the fixed order of arrival; this requirement renders conventional approximate query processing tools inapplicable in a data-stream setting. (Note that, even though random-sample data summaries can be easily constructed in a single pass [23], it is well known that such summaries typically give very poor result estimates for queries involving one or more joins [1, 6, 2][1]).

**Our Contributions.** In this paper, we tackle the hard technical problems involved in the approximate processing of complex (possibly multi-join) aggregate decision-support queries over continuous data streams with limited memory. Our approach is based on randomizing techniques that compute small, pseudo-random *sketch* summaries of the data as it is streaming by. The basic sketching technique was originally introduced for on-line self-join size estimation by Alon, Matias, and Szegedy in their seminal paper [3] and, as we demonstrate in our work, can be generalized to provide approximate answers to complex, multi-join, aggregate SQL queries over streams with explicit and tunable guarantees on the approximation error. An important practical concern that arises in the multi-join context is that the quality of the approximation may degrade as the variance of our randomized sketch synopses increases in an explosive manner with the number of joins involved in the query. To this end, we propose novel *sketch-partitioning* techniques that take advantage of existing approximate statistical information on the stream (e.g., histograms built on archived data) to decompose the sketching problem in a way that provably tightens our estimation guarantees. More concretely, the key contributions of our work are summarized as follows.

• SKETCH-BASED APPROXIMATE PROCESSING ALGORITHMS FOR COMPLEX AGGREGATE QUERIES. We show how small, sketch synopses for data streams can be used to compute provably-accurate approximate answers to aggregate multi-join queries. Our techniques extend and generalize the earlier results of Alon et al. [3, 2] in two important respects. First, our algorithms provide probabilistic accuracy guarantees for queries containing any number of relational joins. Second, we consider a wide range of aggregate operators (e.g., `COUNT`, `SUM`) rather than just simple `COUNT` aggregates. We should also point out that our error-bound derivation for multi-join queries is non-trivial and requires that certain acyclicity restrictions be imposed on the query's join graph.

• SKETCH-PARTITIONING ALGORITHMS TO BOOST ESTIMATION ACCURACY. We demonstrate that (approximate) statistics (e.g., histograms) on the distributions of join-attribute values can be used to reduce the variance in our randomized answer estimate, which is a function of the self-join sizes of the base stream relations. Thus, we propose novel *sketch-partitioning* techniques that exploit such statistics to significantly boost the accuracy of our approximate answers by (1) intelligently partitioning the attribute domains so that the self-join sizes of the resulting partitions are minimized, and (2) judiciously allocating space to independent sketches for each partition. For single-join queries, we develop a sketch-partitioning algorithm that exploits a theorem of Breiman et al. [5] to compute an *optimal* solution, that is provably near-optimal for minimizing the estimate variance. We also present bounds on the error in the final answer as a function of the error in the underlying statistics (used to compute the partitioning). Unfortunately, for queries with more than one join, we demonstrate that the sketch-partitioning problem is $\mathcal{NP}$-hard. Thus, we introduce a partitioning heuristic for multi-joins that can, in fact, be shown to produce near-optimal solutions if the underlying attribute-value distributions are independent.

• EXPERIMENTAL RESULTS VALIDATING OUR SKETCH-BASED TECHNIQUES. We present the results of an experimental study with several real-life and synthetic data sets over a wide range of queries that verify the effectiveness of our sketch-based approach to complex stream-query processing. Specifically, our results indicate that compared to on-line histogram-based methods, sketching can give much more accurate answers that are often superior by factors ranging from three to an order of magnitude. Our experiments also demonstrate that our sketch-partitioning algorithms result in significant reductions in the estimation error (almost a factor of two), even when coarse histogram statistics are employed to select the join-attribute partitions.

Note that, even though we develop our sketching algorithms in the data-stream context, our techniques are more generally applicable to huge Terabyte databases where performing multiple passes over the data for the exact computation of query results may be prohibitively expensive. Our sketch-partitioning algorithms are, in fact, ideal for such "huge database" environments, where an initial pass over the data can be used to compute random samples, approximate histograms, or other statistics which can subsequently be used as the basis for determining the sketch partitions.

## 2. STREAMS AND RANDOM SKETCHES

### 2.1 The Stream Data-Processing Model

We now briefly describe the key elements of our generic architecture for query processing over continuous data streams (depicted in Figure 1); similar architectures for stream processing have been described elsewhere (e.g., [4, 15]). Consider an arbitrary (possibly complex) SQL query $Q$ over a set of relations $R_1, \ldots, R_r$ and let $|R_i|$ denote the total number of tuples in $R_i$. (Extending our architecture to handle multiple queries is straightforward, although interesting research issues, e.g., inter-query space allocation, do arise; we will not consider such issues further in this paper.) In contrast to conventional DBMS query processors, our stream query-processing engine is allowed to see the data tuples in $R_1, \ldots, R_n$ *only once* and in fixed order as they are streaming in from their respective source(s). Backtracking over the data stream and explicit access to past data tuples are impossible. Further, the order of tuple arrival for each relation $R_i$ is arbitrary and duplicate tuples can occur anywhere over the duration of the $R_i$ stream. Hence, our stream data model assumes the most general "unordered, cash-register" rendition of stream data considered by Gilbert et al. [15] for computing one-dimensional Haar wavelets over streaming values and, of course, generalizes their model to multiple, multi-dimensional streams since each $R_i$ can comprise several distinct attributes.

Our stream query-processing engine is also allowed a certain amount of memory, typically significantly smaller than the total size of the data set(s). This memory is used to maintain a concise and accurate *synopsis* of each data stream $R_i$, denoted by $\mathcal{S}(R_i)$. The key constraints imposed on each synopsis $\mathcal{S}(R_i)$ are that (1) it is much smaller than the total number of tuples in $R_i$ (e.g., its size is

[1]The sampling-based *join synopses* of [1] provide a solution to this problem but only for the special case of *static, foreign-key joins*.

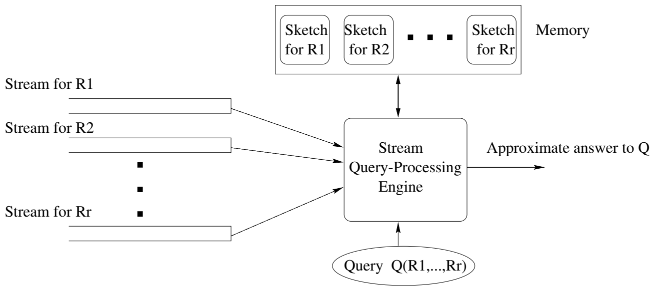


**Figure 1: Stream Query-Processing Architecture.**

logarithmic or polylogarithmic in $|R_i|$), and (2) it can be computed in a single pass over the data tuples in $R_i$ in the (arbitrary) order of their arrival. At any point in time, our query-processing algorithms can combine the maintained synopses $\mathcal{S}(R_1), \ldots, \mathcal{S}(R_n)$ to produce an approximate answer to query $Q$.

## 2.2 Pseudo-Random Sketch Summaries

**The Basic Technique: Self-Join Size Tracking.** Consider a simple stream-processing scenario where the goal is to estimate the size of the self-join of relation $R$ over one of its attributes $R.A$ as the tuples of $R$ are streaming in; that is, we seek to approximate the result of query $Q = \texttt{COUNT}(R \bowtie_A R)$. Letting $\texttt{dom}(A)$ denote the domain of the join attribute[2] and $f(i)$ be the frequency of attribute value $i$ in $R.A$, we want to produce an estimate for the expression $\text{SJ}(A) = \sum_{i \in \texttt{dom}(A)} f(i)^2$ (i.e., the *second moment* of $A$). In their seminal paper, Alon, Matias, and Szegedy [3] prove that *any deterministic algorithm* that produces a tight approximation to $\text{SJ}(A)$ requires at least $\Omega(|\texttt{dom}(A)|)$ bits of storage, rendering such solutions impractical for a data-stream setting. Instead, they propose a *randomized technique* that offers strong probabilistic guarantees on the quality of the resulting $\text{SJ}(A)$ approximation while using only logarithmic space in $|\texttt{dom}(A)|$. Briefly, the basic idea of their scheme is to define a random variable $Z$ that can be easily computed over the streaming values of $R.A$, such that (1) $Z$ is an *unbiased* (i.e., correct on expectation) estimator for $\text{SJ}(A)$, so that $E[Z] = \text{SJ}(A)$; and (2) $Z$ has sufficiently small variance $\text{Var}(Z)$ to provide strong probabilistic guarantees for the quality of the estimate. This random variable $Z$ is constructed on-line from the streaming values of $R.A$ as follows:

- Select a family of *four-wise independent binary random variables* $\{\xi_i : i = 1, \ldots, |\texttt{dom}(A)|\}$, where each $\xi_i \in \{-1, +1\}$ and $P[\xi_i = +1] = P[\xi_i = -1] = 1/2$ (i.e., $E[\xi_i] = 0$). Informally, the four-wise independence condition means that for any 4-tuple of $\xi_i$ variables and for any 4-tuple of $\{-1, +1\}$ values, the probability that the values of the variables coincide with those in the $\{-1, +1\}$ 4-tuple is exactly $1/16$ (the product of the equality probabilities for each individual $\xi_i$). The crucial point here is that, by employing known tools (e.g., orthogonal arrays) for the explicit construction of small sample spaces supporting four-wise independent random variables, such families can be efficiently constructed on-line using only $O(\log |\texttt{dom}(A)|)$ space [3].
- Define $Z = X^2$, where $X = \sum_{i \in \texttt{dom}(A)} f(i)\xi_i$. Note that $X$ is simply a randomized linear projection (inner product) of the frequency vector of $R_i.A$ with the vector of $\xi_i$'s that can be efficiently generated from the streaming values of $A$ as follows: Start with $X = 0$ and simply add $\xi_i$ to $X$ whenever the $i^{th}$ value of $A$ is observed in the stream.

[2] Without loss of generality, we assume that each attribute domain $\texttt{dom}(A)$ is indexed by the set of integers $\{0, 1, \cdots, |\texttt{dom}(A)| - 1\}$, where $|\texttt{dom}(A)|$ denotes the size of the domain.

To further improve the quality of the estimation guarantees, Alon, Matias, and Szegedy propose a standard *boosting technique* that maintains several independent identically-distributed (iid) instantiations of such random variables $Z$ and uses averaging and median-selection operators to boost accuracy and probabilistic confidence. (Independent instances can be constructed by simply selecting independent random seeds for generating the families of four-wise independent $\xi_i$'s for each instance.) More specifically, the synopsis $\mathcal{S}(R_i)$ comprises $s = s_1 \cdot s_2$ randomized linear-projection variables $X_{ij}$, where $s_1$ is a parameter that determines the *accuracy* of the result and $s_2$ determines the *confidence* in the estimate. The final boosted estimate $Y$ of $\text{SJ}(A)$ is the median of $s_2$ random variables $Y_1, \ldots, Y_{s_2}$, each $Y_i$ being the average of $s_1$ iid random variables $X_{ij}^2$, $j = 1, \ldots, s_1$, where each $X_{ij}$ uses the same on-line construction as the variable $X$ (described above). The averaging step is used to reduce the variance, and hence the estimation error (by Chebyshev's inequality), and median-selection is used to boost the confidence in the estimate (by Chernoff bounds). We use the term *atomic sketch* to describe each randomized linear projection $X_{ij}$ of the data stream and the term *sketch* for the overall synopsis $\mathcal{S}$. The following theorem [3] demonstrates that the sketch-based method offers strong probabilistic guarantees for the second-moment estimate while utilizing only logarithmic space in the number of distinct $R.A$ values and the length of the stream.

