Sketch-Based Multi-Query Processing over Data Streams

Alin Dobra[1], Minos Garofalakis[2], Johannes Gehrke[3], and Rajeev Rastogi[2]

[1] University of Florida, Gainesville FL, USA adobra@cise.ufl.edu
[2] Bell Laboratories, Lucent Technologies, Murray Hill NJ, USA
{minos,rastogi}@bell-labs.com
[3] Cornell University, Ithaca NY, USA johannes@cs.cornell.edu

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.

Randomized techniques, based on computing small "sketch" synopses for each stream, have recently been shown to be a very effective tool for approximating the result of a single SQL query over streaming data tuples. In this paper, we investigate the problems arising when data-stream sketches are used to process *multiple* such queries concurrently. We demonstrate that, in the presence of multiple query expressions, intelligently *sharing sketches* among concurrent query evaluations can result in substantial improvements in the utilization of the available sketching space and the quality of the resulting approximation error guarantees. We provide necessary and sufficient conditions for multi-query sketch sharing that guarantee the correctness of the result-estimation process. We also prove that optimal sketch sharing typically gives rise to $\mathcal{NP}$-hard questions, and we propose novel heuristic algorithms for finding good sketch-sharing configurations in practice. Results from our experimental study with realistic workloads verify the effectiveness of our approach, clearly demonstrating the benefits of our sketch-sharing methodology.

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 several times throughout their lifetime. For several emerging application domains, however, data arrives and needs to be processed continuously, 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, 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-rate and massive volumes of stream data include retail-chain transaction processing, ATM and credit card operations, financial tickers, Web-server activity logging, and so on. 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, there has been 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).

Given the large diversity of users and/or applications that a generic query-processing environment typically needs to support, it is evident that any realistic stream-query processor must be capable of effectively handling *multiple* standing queries over a collection of input data streams. Given a collection of queries to be processed over incoming streams, two key effectiveness parameters 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. Memory, in particular, constitutes an important constraint on the design of stream processing algorithms since, in a typical streaming environment, only limited memory resources are made available to each of the standing queries.

Prior Work. The recent surge of interest in data-stream computation has led to several (theoretical and practical) studies proposing novel one-pass algorithms with limited memory requirements for different problems; examples include: quantile and order-statistics computation [1]; distinct-element counting [2, 3]; frequent itemset counting [4]; estimating frequency moments, join sizes, and difference norms [5–7]; and computing one- or multi-dimensional histograms or Haar wavelet decompositions [8, 9]. All these papers rely on an approximate query-processing model, typically based on an appropriate underlying synopsis data structure. The synopses of choice for a number of the above-cited papers are based on the key idea of *pseudo-random sketches* which, essentially, can be thought of as simple, randomized linear projections of the underlying data vector(s) [10]. In fact, in our recent work [11], we have demonstrated the utility of sketch synopses in computing provably-accurate approximate answers for a *single* SQL query comprising (possibly) multiple join operators.

None of these earlier research efforts has addressed the more general problem of effectively providing accurate approximate answers to *multiple* SQL queries over a collection of input streams. Of course, the problem of *multi-query optimization* (that is, optimizing multiple queries for concurrent execution in a conventional DBMS) has been around for some time, and several techniques for extending conventional query optimizers to deal with multiple queries have been proposed [12]. The cornerstone of all these techniques is the discovery of common query sub-expressions whose evaluation can be shared among the query-execution plans produced.

Our Contributions. In this paper, we tackle the problem of efficiently processing multiple (possibly, multi-join) concurrent aggregate SQL queries over a collection of input data streams. Similar to earlier work on data streaming [5, 11], our approach is based on computing small, pseudo-random sketch synopses of the data. We demonstrate that, in the presence of multiple query expressions, intelligently *sharing sketches* among concurrent (approximate) query evaluations can result in substantial improvements in the utilization of the available sketching space and the quality of the resulting approximation error guarantees. We provide necessary and sufficient conditions for multi-query sketch sharing that guarantee the correctness of the resulting sketch-based estimators. We also attack the difficult optimization problem of determining sketch-sharing configurations that are optimal (e.g., under a certain error metric for a given amount of space). We prove that optimal sketch sharing typically gives rise to $\mathcal{NP}$-hard questions, and we propose novel heuristic algorithms for finding effective sketch-sharing configurations in practice. More concretely, the key contributions of our work can be summarized as follows.

• **Multi-Query Sketch Sharing: Concepts and Conditions.** We formally introduce the concept of *sketch sharing* for efficient, approximate multi-query stream processing. Briefly, the basic idea is to share sketch computation and sketching space across several queries in the workload that can effectively use the same sketches over (a subset of) their input streams. Of course, since sketches and sketch-based estimators are probabilistic in nature, we also need to ensure that this sharing does not degrade the correctness and accuracy of our estimates by causing desirable estimator properties (e.g., unbiasedness) to be lost. Thus, we present necessary and sufficient conditions (based on the resulting multi-join graph) that fully characterize such “correct” sketch-sharing configurations for a given query workload.

• **Novel Sketch-Sharing Optimization Problems and Algorithms.** Given that multiple correct sketch-sharing configurations can exist for a given stream-query workload, our processor should be able to identify configurations that are optimal or near-optimal; for example, under a certain (aggregate) error metric for the workload and for a given amount of sketching space. We formulate these sketch-sharing optimization problems for different metrics of interest, and propose novel algorithmic solutions for the two key sub-problems involved, namely: (1) *Space Allocation:* Determine the best amount of space to be given to each sketch for a fixed sketch-sharing configuration; and, (2) *Join Coalescing:* Determine an optimal sketch-sharing plan by deciding which joins in the workload will share sketches. We prove that these optimization problems (under different error metrics) are typically $\mathcal{NP}$-hard; thus, we design heuristic approximation algorithms (sometimes with guaranteed bounds on the quality of the approximation) for finding good sketch-sharing configurations in practice.

• **Implementation Results Validating our Sketch-Sharing Techniques.** We present the results from an empirical study of our sketch-sharing schemes with several synthetic data sets and realistic, multi-query workloads. Our results clearly demonstrate the benefits of effective sketch-sharing, showing that very significant improvements in answer quality are possible compared to a naive, no-sharing approach. Specifically, our experiments indicate that sketch sharing can boost accuracy of query answers by factors ranging from 2 to 4 for a wide range of multi-query workloads.

Due to space constraints, the proofs of our analytical results and several details have been omitted; the complete discussion can be found in the full version of this paper [13].

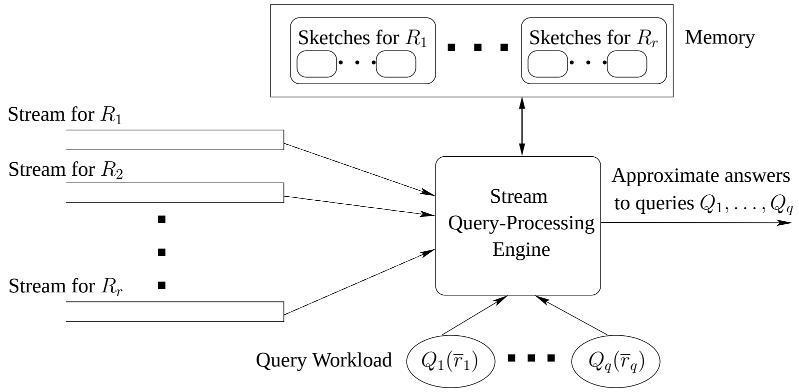


Fig. 1. Stream Multi-Query Processing Architecture.

2 Streams and Random Sketches

2.1 Stream Data-Processing Model

We now briefly describe the key elements of our generic architecture for multi-query processing over continuous data streams (depicted in Fig, 1); similar architectures (for the single-query setting) have been described elsewhere (e.g., [11, 8]). Consider a workload $\mathcal{Q} = \{Q_1, \ldots, Q_q\}$ comprising a collection of (possibly) complex SQL queries $Q_1, \ldots, Q_q$ over a set of relations $R_1, \ldots, R_r$ (of course, each query typically references a subset of the relations/attributes in the input). Also, let $|R_i|$ denote the total number of tuples in R_i. In contrast to conventional DBMS query processors, our stream query-processing engine is allowed to see the data tuples in $R_1, \ldots, R_r$ *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. (Our techniques can also readily handle tuple *deletions* in the streams.)