THEOREM 2.1 ([3]). *The estimate $Y$ computed by the above algorithm satisfies: $P[|Y - SJ(A)| \leq 4/\sqrt{s_1} SJ(A)] \geq 1 - 2^{-s_2/2}$. This implies that the algorithm estimates $SJ(A)$ in one pass with a relative error of at most $\epsilon$ with probability at least $1 - \delta$ (i.e., $P[|Y - SJ(A)| \leq \epsilon \cdot SJ(A)] \geq 1 - \delta$) while using only $O\left(\frac{\log(1/\delta)}{\epsilon^2}(\log|\texttt{dom}(A)| + \log|R|)\right)$ bits of memory.* ∎

**Extensions: Binary Joins, Wavelets, and $L^p$ Differencing.** In a more recent paper, Alon et al. [2] demonstrate how the above algorithm can be extended to deal with deletions in the data stream and demonstrate its benefits experimentally over naive solutions based on random sampling. They also show how their sketch-based approach applies to handling the size-estimation problem for binary joins over a pair of distinct tuple streams. More specifically, consider approximating the result of the query $Q = \texttt{COUNT}(R_1 \bowtie_{R_1.A_1 = R_2.A_2} R_2)$ over two relational streams $R_1$ and $R_2$. (Note that, by the definition of the equi-join operation, the two join attributes have identical value domains, i.e., $\texttt{dom}(A_1) = \texttt{dom}(A_2)$.) As previously, let $\{\xi_i : i = 1, \ldots, |\texttt{dom}(A_1)|\}$ be a family of four-wise independent $\{-1, +1\}$ random variables with $E[\xi_i] = 0$, and define the randomized linear projections $X_1 = \sum_{i \in \texttt{dom}(A_1)} f_1(i)\xi_i$ and $X_2 = \sum_{i \in \texttt{dom}(A_2)} f_2(i)\xi_i$, where $f_1(i)$, $f_2(i)$ represent the frequencies of $R_1.A_1$ and $R_2.A_2$ values, respectively. The following theorem [2] shows how sketching can be applied for estimating binary-join sizes in limited space.

THEOREM 2.2 ([2]). *Let the atomic sketches $X_1$ and $X_2$ be as defined above. Then $E[X_1 X_2] = |R_1 \bowtie_{A_1 = A_2} R_2|$ and $Var(X_1 X_2) \leq 2 \cdot SJ_1(A_1) \cdot SJ_2(A_2)$, where $SJ_1(A_1)$, $SJ_2(A_2)$ is the self-join size of $R_1.A_1$ and $R_2.A_2$, respectively. Thus, averaging over $k = O(SJ_1(A_1) SJ_2(A_2)/(\epsilon^2 L^2))$ iid instantiations of the basic scheme, where $L$ is a lower bound on the join size, guarantees an estimate that lies within constant relative error $\epsilon$ of $|R_1 \bowtie_{A_1 = A_2} R_2|$ with high probability.* ∎

Techniques relying on the same basic idea of compact, pseudo-random sketching have also been proposed recently for other data-stream applications. Gilbert et al. [15] propose the use of sketches for approximately computing *one-dimensional* Haar wavelet coefficients and range aggregates over streaming numeric values. Strauss et al. [11] discuss sketch-based techniques for the on-line estimation of $L^1$ differences between two numeric data streams. None of these earlier studies, however, has considered the hard technical problems involved in using sketching to effectively approximate the results of complex, multi-join aggregate SQL queries over multiple massive data streams.

# 3. APPROXIMATING COMPLEX QUERY ANSWERS USING STREAM SKETCHES

In this section, we describe our sketch-based techniques for computing guaranteed-quality approximate answers to general aggregate operators over complex, multi-join SQL queries spanning multiple streaming relations $R_1, \ldots, R_r$. More specifically, the class of queries that we consider is of the general form: "SELECT AGG FROM $R_1, R_2, \ldots, R_r$ WHERE $\mathcal{E}$", where AGG is an arbitrary aggregate operator (e.g., COUNT, SUM or AVERAGE) and $\mathcal{E}$ represents the conjunction of of $n$ equi-join constraints[3] of the form $R_i.A_j = R_k.A_l$ ($R_i.A_j$ denotes the $j^{th}$ attribute of relation $R_i$).

We first demonstrate how sketches can provide approximate answers with probabilistic quality guarantees to COUNT aggregates, and then show how our results can be generalized to other aggregation operators like SUM. In order to derive probabilistic guarantees on the estimation error, we require that each attribute belonging to a relation appears *at most once* in the join conditions $\mathcal{E}$. Note that this is not a serious restriction, as any set of join conditions can be transformed to satisfy our requirement, as follows. For any attribute $R_i.A_j$ that occurs $m > 1$ times in $\mathcal{E}$, we add $m-1$ new "attributes" to $R_i$, and replace $m-1$ occurrences of $R_i.A_j$ in $\mathcal{E}$, each with a distinct new attribute. These new $m-1$ attributes are exact replicas of $R_i.A_j$, so they all take on values identical to $R_i.A_j$ within each tuple of $R_i$. For instance, if $\mathcal{E} = ((R_1.A_1 = R_2.A_1)$ AND $(R_1.A_1 = R_3.A_1))$, we can modify it to satisfy our our *single attribute-occurrence constraint* by adding a new attribute $A_2$ to $R_1$ which is a replica of $A_1$, and replacing an occurrence of $R_1.A_1$ so that, for example $\mathcal{E} = ((R_1.A_1 = R_2.A_1)$ AND $(R_1.A_2 = R_3.A_1))$. Clearly, this addition of new "attributes" can be carried out only at a conceptual level, e.g., as part of our sketch-computation logic. We assume that $\mathcal{E}$ satisfies our single attribute-occurrence constraint in the remainder of this section.

## 3.1 Using Sketches to Answer COUNT Queries

The output of a COUNT query $Q_{\texttt{COUNT}}$ is the number of tuples in the cross-product of $R_1, \ldots, R_r$ that satisfy the equality constraints in $\mathcal{E}$ over the join attributes. Assume a renaming of the $2n$ join attributes in $\mathcal{E}$ to $A_1, A_2, \ldots, A_{2n}$ such that each equi-join constraint in $\mathcal{E}$ is of the form $A_j = A_{n+j}$, for $1 \leq j \leq n$. Let $\texttt{dom}(A_i) = \{1, \ldots, |\texttt{dom}(A_i)|\}$ be the domain of attribute $A_i$, and $\mathcal{D} = \texttt{dom}(A_1) \times \cdots \times \texttt{dom}(A_{2n})$. Also, let $S_k$ denote the subset of (renamed) attributes from relation $R_k$ appearing in $\mathcal{E}$ and let $\mathcal{D}_k = \texttt{dom}(A_{k_1}) \times \cdots \times \texttt{dom}(A_{k_{|S_k|}})$, where $A_{k_1}, \ldots, A_{k_{|S_k|}}$ are the attributes in $S_k$. An assignment $\mathcal{I}$ assigns values to join attributes from their respective domains. If $\mathcal{I} \in \mathcal{D}$, then each join attribute $A_j$ is assigned a value $\mathcal{I}[j]$ by $\mathcal{I}$. On the other hand, if $\mathcal{I} \in \mathcal{D}_k$, then $\mathcal{I}$ only assigns a value $\mathcal{I}[j]$ to attributes $j \in S_k$. (Henceforth, we will simply use $j$ to refer to attribute $A_j$ when the distinction is clear from the context.) We use $\mathcal{I}[S_k]$ to denote the projection of $\mathcal{I}$ on attributes in $S_k$; note that $\mathcal{I}[S_k] \in \mathcal{D}_k$. Finally, for $\mathcal{I} \in \mathcal{D}_k$, we use $f_k(\mathcal{I})$ to denote the number of tuples in $R_k$ whose value for attribute $j$ equals $\mathcal{I}[j]$ for all $j \in S_k$. Table 1 summarizes some of the key notational conventions used throughout the paper; additional notation will be introduced when necessary.

[3] Simple value-based selections on individual relations are trivial to evaluate over the streaming tuples.

| Symbol | Description |
|---|---|
| $R_1, \ldots, R_r$ | Relations in aggregate query |
| $A_1, \ldots, A_{2n}$ | Attributes over which join is defined |
| $\texttt{dom}(A_j)$ | Domain of attribute $A_j$ |
| $\mathcal{D}$ | $\texttt{dom}(A_1) \times \cdots \times \texttt{dom}(A_{2n})$ |
| $S_k$ | Join attributes in relation $R_k$ |
| $\mathcal{D}_k$ | Projection of $\mathcal{D}$ on attributes in $S_k$ |
| $\text{SJ}_k(S_k)$ | Self-join of relation $R_k$ on attributes in $S_k$ |
| $\mathcal{I}$ | Assignment of values to (a subset of) join attributes |
| $\mathcal{I}[j]$ | Value assigned to attribute $j$ |
| $\mathcal{I}[S_k]$ | Projection of I on attributes in $S_k$ |
| $f_k(\mathcal{I})$ | Number of tuples in $R_k$ that match $\mathcal{I}$ |
| $X_k$ | Atomic sketch for relation $R_k$ |
| $\{\xi_{jl} : l = 1, \ldots, \lvert\texttt{dom}(A_j)\rvert\}$ | Family of four-wise independent random variables for attribute $A_j$ |

**Table 1: Notation.**

The result of our COUNT query can now be expressed as $Q_{\texttt{COUNT}} = \sum_{\mathcal{I} \in \mathcal{D}, \forall j: \mathcal{I}[j] = \mathcal{I}[n+j]} \prod_{k=1}^{r} f_k(\mathcal{I}[S_k])$. This is essentially the product of the number of tuples in each relation that match a value assignment $\mathcal{I}$, summed over all assignments $\mathcal{I} \in \mathcal{D}$ that satisfy the equi-join constraints $\mathcal{E}$. Our sketch-based randomized algorithm for producing a probabilistic estimate of the result of a COUNT query is similar in spirit to the technique originally proposed in [3] and described in Section 2. Essentially, we construct a random variable $X$ that is an unbiased estimator for $Q_{\texttt{COUNT}}$ (i.e., $E[X] = Q_{\texttt{COUNT}}$), and whose variance can be appropriately bounded from above. Then, by employing the standard averaging and median-selection trick of [3], we boost the accuracy and confidence of $X$ to compute an estimate of $Q_{\texttt{COUNT}}$ that guarantees small relative error with high probability.

We now show how such a random variable $X$ can be constructed. For each pair of join attributes $j, n+j$ in $\mathcal{E}$, we build a family of four-wise independent random variables $\{\xi_{j,l} : l = 1, \ldots, |\texttt{dom}(A_j)|\}$, where each $\xi_{j,l} \in \{-1, +1\}$. The key here is that an every equi-join attribute pair $j$ and $n+j$ shares the same $\xi$ family, and so for all $l \in \texttt{dom}(A_j)$, $\xi_{j,l} = \xi_{n+j,l}$; however, we define a distinct $\xi$ family for each of the $n$ distinct equi-join pairs using mutually-independent random seeds to generate each $\xi$ family. Thus, random variables belonging to families defined for different attribute pairs are completely independent of each other. Since, as mentioned earlier, the family for attribute pair $j, n+j$ can be efficiently constructed on-line using only $O(\log|\texttt{dom}(A_j)|)$ space, the space requirements for all $n$ families of random variables is $\sum_{j=1}^{n} O(\log|\texttt{dom}(A_j)|)$.

For each relation $R_k$, we define the atomic sketch for $R_k$, $X_k$ to be equal to $\sum_{\mathcal{I} \in \mathcal{D}_k} (f_k(\mathcal{I}) \prod_{j \in S_k} \xi_{j,\mathcal{I}[j]})$, and define the COUNT estimator random variable as $X = \prod_{k=1}^{r} X_k$ (i.e., the product of the atomic relation sketches $X_k$). Note that each atomic sketch $X_k$ can be efficiently computed as tuples of $R_k$ are streaming in; more specifically, $X_k$ is initialized to 0 and, for each tuple $t$ in the $R_k$ stream, the quantity $\prod_{j \in S_k} \xi_{j,t[j]}$ is added to $X_k$, where $t[j]$ denotes the value of attribute $j$ in tuple $t$.

**Example 1:** Consider the following COUNT query over relations $R_1$, $R_2$ and $R_3$: SELECT COUNT(*) FROM $R_1, R_2, R_3$ WHERE $R_1.A_1 = R_2.A_1$ AND $R_2.A_2 = R_3.A_1$. After renaming, we get

$A_1 = R_1.A_1$, $A_2 = R_2.A_2$, $A_3 = R_2.A_1$, and $A_4 = R_3.A_1$. The first join involves attributes $A_1$ and $A_3$, while the second is on attributes $A_2$ and $A_4$. Thus, we define two families of four-wise independent random variables (one for each join pair): $\{\xi_{1,l} : l = 1, \ldots, |\mathtt{dom}(A_1)|\}$ and $\{\xi_{2,l} : l = 1, \ldots, |\mathtt{dom}(A_2)|\}$. Three separate atomic sketches $X_1$, $X_2$ and $X_3$ are maintained for the three relations, and are defined as follows: $X_1 = \sum_{t \in R_1} \xi_{1,t[1]}$, $X_2 = \sum_{t \in R_2} \xi_{1,t[3]}\xi_{2,t[2]}$, and $X_3 = \sum_{t \in R_3} \xi_{2,t[4]}$. The value of the random variable $X = X_1 X_2 X_3$ gives our final estimate for the result of the COUNT query. ∎

As the following lemma shows, the random variable $X = \prod_{k=1}^{r} X_k = \prod_{k=1}^{r} \sum_{\mathcal{I} \in \mathcal{D}_k} (f_k(\mathcal{I}) \prod_{j \in S_k} \xi_{j,\mathcal{I}[j]})$ is indeed an *unbiased* estimator for our COUNT aggregate.

LEMMA 3.1. *The random variable $X = \prod_{k=1}^{r} X_k$ is an unbiased estimator for $Q_{\mathtt{COUNT}}$; that is $E[X] = Q_{\mathtt{COUNT}}$.* ∎

As in traditional query processing, the *join graph* for our input query $Q_{\mathtt{COUNT}}$ is defined as an undirected graph consisting of a node for each relation $R_i$, $i = 1, \ldots, r$, and an edge for each join-attribute pair $j, n+j$ between the relation nodes containing the join attributes $j$ and $n+j$. Our computation of tight upper and lower bounds on the variance of $X$ relies on the assumption that the join graph for $Q_{\mathtt{COUNT}}$ is *acyclic*. Thus, the probabilistic quality guarantees provided by our techniques are valid only for acyclic multi-join queries. This is not a serious limitation, since many SQL join queries encountered in database practice are in fact acyclic; this includes chain joins (see Example 3.1) as well as star joins (the dominant form of queries over the star/snowflake schemas of modern data warehouses [7]). Under this acyclicity assumption, the following lemma bounds the variance of our unbiased estimator $X$ for $Q_{\mathtt{COUNT}}$. To simplify the statement of our result, let $\mathrm{SJ}_k(S_k) = \sum_{\mathcal{I} \in \mathcal{D}_k} f_k(\mathcal{I})^2$ denote the size of the self-join of relation $R_k$ over all attributes in $S_k$.

LEMMA 3.2. *Assume that the join graph for $Q_{\mathtt{COUNT}}$ is* acyclic. *Then, for the random variable $X = \prod_{k=1}^{r} X_k$:*

$$\prod_{k=1}^{r} SJ_k(S_k) - \sum_{\mathcal{I} \in \mathcal{D}, \mathcal{I}[j] = \mathcal{I}[n+j]} \prod_{k=1}^{r} f_k(\mathcal{I}[S_k])^2 \leq \quad Var(X)$$

$$\leq ((2^n - 1)^2 + 1) \left( \prod_{k=1}^{r} SJ_k(S_k) - \sum_{\mathcal{I} \in \mathcal{D}, \mathcal{I}[j] = \mathcal{I}[n+j]} \prod_{k=1}^{r} f_k(\mathcal{I}[S_k]) \right)^2$$

∎

The final estimate $Y$ for $Q_{\mathtt{COUNT}}$ is chosen to be the median of $s_2$ random variables $Y_1, \ldots, Y_{s_2}$, each $Y_i$ being the average of $s_1$ iid random variables $X_{ij}$, $1 \leq j \leq s_1$, where each $X_{ij}$ is constructed on-line in a manner identical to the construction of $X$ above. Thus, the total size of our sketch synopsis for $Q_{\mathtt{COUNT}}$ is $O(s_1 \cdot s_2 \cdot \sum_{j=1}^{n} \log |\mathtt{dom}(A_j)|)$[4]. The values of $s_1$ and $s_2$ for achieving a certain degree of accuracy with high probability are derived based on the following theorem that summarizes our results in this section.

THEOREM 3.1. *Let $Q_{\mathtt{COUNT}}$ be an acyclic, multi-join* COUNT *query over relations $R_1, \ldots, R_r$, such that $Q_{\mathtt{COUNT}} \geq L$ and $SJ_k(S_k) \leq U_k$. Then, using a sketch of size $O(\frac{2^{2n} (\prod_{k=1}^{r} U_k) \log(1/\delta)}{L^2 \epsilon^2} \sum_{j=1}^{n} \log |\mathtt{dom}(A_j)|)$, it is possible to approximate $Q_{\mathtt{COUNT}}$ so that the relative error of the estimate is at most $\epsilon$ with probability at least $1 - \delta$.* ∎

[4]Note that this includes the $s_1 \cdot s_2 \cdot (n+1)$ space required for storing the $s_1 \cdot s_2 \cdot r\ X_{ij}$ variables for the $r = n+1$ relations.

## 3.2 Using Sketches to Answer SUM Queries

Our sketch-based approach for approximating complex COUNT aggregates can also be extended to compute approximate answers for complex queries with other aggregate functions, like SUM, over relation streams. A SUM query has the form SELECT SUM($R_i.A_j$) FROM $R_1, R_2, \ldots, R_r$ WHERE $\mathcal{E}$. As earlier, let $A_1, \ldots, A_{2n}$ be a renaming of the $2n$ join attributes in $\mathcal{E}$ and, without loss of generality , let $R_i = R_1$ and $A_{2n+1}$ denote the attribute in $R_1$ whose value is summed in the join result. Further, for an assignment of values $\mathcal{I} \in \mathcal{D}_1$ to all the join attributes in $R_1$, let SUM($\mathcal{I}$) = $\sum_{t \in R_1, \forall j \in S_1 : t[j] = \mathcal{I}[j]} t[A_{2n+1}]$; thus, SUM($\mathcal{I}$) is basically the sum of the values taken by attribute $A_{2n+1}$ in all tuples $t$ in $R_1$ that match $\mathcal{I}$ on the join attributes $S_1$. The result of our SUM query is a scalar quantity $Q_{\mathtt{SUM}}$ whose value can be expressed as $\sum_{\mathcal{I} \in \mathcal{D}, \forall j : \mathcal{I}[j] = \mathcal{I}[n+j]} \mathtt{SUM}(\mathcal{I}[S_1]) \cdot \prod_{k=2}^{r} f_k(\mathcal{I}[S_k])$.

Similar to the COUNT case, in order to approximate $Q_{\mathtt{SUM}}$ over a data stream, we utilize families of four-wise independent random variables $\xi$ to build atomic sketches $X_k$ for each relation, using distinct, independent $\xi$ families for each pair of join attributes. The atomic sketches $X_k$ for $k = 2, \ldots, X_k$ are also defined as described earlier for COUNT queries; that is, $X_k = \sum_{\mathcal{I} \in \mathcal{D}_k} (f_k(\mathcal{I}) \prod_{j \in S_k} \xi_{j,\mathcal{I}[j]})$. However, for the relation $R_1$ containing the SUM attribute, $X_1$ is defined in a slightly different manner as $X_1 = \sum_{\mathcal{I} \in \mathcal{D}_1} (\mathtt{SUM}(\mathcal{I}) \prod_{j \in S_1} \xi_{j,\mathcal{I}[j]})$. Note that $X_1$ can be efficiently maintained on the streaming tuples of $R_1$ by simply adding the quantity $t[A_{2n+1}] \cdot \prod_{j \in S_1} \xi_{j,t[j]}$ for each incoming $R_1$ tuple $t$. Using arguments similar to those in Lemmas 3.1 and 3.2, the random variable $X = \prod_{k=1}^{r} X_k$ can be shown to have an expected value of $Q_{\mathtt{SUM}}$, and (assuming an acyclic join graph) a variance that is bounded by terms similar to those in Lemma 3.2 [9]. These results can be used to build sketch synopses for $Q_{\mathtt{SUM}}$ with probabilistic accuracy guarantees similar to those stated in Theorem 3.1.

# 4. IMPROVING ANSWER QUALITY: SKETCH PARTITIONING

In the proof of Theorem 3.1, to ensure an upper bound of $\epsilon$ on the relative error of our estimate for $Q_{\mathtt{COUNT}}$ with high probability we require that, for each $i$, $\mathrm{Var}(Y_i) \leq \frac{\epsilon^2 L^2}{8}$; this is achieved by defining each $Y_i$ as the average of $s_1$ iid instances of the atomic-sketch estimator $X$, so that $\mathrm{Var}(Y_i) = \frac{\mathrm{Var}(X)}{s_1}$. Then, since by Lemma 3.2, $\mathrm{Var}(X) \leq 2^{2n} \cdot \prod_{k=1}^{r} \mathrm{SJ}_k(S_k)$, averaging over $s_1 \geq \frac{2^{2n+3} \cdot \prod_{k=1}^{r} \mathrm{SJ}_k(S_k)}{\epsilon^2 L^2}$ iid copies of $X$, allows us to guarantee the required upper bound on the variance of $Y_i$. An important practical concern for multi-join queries is that (as is evident from Lemma 3.2) our upper bound on the $\mathrm{Var}(X)$ and, therefore, the number of $X$ instances $s_1$ required to guarantee a given level of accuracy increases explosively with the number of joins $n$ in the query.

To deal with this problem, in this section, we propose novel *sketch-partitioning* techniques that exploit approximate statistics on the streams to decompose the sketching problem in a way that provably tightens our estimation guarantees. The basic idea is that, by intelligently partitioning the domain of join attributes in the query and estimating portions of $Q_{\mathtt{COUNT}}$ individually on each partition, we can significantly reduce the storage (i.e., number of iid $X$ copies) required to approximate each $Y_i$ within a given level of accuracy. (Of course, our sketch-partitioning results are equally applicable to the dual optimization problem; that is, maximizing the estimation accuracy for a given amount of sketching space.) Our techniques can also be extended in a natural way to other aggregation operators (e.g., SUM, VARIANCE) similar to the generalization described in Section 3.2.

The key observation we make is that, given a desired level of accuracy, the number of required iid copies of $X$, is proportional to the *product of the self-join sizes* of relations $R_1, \ldots, R_r$ over the join attributes (Theorem 3.1). Further, in practice, join-attribute domains are frequently skewed and the skew is often concentrated in different regions for different attributes. As a consequence, we can exploit approximate knowledge of the data distribution(s) to intelligently partition the domains of (some subset of) join attributes so that, for each resulting partition $\mathbf{p}$ of the combined attribute space, the product of self-join sizes of relations restricted to $\mathbf{p}$ is very small compared to the same product over the entire (unpartitioned) attribute space (i.e., $\prod_{k=1}^{r} \mathrm{SJ}_k(S_k)$). Thus, letting $X_{\mathbf{p}}$ denote an atomic-sketch estimator for the portion of $Q_{\texttt{COUNT}}$ that corresponds to partition $\mathbf{p}$ of the attribute space, we can expect the variance $\mathrm{Var}(X_{\mathbf{p}})$ to be much smaller than $\mathrm{Var}(X)$.