Our stream query-processing engine is also allowed a certain amount of memory, typically significantly smaller than the total size of the data. This memory is used to maintain a set of concise *synopses* for each data stream R_i. The key constraints imposed on such synopses are that: (1) they are much smaller than the total number of tuples in R_i (e.g., their size is logarithmic or polylogarithmic in $|R_i|$); and, (2) they can be computed quickly, 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 collection of synopses to produce approximate answers to all queries in $\mathcal{Q}$.

2.2 Approximating Single-Query Answers with Pseudo-Random Sketches

The Basic Technique: Binary-Join Size Tracking [5, 6]. Consider a simple stream-processing scenario where the goal is to estimate the size of a binary join of two streams

R_1 and R_2 on attributes $R_1.A_1$ and $R_2.A_2$, respectively. That is, we seek to approximate the result of query $Q = \mathtt{COUNT}(R_1 \bowtie_{R_1.A_1=R_2.A_2} R_2)$ as the tuples of R_1 and R_2 are streaming in. Let $\mathtt{dom}(A)$ denote the domain of an attribute A [4] and $f_R(i)$ be the frequency of attribute value i in $R.A$. (Note that, by the definition of the equi-join operator, the two join attributes have identical value domains, i.e., $\mathtt{dom}(A_1) = \mathtt{dom}(A_2)$.) Thus, we want to produce an estimate for the expression $Q = \sum_{i\in\mathtt{dom}(A_1)} f_{R_1}(i) f_{R_2}(i)$. Clearly, estimating this join size exactly requires at least $\Omega(|\mathtt{dom}(A_1)|)$ space, making an exact solution impractical for a data-stream setting. In their seminal work, Alon et al. [5, 6] propose a randomized technique that can offer strong probabilistic guarantees on the quality of the resulting join-size estimate while using space that can be significantly smaller than $|\mathtt{dom}(A_1)|$.

Briefly, the basic idea of their scheme is to define a random variable X_Q that can be easily computed over the streaming values of $R_1.A_1$ and $R_2.A_2$, such that (1) X_Q is an *unbiased* (i.e., correct on expectation) estimator for the target join size, so that $E[X_Q] = Q$; and, (2) X_Q's variance ($\mathrm{Var}(X_Q)$) can be appropriately upper-bounded to allow for probabilistic guarantees on the quality of the Q estimate. This random variable X_Q is constructed on-line from the two data streams as follows:

- Select a family of *four-wise independent binary random variables* $\{\xi_i : i = 1, \ldots, |\mathtt{dom}(A_1)|\}$, 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 ξ_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 ξ_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 independence, such families can be efficiently constructed on-line using only $O(\log |\mathtt{dom}(A_1)|)$ space [6].
- Define $X_Q = X_1 \cdot X_2$, where $X_k = \sum_{i\in\mathtt{dom}(A_1)} f_{R_k}(i)\xi_i$, for $k = 1, 2$. Quantities X_1 and X_2 are called the *atomic sketches* of relations R_1 and R_2, respectively. Note that each X_k is simply a randomized linear projection (inner product) of the frequency vector of $R_k.A_k$ with the vector of ξ_i's that can be efficiently generated from the streaming values of A_k as follows: Start with $X_k = 0$ and simply add ξ_i to X_k whenever the i^{th} value of A_k is observed in the stream.

The quality of the estimation guarantees can be improved using a standard *boosting technique* that maintains several independent identically-distributed (iid) instantiations of the above process, and uses averaging and median-selection operators over the X_Q estimates to boost accuracy and probabilistic confidence [6]. (Independent instances can be constructed by simply selecting independent random seeds for generating the families of four-wise independent ξ_i's for each instance.) As above, we use the term *atomic sketch* to describe each randomized linear projection computed over a data stream. Letting SJ_k ($k = 1, 2$) denote the self-join size of $R_k.A_k$ (i.e., $\mathrm{SJ}_k = \sum_{i\in\mathtt{dom}(A_k)} f_{R_k}(i)^2$),

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

the following theorem [5] shows how sketching can be applied for estimating binary-join sizes in limited space. (By standard Chernoff bounds [14], using median-selection over $O(\log(1/\delta))$ of the averages computed in Theorem 1 allows the confidence in the estimate to be boosted to $1-\delta$, for any pre-specified $\delta < 1$.)