Now, consider a scheme that averages over $s_{\mathbf{p}}$ iid instances of the atomic sketch $X_{\mathbf{p}}$ for partition $\mathbf{p}$, and defines each $Y_i$ as the sum of these averages over all partitions $\mathbf{p}$. We can then show that $E[Y_i] = Q_{\texttt{COUNT}}$ and $\mathrm{Var}(Y_i) = \sum_{\mathbf{p}} \frac{\mathrm{Var}(X_{\mathbf{p}})}{s_{\mathbf{p}}}$. Clearly, achieving small self-join sizes and variances $\mathrm{Var}(X_{\mathbf{p}})$ for the attribute-space partitions $\mathbf{p}$ means that the total number of iid sketch instances $\sum_{\mathbf{p}} s_{\mathbf{p}}$ required to guarantee that $\mathrm{Var}(Y_i) \leq \frac{\epsilon^2 L^2}{8}$ is also small; this, in turn, implies a smaller storage requirement for the prescribed accuracy level of our $Y_i$ estimator[5]. We formalize the above intuition in the following subsection and then present our sketch-partitioning results and algorithms for both single- and multi-join queries.

## 4.1 Our General Technique

Consider once again the $Q_{\texttt{COUNT}}$ aggregate query (Section 3). In general, our sketch-partitioning techniques partition the domain of each join attribute $A_j$ into $m_j \geq 1$ disjoint subsets denoted by $P_{j,1}, \ldots, P_{j,m_j}$. Further, the domains of a join-attribute pair $A_j$ and $A_{n+j}$ are partitioned identically (note that $\texttt{dom}(A_j) = \texttt{dom}(A_{n+j})$). This partitioning on individual attributes induces a partitioning of the combined (multi-dimensional) join-attribute space, which we denote by $\mathcal{P}$. Thus, $\mathcal{P} = \{(P_{1,l_1}, \ldots, P_{n,l_n}) : 1 \leq l_j \leq m_j\}$. Each element $\mathbf{p} \in \mathcal{P}$ identifies a unique partition of the global attribute space, and we represent by $\mathcal{D}_{\mathbf{p}}$ the restriction of this global attribute space $\mathcal{D}$ to $\mathbf{p}$; in other words, $\mathcal{D}_{\mathbf{p}} = \{\mathcal{I} \in \mathcal{D} : \mathcal{I}[j], \mathcal{I}[n+j] \in \mathbf{p}[j], \forall j\}$, where $\mathbf{p}[j]$ denotes the partition of attribute $j$ in $\mathbf{p}$. Similarly, $\mathcal{D}_{k,\mathbf{p}}$ is the projection of $\mathcal{D}_{\mathbf{p}}$ on the join attributes in relation $R_k$.

For each partition $\mathbf{p} \in \mathcal{P}$, we construct random variables $X_{\mathbf{p}}$ that estimate $Q_{\texttt{COUNT}}$ on the domain space $\mathcal{D}_{\mathbf{p}}$, in a manner similar to the atomic sketch $X$ in Section 3. Thus, for each partition $\mathbf{p}$ and join attribute pair $j$, $n+j$, we have an independent family of random variables $\{\xi_{j,l,\mathbf{p}} : l \in \mathbf{p}[j]\}$, and for each (relation, partition) pair $(R_k, \mathbf{p})$, we define a random variable $X_{k,\mathbf{p}} = \sum_{\mathcal{I} \in \mathcal{D}_{k,\mathbf{p}}} (f_k(\mathcal{I}) \prod_{j \in S_k} \xi_{j,\mathcal{I}[j],\mathbf{p}})$. Variable $X_{\mathbf{p}}$ is then obtained as the product of $X_{k,\mathbf{p}}$'s over all relations, i.e., $X_{\mathbf{p}} = \prod_{k=1}^{r} X_{k,\mathbf{p}}$. It is easy to verify that $E[X_{\mathbf{p}}]$ is equal to the number of tuples in the join result for partition $\mathbf{p}$ and thus, by linearity of expectation, $E[\sum_{\mathbf{p}} X_{\mathbf{p}}] = \sum_{\mathbf{p}} E[X_{\mathbf{p}}] = Q_{\texttt{COUNT}}$.

By independence across partitions, we have $\mathrm{Var}(\sum_{\mathbf{p}} X_{\mathbf{p}}) = \sum_{\mathbf{p}} \mathrm{Var}(X_{\mathbf{p}})$. As in Section 3, to reduce the variance of our partitioned estimator, we construct iid instances of each $X_{\mathbf{p}}$. However, since $\mathrm{Var}(X_{\mathbf{p}})$ may differ widely across the partitions, we can obtain larger reductions in the overall variance by maintaining a larger number of copies for partitions with a higher variance. Let $s_{\mathbf{p}}$ denote the number of iid copies of the sketch $X_{\mathbf{p}}$ maintained for partition $\mathbf{p}$ and let $Y_{i,\mathbf{p}}$ be the average of these $s_{\mathbf{p}}$ copies. Then, we compute $Y_i$ as $\sum_{\mathbf{p}} Y_{i,\mathbf{p}}$ (averaging over iid copies does not alter the expectation, so that $E[Y_i] = Q_{\texttt{COUNT}}$.

The success of our sketch-partitioning approach clearly hinges on being able to efficiently compute the $s_{\mathbf{p}}$ iid instances of $X_{k,\mathbf{p}}$ for each (relation, partition) pair as data tuples are streaming in. For each partition $\mathbf{p}$, we maintain $s_{\mathbf{p}}$ independent families $\xi_{j,\mathbf{p}}$ of variables for each attribute pair $j, n+j$, where each family is generated using an independent random seed. Further, for every tuple $t \in R_k$ in the stream and for every partition $\mathbf{p}$ such that $t$ lies in $\mathbf{p}$ (that is, $t \in \mathcal{D}_{k,\mathbf{p}}$), we add to $X_{k,\mathbf{p}}$ the quantity $\prod_{j \in S_k} \xi_{j,t[j],\mathbf{p}}$. (Note that a tuple $t$ in $R_k$ typically carries only a subset of the join attributes, so it can belong to multiple partitions $\mathbf{p}$.) Our sketch-partitioning techniques make the process of identifying the relevant partitions for a tuple very efficient by using the (approximate) stream statistics to group contiguous regions of values in the domain of each attribute $A_j$ into a small number of coarse *buckets* (e.g., histogram statistics trivially give such a bucketization). Then, each of the $m_j$ partitions for attribute $A_j$ comprises a subset of such buckets and each bucket stores an identifier for its corresponding partition. Since the number of such buckets is typically small, given an incoming tuple $t$, the bucket containing $t[j]$ (and, therefore, the relevant partition along $A_j$) can be determined very quickly (e.g., using binary or linear search). This allows us to very efficiently determine the relevant partitions $\mathbf{p}$ for streaming data tuples.

The total storage required for the atomic sketches over all the partitions is $O(\sum_{\mathbf{p}} s_{\mathbf{p}} \sum_{j=1}^{n} \log |\texttt{dom}(A_j)|)$ to compute each $Y_i$. For the sake of simplicity, we approximate the storage overhead for each $\xi_{j,\mathbf{p}}$ family for partition $\mathbf{p}$ by the constant $O(\sum_{j=1}^{n} \log |\texttt{dom}(A_j)|)$ instead of the more precise (and less pessimistic) $O(\sum_{j=1}^{n} \log |\mathbf{p}[j]|)$. Our sketch-partitioning approach still needs to address two very important issues: (1) Selecting a good set of partitions $\mathcal{P}$; and (2) Determining the number of iid copies $s_{\mathbf{p}}$ of $X_{\mathbf{p}}$ to be constructed for each partition $\mathbf{p}$. Clearly, effectively addressing these issues is crucial to our final goal of minimizing the overall space allocated to the sketch while guaranteeing a a certain degree of accuracy $\epsilon$ for each $Y_i$. Specifically, we aim to compute a partitioning $\mathcal{P}$ and allocating space $s_{\mathbf{p}}$ to each partition $\mathbf{p}$ such that $\mathrm{Var}(Y_i) \leq \frac{\epsilon^2 L^2}{8}$ and $\sum_{\mathbf{p} \in \mathcal{P}} s_{\mathbf{p}}$ is minimized.

Note that, by independence across partitions and the iid characteristics of individual atomic sketches, we have $\mathrm{Var}(Y_i) = \sum_{\mathbf{p}} \frac{\mathrm{Var}(X_{\mathbf{p}})}{s_{\mathbf{p}}}$. Given a attribute-space partitioning $\mathcal{P}$, the problem of choosing the optimal allocation of $s_{\mathbf{p}}$'s that minimizes the overall sketch space while guaranteeing an upper bound on $\mathrm{Var}(Y_i)$ can be formulated as a concrete optimization problem. The following theorem describes how to compute such an optimal allocation.

THEOREM 4.1. *Consider a partitioning $\mathcal{P}$ of the join-attribute domains. Then, allocating space $s_{\mathbf{p}} = \frac{8\sqrt{Var(X_{\mathbf{p}})}\sum_{\mathbf{p}}\sqrt{Var(X_{\mathbf{p}})}}{\epsilon^2 L^2}$ to each $\mathbf{p} \in \mathcal{P}$ ensures that $Var(Y_i) \leq \frac{\epsilon^2 L^2}{8}$ and $\sum_{\mathbf{p}} s_{\mathbf{p}}$ is minimum.* ∎

From the above theorem, it follows that, given a partitioning $\mathcal{P}$, the *optimal space allocation* for a given level of accuracy requires a total sketch space of: $\sum_{\mathbf{p}} s_{\mathbf{p}} = \frac{8(\sum_{\mathbf{p}}\sqrt{\mathrm{Var}(X_{\mathbf{p}})})^2}{\epsilon^2 L^2}$. Obviously, this means that the *optimal partitioning* $\mathcal{P}$ with respect to minimizing the overall space requirements for our sketches is one that minimizes the sum $\sum_{\mathbf{p}} \sqrt{\mathrm{Var}(X_{\mathbf{p}})}$. Thus, in the remainder of this section, we focus on techniques for computing such an optimal partitioning $\mathcal{P}$; once $\mathcal{P}$ has been found, we use Theorem 4.1

[5]Given $\mathrm{Var}(Y_i) \leq \frac{\epsilon^2 L^2}{8}$, a relative error at most $\epsilon$ with probability at least $1-\delta$ can be guaranteed by selecting the median of $s_2 = \log(1/\delta)$ $Y_i$ instantiations.

to compute the optimal space allocation for each partition. We first consider the simpler case of single-join queries, and address multi-join queries in Section 4.3.

## 4.2 Sketch-Partitioning for Single-Join Queries

We describe our techniques for computing an effective partitioning $\mathcal{P}$ of the attribute space for the estimation of COUNT queries over single joins of the form $R_1 \bowtie_{A_1=A_2} R_2$. Since we only consider a single join-attribute pair (and, of course $\mathtt{dom}(A_1) = \mathtt{dom}(A_2)$), for notational simplicity, we ignore the additional subscript for join attributes wherever possible. Our partitioning algorithms rely on knowledge of approximate frequency statistics for attributes $A_1$ and $A_2$. Typically, such approximate statistics are available in the form of per-attribute *histograms* that split the underlying data domain $\mathtt{dom}(A_j)$ into a sequence of contiguous regions of values (termed *buckets*) and store some coarse aggregate statistics (e.g., number of tuples and number of distinct values) within each bucket.

### 4.2.1 Binary Sketch Partitioning

Consider the simple case of a *binary* partitioning $\mathcal{P}$ of $\mathtt{dom}(A_1)$ into two subsets $P_1$ and $P_2$; that is, $\mathcal{P} = \{P_1, P_2\}$. Let $f_k(i)$ denote the frequency of value $i \in \mathtt{dom}(A_1)$ in relation $R_k$. For each relation $R_k$, we associate with the (relation, partition) pair $(R_k, P_l)$ a random variable $X_{k,P_l} = \sum_{i \in P_l} f_k(i)\xi_{i,P_l}$, where $l, k \in \{1, 2\}$. We can now define $X_{P_l} = X_{1,P_l} X_{2,P_l}$ for $l \in \{1, 2\}$. It is obvious that $E[X_{P_l}] = |R_1 \bowtie_{A_1=A_2 \wedge A_1 \in P_l} R_2|$ (i.e., the partial COUNT over $P_l$), and it is easy to check that the variance $\mathrm{Var}(X_{P_l})$ is as follows [2]:

$$\mathrm{Var}(X_{P_l}) = \sum_{i\in P_l} f_1(i)^2 \sum_{i \in P_l} f_2(i)^2 \quad + \quad \left(\sum_{i\in P_l} f_1(i) f_2(i)\right)^2 \\ - \quad 2 \sum_{i \in P_l} f_1(i)^2 f_2(i)^2 \quad (1)$$

Theorem 4.1 tells us that the overall storage is proportional to $\sqrt{\mathrm{Var}(X_{P_1})} + \sqrt{\mathrm{Var}(X_{P_2})}$. Thus, to minimize the total sketching space through partitioning, we need to find the partitioning $\mathcal{P} = \{P_1, P_2\}$ that minimizes $\sqrt{\mathrm{Var}(X_{P_1})} + \sqrt{\mathrm{Var}(X_{P_2})}$. Unfortunately, the minimization problem using the exact values for $\mathrm{Var}(X_{P_1})$ and $\mathrm{Var}(X_{P_2})$ as given in Equation (1) is very hard; we conjecture this optimization problem to be $\mathcal{NP}$-hard and leave proof of this statement for future work. Fortunately, however, due to Lemma 3.2, we know that the variance $\mathrm{Var}(X_{P_l})$ lies in between $\sum_{i\in P_l} f_1(i)^2 \sum_{i\in P_l} f_2(i)^2 - \sum_{i\in P_l} f_1(i)^2 f_2(i)^2$ and $2 \cdot (\sum_{i\in P_l} f_1(i)^2 \sum_{i\in P_l} f_2(i)^2 - \sum_{i\in P_l} f_1(i)^2 f_2(i)^2)$. In general, one can expect the first term $\sum_{i\in P_l} f_1(i)^2 \sum_{i\in P_l} f_2(i)^2$ (i.e., the product of the self-join sizes) to dominate the above bounds. We now demonstrate that, under a loose condition on join-attribute distributions, we can find a close to $\sqrt{2}$-approximation to the optimal value for $\sqrt{\mathrm{Var}(X_{P_1})} + \sqrt{\mathrm{Var}(X_{P_2})}$ by simply substituting $\mathrm{Var}(X_{P_l})$ with $\sum_{i\in P_l} f_1(i)^2 \sum_{i\in P_l} f_2(i)^2$, the product of self-join sizes of the two relations.

Specifically, suppose that we define the join of $R_1$ and $R_2$ to be $\gamma$-*spread* if and only if the condition $\sum_{j\neq i} f_1(j) f_2(j) \geq \gamma \cdot f_1(i) f_2(i)$ holds for all $i \in \mathtt{dom}(A_1)$, for some constant $\gamma > 1$. Essentially, the $\gamma$-spread condition states that not too much of the join-frequency "mass" is concentrated at any single point of the join-attribute domain $\mathtt{dom}(A_1)$. We typically expect the $\gamma$-spread condition to be satisfied in most practical scenarios; violating the condition requires not only $f_1(i)$ and $f_2(i)$ to be severely skewed, but also that their skews are aligned so that they result in extreme skew in the resulting join-frequency vector $f_1(i) f_2(i)$. When no such extreme scenarios arise, and for reasonably-sized join attribute domains, we typically expect the $\gamma$ parameter in the $\gamma$-spread definition to be fairly large; for example, when the $f_1(i) f_2(i)$ distribution is approximately uniform, the $\gamma$-spread condition is satisfied with $\gamma = O(|\mathtt{dom}(A_1)|) >> 1$.

THEOREM 4.2. *For a $\gamma$-spread join $R_1 \bowtie R_2$, determining the optimal solution to the binary-partitioning problem using the self-join-size approximation to the variance guarantees a $\sqrt{2}/(1 - \frac{2}{\sqrt{1+\gamma}})$-factor approximation to the optimal binary partitioning (with respect to the summed square roots of partition variances). In general, if $m$ domain partitions are allowed, the optimal self-join-size solution guarantees a $\sqrt{2}/(1 - \frac{m}{\sqrt{1+\gamma}})$-factor approximation.* ∎

Given the approximation guarantees in Theorem 4.2, we consider the simplified partitioning problem that uses the self-join size approximation for the partition variances; that is, we aim to find a partitioning $\mathcal{P}$ that minimizes the function:

$$\mathcal{F}(\mathcal{P}) = \sqrt{\sum_{i\in P_1} f_1(i)^2 \sum_{i\in P_1} f_2(i)^2} + \sqrt{\sum_{i\in P_2} f_1(i)^2 \sum_{i\in P_2} f_2(i)^2}. \quad (2)$$

Clearly, a brute-force solution to this problem is extremely inefficient as it requires $O(2^{\mathtt{dom}(A_1)})$ time (proportional to the number of all possible partitionings of $\mathtt{dom}(A_1)$). Fortunately, we can take advantage of the following classic theorem from the classification-tree literature [5] to design a much more efficient *optimal* algorithm.

THEOREM 4.3 ([5]). *Let $\Phi(x)$ be a concave function of $x$ defined on some compact domain $\hat{\mathcal{D}}$. Let $P = \{1, \ldots, d\}, d >= 2$, and $\forall i \in P$ let $q_i > 0$ and $r_i$ be real numbers with values in $\hat{\mathcal{D}}$ not all equal. Then one of the partitions $\{P_1, P_2\}$ of $P$ that minimizes $\sum_{i\in P_1} q_i \Phi(\frac{\sum_{i\in P_1} q_i r_i}{\sum_{i\in P_1} q_i}) + \sum_{i\in P_2} q_i \Phi(\frac{\sum_{i\in P_2} q_i r_i}{\sum_{i\in P_2} q_i})$ has the property that $\forall i_1 \in P_1, \forall i_2 \in P_2,\ r_{i_1} < r_{i_2}$.* ∎

To see how Theorem 4.3 applies to our partitioning problem, let $i \in \mathtt{dom}(A_1)$, and set $r_i = \frac{f_1(i)^2}{f_2(i)^2}$, $q_i = \frac{f_2(i)^2}{\sum_{j\in \mathtt{dom}(A_1)} f_2(j)^2}$. Substituting in Equation (2), we obtain:

$$\begin{aligned}\mathcal{F}(\mathcal{P}) &= \sqrt{\sum_{i\in P_1} f_2(i)^2 \sum_{i\in P_1} f_2(i)^2 r_i} + \sqrt{\sum_{i\in P_2} f_2(i)^2 \sum_{i\in P_2} f_2(i)^2 r_i} \\ &= \sum_i f_2(i)^2 \left[\sqrt{\sum_{i\in P_1} q_i \sum_{i\in P_1} q_i r_i} + \sqrt{\sum_{i\in P_1} q_i \sum_{i\in P_1} q_i r_i}\right] \\ &= \sum_i f_2(i)^2 \left[\sum_{i\in P_1} q_i \sqrt{\frac{\sum_{i\in P_1} q_i r_i}{\sum_{i\in P_1} q_i}} + \sum_{i\in P_2} q_i \sqrt{\frac{\sum_{i\in P_2} q_i r_i}{\sum_{i\in P_2} q_i}}\right]\end{aligned}$$

Except for the constant factor $\sum_{i\in \mathtt{dom}(A_1)} f_2(i)^2$ (which is always nonzero if $R_2 \neq \phi$), our objective function $\mathcal{F}$ now has exactly the form prescribed in Theorem 4.3 with $\Phi(x) = \sqrt{x}$. Since $f_1(i) \geq 0, f_2(i) \geq 0$ for $i \in \mathtt{dom}(A_1)$, we have $r_i \geq 0, q_i \geq 0$, and $\forall P_l \subseteq \mathtt{dom}(A_1)$, $\frac{\sum_{i\in P_l} q_i}{\sum_{i\in P_l} q_i r_i} \geq 0$. So, all that remains to be shown is that $\sqrt{x}$ is concave on $\mathtt{dom}(A_1)$. Since concaveness is equivalent to negative second derivative and $(\sqrt{x})'' = -1/4x^{-3/2} \leq 0$, Theorem 4.3 applies.

Applying Theorem 4.3 essentially reduces the search space for finding an optimal partitioning of $\mathtt{dom}(A)$ from exponential to linear, since only partitionings in the order of increasing $r_i$'s need to be considered. Thus, our optimal binary-partitioning algorithm for minimizing $\mathcal{F}(\mathcal{P})$ simply orders the domain values in increasing order of frequency ratios $\frac{f_1(i)}{f_2(i)}$, and only considers partition boundaries between two consecutive values in that order; the partitioning with the smallest resulting value for $\mathcal{F}(\mathcal{P})$ gives the optimal solution.

**Example 2:** Consider the join $R_1 \bowtie_{A_1=A_2} R_2$ of two relations $R_1$ and $R_2$ with $\mathtt{dom}(A_1) = \mathtt{dom}(A_2) = \{1,2,3,4\}$. Also, let the frequency $f_k(i)$ of domain values $i$ for relations $R_1$ and $R_2$ be as follows:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $f_1(i)$ | 20 | 5 | 10 | 2 |
| $f_2(i)$ | 2 | 15 | 3 | 10 |

Without partitioning, the number of copies $s_1$ of the atomic-sketch estimator $X$, so that $\mathrm{Var}(Y_i) \leq \frac{\epsilon^2 L^2}{8}$ is given by $s_1 = \frac{8\mathrm{Var}(X)}{\epsilon^2 L^2}$, where $\mathrm{Var}(X) = 529\cdot 338 + 165^2 - 2\cdot 8525 = 188977$ by Equation (1). Now consider the binary partitioning $\mathcal{P}$ of $\mathtt{dom}(A_1)$ into $P_1 = \{1,3\}$ and $P_2 = \{2,4\}$. The total number of copies $\sum_{\mathbf{P}} s_{\mathbf{P}}$ of the sketch estimators $X_{\mathbf{P}}$ for partitions $P_1$ and $P_2$ is $\sum_{\mathbf{P}} s_{\mathbf{P}} = \frac{8(\sqrt{\mathrm{Var}(X_{P_1})}+\sqrt{\mathrm{Var}(X_{P_2})})^2}{\epsilon^2 L^2}$ (by Theorem (4.1)), where $(\sqrt{\mathrm{Var}(X_{P_1})} + \sqrt{\mathrm{Var}(X_{P_2})})^2 = (\sqrt{6400} + \sqrt{6400})^2 = 25600$. Thus, using this binary partitioning, the sketching space requirements are reduced by a factor of $\frac{s_1}{\sum_{\mathbf{P}} s_{\mathbf{P}}} = \frac{188977}{25600} \approx 7.5$.

Note that the partitioning $\mathcal{P}$ with $P_1 = \{1,3\}$ and $P_2 = \{2,4\}$ also minimizes the function $\mathcal{F}(\mathcal{P})$ defined in Equation (2). Thus, our approximation algorithm based on Theorem 4.3 returns the above partitioning $\mathcal{P}$. Essentially, since $r_1 = \frac{20^2}{2^2} = 100$, $r_2 = \frac{5^2}{15^2} = 1/9$, $r_3 = \frac{10^2}{3^2} = 100/9$ and $r_4 = \frac{2^2}{10^2} = 1/25$, only the three split points in the sequence $4,2,3,1$ of domain values arranged in the increasing order of $r_i$ need to be considered. Of the three potential split points, the one between 2 and 3 results in the smallest value (177) for $\mathcal{F}(\mathcal{P})$. ∎

### 4.2.2 K-ary Sketch Partitioning

We now describe how to extend our earlier results to more general partitionings comprising $m \geq 2$ domain partitions. By Theorem 4.1, we aim to find a partitioning $\mathcal{P} = \{P_1, \ldots, P_m\}$ of $\mathtt{dom}(A_1)$ that minimizes $\sqrt{\mathrm{Var}(X_{P_1})} + \ldots + \sqrt{\mathrm{Var}(X_{P_m})}$, where each $\mathrm{Var}(X_{P_l})$ is computed as in Equation (1). Once again, given the approximation guarantees of Theorem 4.2, we substitute the complicated variance formulas with the product of self-join sizes; thus, we seek to find a partitioning $\mathcal{P} = \{P_1, \ldots, P_m\}$ that minimizes the function:

$$\mathcal{F}(\mathcal{P}) = \sqrt{\sum_{i\in P_1} f_1(i)^2 \sum_{i\in P_1} f_2(i)^2} + \ldots + \sqrt{\sum_{i\in P_m} f_1(i)^2 \sum_{i\in P_m} f_2(i)^2} \tag{3}$$

A brute-force solution to minimizing $\mathcal{F}(\mathcal{P})$ requires an impractical $O(m^{\mathtt{dom}(A_1)})$ time. Fortunately, we have shown the following generalization of Theorem 4.3 that allows us to drastically reduce the problem search space and design a much more efficient algorithm.

THEOREM 4.4. *Consider the function $\Psi(P_1, \ldots, P_m) = \sum_{l=1}^{m} \sum_{i\in P_l} q_i \Phi(\frac{\sum_{i\in P_l} q_i r_i}{\sum_{i\in P_l} q_i})$, where $\Phi$, $q_i$ and $r_i$ are defined as in Theorem 4.3 and $\{P_1, \ldots, P_m\}$ is a partitioning of $P = \{1, \ldots, d\}$. Then among the partitionings that minimize $\Psi(P_1, \ldots, P_m)$ there is one partitioning $\{P_1, \ldots, P_m\}$ with the following property $\pi$: $\forall l, l' \in \{1, \ldots, m\} \ : \ l < l' \implies \forall i \in P_l \ \forall i' \in P_{l'} \ \ r_i < r_{i'}$.* ∎

As described in Section *4.2.1*, our objective function $\mathcal{F}(\mathcal{P})$ can be expressed as $\sum_{i\in\mathtt{dom}(A_1)} f_2(i)^2 \Psi(P_1, \ldots, P_m)$, where $\Phi(x) = \sqrt{x}$, $r_i = \frac{f_1(i)^2}{f_2(i)^2}$ and $q_i = \frac{f_2(i)^2}{\sum_{j\in\mathtt{dom}(A_1)} f_2(j)^2}$; thus, minimizing $\mathcal{F}(\{P_1, \ldots, P_m\})$ is equivalent to minimizing $\Psi(P_1, \ldots, P_m)$. By Theorem 4.4, to find the optimal partitioning for $\Psi$, all we have to do is to consider an arrangement of elements $i$ in $P = \{1, \ldots, d\}$ in the order of increasing $r_i$'s, and find $m-1$ split points in this sequence such that $\Psi$ for the resulting $m$ partitions is as small as possible. The optimum $m-1$ split points can be efficiently found using *dynamic programming*, as follows. Without loss of generality, assume that $1, \ldots, d$ is the sequence of elements in $P$ in increasing value of $r_i$. For $1 \leq u \leq d$ and $1 \leq v \leq m$, let $\psi(u,v)$ be the value of $\Psi$ for the optimal partitioning of elements $1, \ldots u$ (in order of increasing $r_i$) in $v$ parts. The equations describing our dynamic-programming algorithm are:

$$\psi(u,1) \;=\; \sum_{i=1}^{u} q_i \Phi(\frac{\sum_{i=1}^{u} q_i r_i}{\sum_{i=1}^{u} q_i})$$

$$\psi(u,v) \;=\; \min_{1\leq j<u} \left\{ \psi(j, v-1) + \sum_{i=j+1}^{u} q_i \Phi(\frac{\sum_{i=j+1}^{u} q_i r_i}{\sum_{i=j+1}^{u} q_i}) \right\}, \ \ v > 1$$

The correctness of our algorithm is based on the linearity of $\Psi$. Also let $p(u,v)$ be the index of the last element in partition $v-1$ of the optimal partitioning of $1, \ldots, u$ in $v$ parts (so that the last partition consists of elements between $p(u,v)+1$ and $u$). Then, $p(u,1) = 0$ and for $v > 1$, $p(u,v) = \arg\min_{1\leq j<u}\{\psi(j,v-1) + \sum_{i=j+1}^{u} q_i \Phi(\frac{\sum_{i=j+1}^{u} q_i r_i}{\sum_{i=j+1}^{u} q_i})\}$. The actual best partitioning can then be reconstructed from the values of $p(u,v)$ in time $O(m)$; essentially, the $(m-1)^{th}$ split point of the optimal partitioning is $p(d,m)$, the split point preceding it is $p(p(d,m), m-1)$, and so on. The space complexity of the algorithm is obviously $O(md)$ and the time complexity is $O(md^2)$, since we need $O(d)$ time to find the index $j$ that achieves the minimum for a fixed $u$ and $v$, and the function $\Phi()$ for sequences of consecutive elements can be computed in time $O(d^2)$.

## 4.3 Sketch-Partitioning for Multi-Join Queries

**Queries Containing 2 Joins.** When queries contain 2 or more joins, unfortunately, the problem of computing an optimal partitioning becomes intractable. Consider the problem of estimating the join-size of the following query over three relations $R_1$ (containing attribute $A_1$), $R_2$ (containing attributes $A_2$ and $A_3$) and $R_3$ (containing attribute $A_4$): $R_1 \bowtie_{A_1=A_3} R_2 \bowtie_{A_2=A_4} R_3$. We are interested in computing a partitioning $\mathcal{P}$ of attribute domains $A_1$ and $A_2$ such that $|\mathcal{P}| \leq K$ and the quantity $\sum_{\mathbf{P}} \sqrt{\mathrm{Var}(X_{\mathbf{P}})}$ is minimized. Let the partitions of $\mathtt{dom}(A_j)$ be $P_{j,1}, \ldots, P_{j,m_j}$. Then the number of partitions in $\mathcal{P}$, $|\mathcal{P}| = m_1 m_2$. Also, for values $i, j$, let $f_1(i)$, $f_2(i,j)$ and $f_3(j)$ be the frequencies of the values in relations $R_1$, $R_2$ and $R_3$, respectively.

Due to Lemma 3.2, for a partition $(P_{1,l_1}, P_{2,l_2}) \in \mathcal{P}$,

$$\mathrm{Var}(X_{(P_{1,l_1},P_{2,l_2})}) \leq$$
$$10 \cdot \left( \sum_{i \in P_{1,l_1}} f_1(i)^2 \sum_{(i,j) \in (P_{1,l_1},P_{2,l_2})} f_2(i,j)^2 \sum_{j \in P_{2,l_2}} f_3(j)^2 - \sum_{(i,j) \in (P_{1,l_1},P_{2,l_2})} f_1(i)^2 f_2(i,j)^2 f_3(j)^2) \right)$$

Since the first term in the above equation for variance is the dominant term, for the sake of simplicity, we focus on computing a partitioning $\mathcal{P}$ that minimizes the following quantity:

$$\sum_{l_1=1}^{m_1} \sum_{l_2=1}^{m_2} \sqrt{\sum_{i \in P_{1,l_1}} f_1(i)^2 \sum_{(i,j) \in (P_{1,l_1},P_{2,l_2})} f_2(i,j)^2 \sum_{j \in P_{2,l_2}} f_3(j)^2} \quad (4)$$

Unfortunately, we have shown that computing such an optimal partitioning is $\mathcal{NP}$-hard based on a reduction from the MINIMUM SUM OF SQUARES problem [9].

THEOREM 4.5. *The problem of computing a partitioning* $P_{j,1}, \ldots, P_{j,m_j}$ *of* $\mathtt{dom}(A_j)$ *for join attribute* $A_j$, $j = 1, 2$ *such that* $|\mathcal{P}| = m_1 m_2 \leq K$ *and the quantity in Equation (4) is minimized is* $\mathcal{NP}$*-hard.* ∎

In the following subsection, we present a simple heuristic for partitioning attribute domains of multi-join queries that is optimal if attribute value distributions within each relation are independent.

**Optimal Partitioning Algorithm for Independent Join Attributes.** For general multi-join queries, the partitioning problem involves computing a partitioning $P_{j,1}, \ldots, P_{j,m_j}$ of each join attribute domain $\mathtt{dom}(A_j)$ such that $|\mathcal{P}| = \prod_{j=1}^{n} m_j \leq K$ and the quantity $\sum_{\mathbf{p}} \sqrt{\mathrm{Var}(X_{\mathbf{p}})}$ is minimized. Ignoring constants and retaining only the dominant self-join term of $\mathrm{Var}(X_{\mathbf{p}})$ for each partition $\mathbf{p}$ (see Lemma 3.2), our problem reduces to computing a partitioning that minimizes the quantity $\sum_{\mathbf{p}} \sqrt{\prod_{k=1}^{r} \sum_{\mathcal{I} \in \mathcal{D}_{k,\mathbf{p}}} f_k(\mathcal{I})^2}$. Since the 2-join case is a special instance of the general multi-join problem, due to Theorem 4.5, our simplified optimization problem is also $\mathcal{NP}$-hard. However, if we assume that the join attributes in each relation are independent, then a polynomial-time dynamic programming solution can be devised for computing the optimal partitioning. We will employ this optimal dynamic programming algorithm for the independent attributes case as a heuristic for splitting attribute domains for multi-join queries even when attributes may not satisfy our independence assumption.

Suppose that for a relation $R_k$, join attribute $j \in S_k$ and value $i \in \mathtt{dom}(A_j)$, $f_{k,j}(i)$ denotes the number of tuples in $R_k$ for whom $A_j$ has value $i$. Then, the attribute value independence assumption implies that for $\mathcal{I} \in \mathcal{D}_k$, $f_k(\mathcal{I}) = |R_k| \prod_{j \in S_k} \frac{f_{k,j}(\mathcal{I}[j])}{|R_k|}$. This is because the independence of attributes implies the fact that the probability of a particular set of values for the join attributes is the product of the probabilities of each of the values in the set. Under this assumption, one can show that the optimization problem for multiple joins can be decomposed to optimizing the product of single joins. Recall that attributes $j$ and $n+j$ form a join pair, and in the following, we will denote by $R(j)$ the relation containing attribute $A_j$.

THEOREM 4.6. *If relations* $R_1, \ldots, R_r$ *satisfy the attribute value independence assumption, then* $\sum_{\mathbf{p}} \sqrt{\prod_{k=1}^{r} \sum_{\mathcal{I} \in \mathcal{D}_{k,\mathbf{p}}} f_k(\mathcal{I})^2}$ *is simply*

$$\frac{\prod_{k=1}^{r} |R_k|}{\prod_{j=1}^{n} |R(j)||R(n+j)|} \prod_{j=1}^{n} \cdot \sum_{l=1}^{m_j} \sqrt{\sum_{i \in P_{j,l}} f_{R(j),j}(i)^2 \sum_{i \in P_{j,l}} f_{R(n+j),j}(i)^2}$$

∎

Thus, due to Theorem 4.6, and since $\frac{\prod_{k=1}^{r} |R_k|}{\prod_{j=1}^{n} |R(j)||R(n+j)|}$ is a constant independent of the partitioning, we simply need to compute $m_j$ partitions for each attribute $A_j$ such that the product of $\sum_{l=1}^{m_j} \sqrt{\sum_{i \in P_{j,l}} f_{R(j),j}(i)^2 \sum_{i \in P_{j,l}} f_{R(n+j),j}(i)^2}$ for $j = 1, \ldots, n$ is minimized and $\prod_{j=1}^{n} m_j \leq K$. Clearly, the dynamic programming algorithm from Section 4.2.2 can be employed to efficiently compute, for a given value of $m_j$, the optimal $m_j$ partitions (denoted by $P_{j,1}^{opt}, \ldots, P_{j,m_j}^{opt}$) for an attribute $j$ that minimize $\sum_{l=1}^{m_j} \sqrt{\sum_{i \in P_{j,l}} f_{R(j),j}(i)^2 \sum_{i \in P_{j,l}} f_{R(n+j),j}(i)^2}$. Let $Q(j, m_j)$ denote this quantity for the optimal partitions; then, our problem is to compute the values $m_1, \ldots, m_n$ for the $n$ attributes such that $\prod_{j=1}^{n} m_j \leq K$ and $\prod_{j=1}^{n} Q(j, m_j)$ is minimum. This can be efficiently computed using dynamic programming as follows. Suppose $M(u, v)$ denotes the minimum value for $\prod_{j=1}^{u} Q(j, m_j)$ such that $m_1, \ldots, m_u$ satisfy the constraint that $\prod_{j=1}^{u} m_j \leq v$, for $1 \leq u \leq n$ and $1 \leq v \leq K$. Then, one can define $M(u, v)$ recursively as follows:

$$M(u,v) = \begin{cases} Q(u,v) & \text{if } u = 1 \\ \min_{1 \leq l \leq v} \{M(u-1,l) \cdot Q(u, \lfloor \frac{v}{l} \rfloor)\} & \text{otherwise} \end{cases}$$

Clearly, $M(n, K)$ can be computed using dynamic programming, and it corresponds to the minimum value of function $\sum_{\mathbf{p}} \sqrt{\prod_{k=1}^{r} \sum_{\mathcal{I} \in \mathcal{D}_{k,\mathbf{p}}} f_k(\mathcal{I})^2}$ for the optimal partitioning when attributes are independent. Furthermore, if $\mathcal{P}(u, v)$ denotes the optimal $v$ partitions of the attribute space over $A_1, \ldots, A_u$, then $\mathcal{P}(u, v) = \{P_{1,1}^{opt}, \ldots, P_{1,v}^{opt}\}$ if $u = 1$. Otherwise, $\mathcal{P}(u, v) = \mathcal{P}(u-1, l_0) \times \{P_{u,1}^{opt}, \ldots, P_{u,\lfloor \frac{v}{l_0} \rfloor}^{opt}\}$, where $l_0 = \arg\min_{1 \leq l \leq v} \{M(u-1,l) \cdot Q(u, \lfloor \frac{v}{l} \rfloor)\}$.

Computing $Q(u, v)$ for $1 \leq u \leq n$ and $1 \leq v \leq K$ using the dynamic programming algorithm from Section 4.2.2 takes $O(\sum_{j=1}^{n} |\mathtt{dom}(A_j)|^2 K)$ time in the worst case. Furthermore, using the computed $Q(u, v)$ values to compute $M(n, K)$ has a worst-case time complexity of $O(nK)$. Thus, overall, the dynamic programming algorithm for computing $M(n, K)$ has a worst-case time complexity of $O((n + \sum_{j=1}^{n} |\mathtt{dom}(A_j)|^2)K)$. The space complexity of the dynamic programming algorithm is $O(\max_j |\mathtt{dom}(A_j)|K)$, since computation of $M$ for a specific value of $u$ requires only $M$ values for $u-1$ and $Q$ values of $u$ to be kept around.

Note that since building good one-dimensional histograms on streams is much easier than building multi-dimensional histograms, in practice, we expect the partitioning of the domain of join attributes to be made based exclusively on such histograms. In this case, the independence assumption will need to be made anyway to approximate the multi-dimensional frequencies, and so the optimum solution can be found using the above proposed method.

## 5. EXPERIMENTAL STUDY

In this section, we present the results of an extensive experimental study of our sketch-based techniques for processing queries in a streaming environment. Our objective was twofold: We wanted to (1) compare our sketch-based method of approximately answering complex queries over data streams with traditional histogram-based methods, and (2) examine the impact of sketch partitioning on the quality of the computed approximations. Our experiments consider a wide range of `COUNT` queries on both synthetic and real-life data sets. The main findings of our study can be summarized as follows.

• **Improved Query Answer Quality.** Our sketch-based algorithms are quite accurate when estimating the results of complex aggregate queries. Even with few kilobytes of memory, the relative error in final answer is frequently less than 10%. Our experiments also show that our sketch-based method gives much more accurate answers than on-line histogram-based methods, the improvement in accuracy ranging from a factor of three to over an order of magnitude.

• **Effectiveness of Sketch Partitioning.** Our study shows that partitioning attribute domains (using our dynamic programming heuristic to compute the partitions) and carefully allocating the available memory to sketches for the partitions can significantly boost the quality of returned estimates.

• **Impact of Approximate Attribute Statistics.** Our experiments show that sketch partitioning is still very effective and robust even if only very rough and approximate attribute statistics for computing partitions are available.

Thus our experimental results validate the thesis of this paper that sketches are a viable, effective tool for answering complex aggregate queries over data streams, and that a careful allocation of available space through sketch partitioning is important in practice. In the next section, we describe our experimental setup and methodology. All experiments in this paper were performed on a Pentium III with 1 GB of main memory, running Redhat Linux 7.2.

## 5.1 Experimental Testbed and Methodology

**Algorithms for Query Answering.** We focused on algorithms that are truly on-line in that they can work exclusively with a limited amount of main memory and a small per-tuple processing overhead. Since histograms are a popular data reduction technique for approximate query answering [20], and a number of algorithms for constructing equi-depth histograms on-line have been proposed recently [21, 16], we consider equi-depth histograms in our study. However, we do not consider random-sample data summaries since these have been shown to perform poorly for queries with one or more joins [1, 6, 2].

• *Equi-Depth Histograms.* We construct one-dimensional equi-depth histograms off-line since space-efficient on-line algorithms for histograms are still being proposed in the literature, and we would like our study to be valid for the best single-pass algorithms of the future. We do not consider multi-dimensional histograms in our experiments since their construction typically involves multiple passes over the data. (The technique of Gibbons et al. [14] for constructing approximate multi-dimensional histograms utilizes a backing sample and thus cannot be used in our setting.) Consequently, we use the attribute value independence assumption to approximate the value distribution for multiple attributes from the individual attribute histograms. Thus, by assuming that values within each bucket are distributed uniformly and attributes are independent, the entire relation can be approximated and we use this approximation to answer queries. Note that a one-dimensional histogram with $b$ buckets requires $2b$ words (4-byte integers) of storage, one word for each bucket boundary and one for each bucket count.

• *Sketches.* We use our sketch-based algorithm from Section 3 for answering queries, and the dynamic programming-based algorithm from Section 4.3 for computing partitions. We employ sophisticated de-randomization techniques to dramatically reduce the overhead for generating the $\xi_j$ families of independent random variables[6]. Thus, when attribute domains are not partitioned, the total storage requirement for a sketch is approximately $s_1 \cdot s_2 \cdot r$ words, which is essentially the overhead of storing $s_1 \cdot s_2$ random variables for the $r$ relations. On the other hand, in case attributes are split, then the space overhead for the sketch is approximately $\sum_{\mathbf{p}} s_{\mathbf{p}} \cdot s_2 \cdot r$ words. In our experiments, we found that smaller values for $s_2$ generally resulted in better accuracy, and so we set $s_2$ to 2 in all our experiments.

In each experiment, we allocate the same amount of memory to histograms and sketches.

**Data Sets.** We used two real-life and several synthetic data sets in our experiments. We used the synthetic data generator employed previously in [24, 6] to generate data sets with very different characteristics for a wide variety of parameter settings.

• *Census data set (www.bls.census.gov).* This data set was taken from the Current Population Survey (CPS) data, which is a monthly survey of about 50,000 households conducted by the Bureau of the Census for the Bureau of Labor Statistics. Each month's data contains around 135,000 tuples with 361 attributes, of which we used five attributes in our study: age, income, education, weekly wage and weekly wage overtime. The income attribute is discretized and has a range of 1:14, and education is a categorical attribute with domain 1:46. The three numeric attributes age, weekly wage and weekly wage overtime have ranges of 1:99, 0:288416 and 0:288416, respectively. Our study use data from two months (August 1999 and August 2001) containing 72100 and 81600 records,[7] respectively, with a total size of 6.51 MB.

• *Synthetic data sets.* We used the synthetic data generator from [24] to generate relations with 1, 2 and 3 attributes. The data generator works by populating uniformly distributed rectangular regions in the multi-dimensional attribute space of each relation. Tuples are distributed across regions and within regions using a Zipfian distribution with values $z_{inter}$ and $z_{intra}$, respectively. We set the parameters of the data generator to the following default values: size of each domain=1024, number of regions=10, volume of each region=1000–2000, skew across regions ($z_{inter}$)=1.0, skew within each region ($z_{intra}$) =0.0–0.5 and number of tuples in each relation = 10,000,000. By clustering tuples within regions, the data generator used in [24] is able to model correlation among attributes within a relation. However, in practice, join attributes belonging to different relations are frequently correlated. In order to capture this attribute dependency across relations, we introduce a new *perturbation* parameter $p$ (with default value 1.0). Essentially, relation $R_2$ is generated from relation $R_1$ by perturbing each region $r$ in $R_1$ using parameter $p$ as follows. Consider the rectangular space around the center of $r$ obtained as a result of shrinking $r$ by a factor $p$ along each dimension. The new center for region $r$ in $R_2$ is selected to be a random point in the shrunk space.

**Queries.** The workload used to evaluate the various approximation techniques consists of three main query types: (1) Chain JOIN-COUNT Queries: We join two or more relations on one or more attributes such that the join graph forms a chain, and we return the number of tuples in the result of the join as output of the query; (2) Star JOIN-COUNT Queries: We join two or more relations on one or more attributes such that the join graph forms a star, and we return the number of tuples in the output of the query; (3) Self-

[6]A detailed discussion of this is outside the scope of this paper.

[7]We excluded records with missing values.

join JOIN-COUNT Queries: We self-join a relation on one or more attributes, and we return the number of tuples in the output of the query. We believe that the above-mentioned query types are fairly representative of typical query workloads over data streams.

**Answer-Quality Metrics.** In our experiments we use the absolute relative error ($\frac{|actual-approx|}{actual}$) in the aggregate value as a measure of the accuracy of the approximate query answer. We repeat each experiment 100 times, and use the average value for the errors across the iterations as the final error in our plots.

## 5.2 Results: Sketches vs. Histograms

**Synthetic Data Sets.** Figure 2 depicts the error due to sketches and histograms for a self-join query as the amount of available memory is increased. It is interesting to observe that the relative error due to sketches is almost an order of magnitude lower than histograms. The self-join query in Figure 2 is on a relation with a single attribute whose domain size is 1024000. Further, the one-dimensional data set contains 10,000 regions with volumes between 1 and 5, and a skew of 0.2 across the relations ($z_{inter}$). Histograms perform very poorly on this data set since a few buckets cannot accurately capture the data distribution of such a large, sparse domain with so many regions.

**Real-life Data Sets.** The experimental results with the Census 1999 and 2001 data sets are depicted in Figures 3–5. Figure 3 is a join of the two relations on the Weekly Wage attribute and Figure 4 involves joining the relations on the Age and Education attributes. Finally, Figure 5 contains the result of a star query involving four copies of the 2001 Census data set, with center of the star joined with the three other copies on attributes Age, Education and Income. Observe that histograms perform worse than sketches for all three query types; their inferior performance for the first join query (see Figure 3) can be attributed to the large domain size of Weekly Wage (0:288416), while their poor accuracies for the second and third join queries (see Figures 4 and 5) are due to the inherent problems of approximating multi-dimensional distributions from one-dimensional statistics. Specifically, the accuracy of the approximate answers due to histograms suffers because the attribute value independence assumption leads to erroneous estimates for the multi-dimensional frequency histograms of each relation. Note that this also causes the error for histogram-based data summaries to improve very little as more memory is made available to the streaming algorithms. On the other hand, the relative error with sketches decreases significantly as the space allocated to sketches is increased – this is only consistent with theory since according to Theorem 3.1, the sketch error is inversely proportional to the square root of sketch storage. It is worth noting that the relative error of the aggregates for sketches is very low; for all three join queries, it is less than 2% with only a few kilobytes of memory.

## 5.3 Results: Sketch Partitioning

In this set of experiments, each sketch is allocated a fixed amount of memory, and the number of partitions is varied. Also, the sketch partitions are computed using approximate statistics from histograms with 25, 50 and 100 buckets (we plot a separate curve for each histogram size value). Intuitively, histograms with fewer buckets occupy less space, but also introduce more error into the frequency statistics for the attributes based on which the partitions are computed. Thus, our objective with this set of experiments is to show that even with approximate statistics from coarse-grained small histograms, it is possible to use our dynamic programming heuristic to compute partitions that boost the accuracy of estimates.

**Synthetic Data Sets.** Figure 6 illustrates the benefits of partitioning attribute domains, on the accuracy of estimates for a chain join query involving three two-dimensional relations, in which the two attributes of a central relation are joined with one attribute belonging to each of the other two relations. The memory allocated to the sketch for the query is 9000 words.

Clearly, the graph points to two definite trends. First, as the number of sketch partitions increases, the error in the computed aggregates becomes smaller. The second interesting trend is that as histograms become more accurate due to an increased number of buckets, the computed sketch partitions are more effective in terms of reducing error. There are also two further observations that are interesting. First, most of the error reduction occurs for the first few partitions and after a certain point, the incremental benefits of further partitioning are minor. For instance, four partitions result in most of the error reduction, and very little gain is obtained beyond four sketch partitions. Second, even with partitions computed using very small histograms and crude attribute statistics, significant reductions in error are realized. For instance, for an attribute domain of size 1024, even with 25 buckets we are able to reduce error by a factor of 2 using sketch partitioning. Also, note that our heuristic based on dynamic programming for splitting multiple join attributes (see Section 4.3) performs quite well in practice and is able to achieve significant error reductions.

**Real-life Data Sets.** Sketch partitioning also improves the accuracy of estimates for the Census 1999 and 2001 real-life data sets, as depicted in Figure 7. As for synthetic data sets, we allocate a fixed amount, 4000 words, of memory to the sketch for the query, and vary the number of partitions. Also, histograms with 25, 50 and 100 buckets are used to compute sketch partitions. Figure 7 is the join of the two relations on attribute Weekly Wage Overtime for Census 1999 and attribute Weekly Wage for Census 2001.

From the figure, we can conclude that the real-life data sets exhibit the same trends that were previously observed for synthetic data sets. The benefits of sketch partitioning in terms of significant reductions in error are similar for both sets of experiments. Note also that histograms with a small number of buckets are effective for partitioning sketches, even though they give a poor estimate of the join-size for the experiment in Figure 3. This suggests that merely guessing the shape of the distributions is sufficient in most practical situations to allow good sketch partitions to be built.

# 6. CONCLUSIONS

In this paper, we considered the problem of approximatively answering general *aggregate* SQL queries over continuous data streams with limited memory. Our approach is based on computing small "sketch" summaries of the streams that are used to provide approximate answers of complex multi-join aggregate queries with provable approximation guarantees. Furthermore, since the degradation of the approximation quality due to the high variance of our randomized sketch synopses may be a concern in practical situations, we developed novel *sketch-partitioning* techniques. Our proposed methods take advantage of existing statistical information on the stream to intelligently partition the domain of the underlying attribute(s) and, thus, decompose the sketching problem in a way that provably tightens the approximation guarantees. Finally, we conducted an extensive experimental study with both synthetic and real-life data sets to determine the effectiveness of our sketch-based techniques and the impact of sketch partitioning on the quality of computed approximations. Our results demonstrate that (a) our sketch-based technique provides approximate answers of better quality than histograms (by factors ranging from three to an order of magnitude), and (b) sketch partitioning, even when based on coarse statistics, is an effective way to boost the accuracy of our

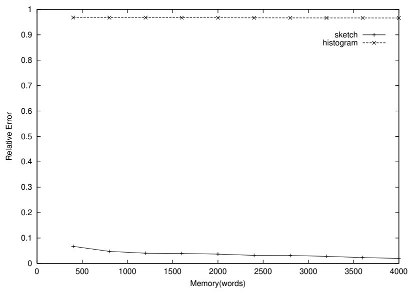


**Figure 2:** **Self-join (1 Attribute)**

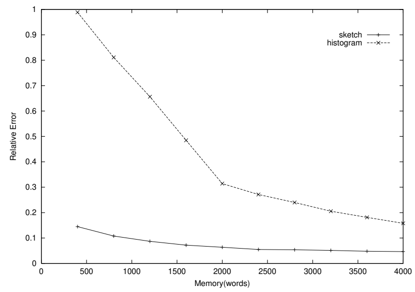


**Figure 3:** **Join (1 Attribute)**

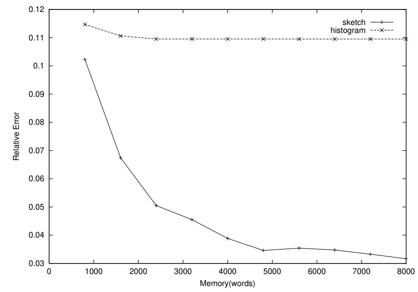


**Figure 4:** **Join (2 Attributes)**

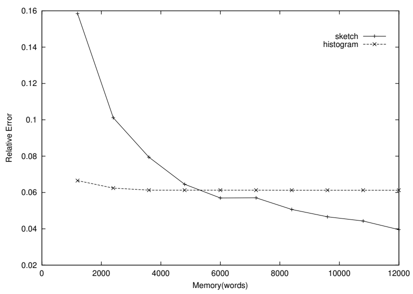


**Figure 5:** **Join (3 Attributes)**

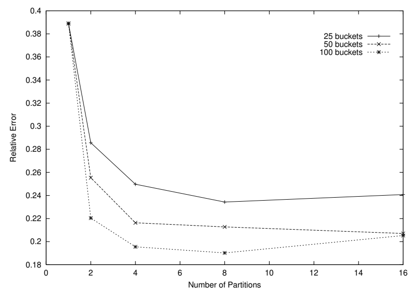


**Figure 6:** **Join (3 Relations)**

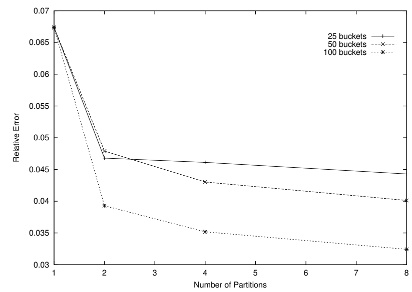


**Figure 7:** **Join (1 Attribute)**

estimates (by a factor of almost two).

# 7. REFERENCES


[1] S. Acharya, P.B. Gibbons, V. Poosala, and S. Ramaswamy. "Join Synopses for Approximate Query Answering". In *Proc. of the 1999 ACM SIGMOD Intl. Conf. on Management of Data*, May 1999.

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

[3] 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.

[4] S. Babu and J. Widom. "Continous Queries over Data Streams". *ACM SIGMOD Record*, 30(3), September 2001.

[5] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *"Classification and Regression Trees"*. Chapman & Hall, 1984.

[6] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. "Approximate Query Processing Using Wavelets". In *Proc. of the 26th Intl. Conf. on Very Large Data Bases*, September 2000.

[7] S. Chaudhuri and U. Dayal. "An Overview of Data Warehousing and OLAP Technology". *ACM SIGMOD Record*, 26(1), March 1997.

[8] M. Datar, A. Gionis, P. Indyk, and R. Motwani. "Maintaining Stream Statistics over Sliding Windows". In *Proc. of the 13th Annual ACM-SIAM Symp. on Discrete Algorithms*, January 2002.

[9] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. "Processing Complex Aggregate Queries over Data Streams". Bell Labs Tech. Memorandum, March 2002.

[10] P. Domingos and G. Hulten. "Mining high-speed data streams". In *Proc. of the Sixth ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, August 2000.

[11] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. "An Approximate $L^1$-Difference Algorithm for Massive Data Streams". In *Proc. of the 40th Annual IEEE Symp. on Foundations of Computer Science*, October 1999.

[12] M. Garofalakis and P.B. Gibbons. "Approximate Query Processing: Taming the Terabytes". Tutorial in *27th Intl. Conf. on Very Large Data Bases*, September 2001.

[13] J. Gehrke, F. Korn, and D. Srivastava. "On Computing Correlated Aggregates over Continual Data Streams". In *Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of Data*, September 2001.

[14] P.B. Gibbons, Y. Matias, and V. Poosala. "Fast Incremental Maintenance of Approximate Histograms". In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, August 1997.

[15] A.C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M.J. Strauss. "Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries". In *Proc. of the 27th Intl. Conf. on Very Large Data Bases*, September 2000.

[16] M. Greenwald and S. Khanna. "Space-efficient online computation of quantile summaries". In *Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of Data*, May 2001.

[17] S. Guha, N. Koudas, and K. Shim. "Data streams and histograms". In *Proc. of the 2001 ACM Symp. on Theory of Computing (STOC)*, July 2001.

[18] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. "Clustering data streams". In *Proc. of the 2000 Annual Symp. on Foundations of Computer Science (FOCS)*, November 2000.

[19] P.J. Haas and J.M. Hellerstein. "Ripple Joins for Online Aggregation". In *Proc. of the 1999 ACM SIGMOD Intl. Conf. on Management of Data*, May 1999.

[20] Y.E. Ioannidis and V. Poosala. "Histogram-Based Approximation of Set-Valued Query Answers". In *Proc. of the 25th Intl. Conf. on Very Large Data Bases*, September 1999.

[21] G. Manku, S. Rajagopalan, and B. Lindsay. "Random sampling techniques for space efficient online computation of order statistics of large datasets". In *Proc. of the 1999 ACM SIGMOD Intl. Conf. on Management of Data*, May 1999.

[22] Y. Matias, J.S. Vitter, and M. Wang. "Dynamic Maintenance of Wavelet-Based Histograms". In *Proc. of the 26th Intl. Conf. on Very Large Data Bases*, September 2000.

[23] J.S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1), 1985.

[24] J.S. Vitter and M. Wang. "Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets". In *Proc. of the 1999 ACM SIGMOD Intl. Conf. on Management of Data*, May 1999.