Theorem 1 ([5]). Let the atomic sketches X_1 and X_2 be as defined above. Then $E[X_Q] = E[X_1X_2] = Q$ and $\text{Var}(X_Q) \leq 2 \cdot \text{SJ}_1 \cdot \text{SJ}_2$. Thus, averaging the X_Q estimates over $O(\frac{\text{SJ}_1\text{SJ}_2}{Q^2\epsilon^2})$ iid instantiations of the basic scheme, guarantees an estimate that lies within a relative error of ϵ from Q with constant probability. □

This theoretical result suggests that the sketching technique is an effective estimation tool only for joins with reasonably high cardinality (with respect to the product of the individual self-join sizes); thus, it may perform poorly for very selective, low-cardinality joins. Note, however, that the strong lower bounds shown by Alon et al. [5] indicate that *any* approximate query processing technique is doomed to perform poorly (i.e., use large amounts of memory) for such low-cardinality joins. Thus, it appears that the only effective way to deal with such very selective joins is through exact computation.

Single Multi-Join Query Answering [11]. In our recent work [11], we have extended sketch-based techniques to approximate the result of a single *multi-join* aggregate SQL query over a collection of streams.[5] More specifically, our work in [11] focuses on approximating a multi-join stream query Q of the form: "`SELECT COUNT FROM` R_1, R_2, ..., R_r `WHERE` $\mathcal{E}$", where $\mathcal{E}$ represents the conjunction of of n equi-join constraints of the form $R_i.A_j = R_k.A_l$ ($R_i.A_j$ denotes the j^{th} attribute of relation R_i). The extension to other aggregate functions, e.g., `SUM`, is fairly straightforward [11]; furthermore, note that dealing with single-relation selections is similarly straightforward (simply filter out the tuples that fail the selection predicate from the relational stream).

Our development in [11] also assumes that each attribute $R_i.A_j$ appears in $\mathcal{E}$ at most once; this requirement can be easily achieved by simply renaming repeating attributes in the query. In what follows, we describe the key ideas and results from [11] based on the join-graph model of the input query Q, since this will allow for a smoother transition to the multi-query case (Section 3).

Given stream query Q, we define the *join graph* of Q (denoted by $\mathcal{J}(Q)$), as follows. There is a distinct vertex v in $\mathcal{J}(Q)$ for each stream R_i referenced in Q (we use $R(v)$ to denote the relation associated with vertex v). For each equality constraint $R_i.A_j = R_k.A_l$ in $\mathcal{E}$, we add a distinct undirected edge $e =< v, w >$ to $\mathcal{J}(Q)$, where $R(v) = R_i$ and $R(w) = R_k$; we also label this edge with the triple $< R_i.A_j, R_k.A_l, Q >$ that specifies the attributes in the corresponding equality constraint and the enclosing query Q (the query label is used in the multi-query setting). Given an edge $e =< v, w >$ with label $< R_i.A_j, R_k.A_l, Q >$, the three components of e's label triple can be obtained as $A_v(e)$, $A_w(e)$ and $Q(e)$. (Clearly, by the definition of equi-joins, $\texttt{dom}(A_v(e)) = \texttt{dom}(A_w(e))$.) Note that there may be multiple edges between a pair of vertices in the

[5] [11] also describes a *sketch-partitioning* technique for improving the quality of basic sketching estimates; this technique is essentially orthogonal to the multi-query problems considered in this paper, so we do not discuss it further.

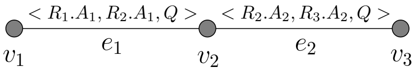


Fig. 2. Example Query Join Graph.

join graph, but each edge has its own distinct label triple. Finally, for a vertex v in $\mathcal{J}(Q)$, we denote the attributes of $R(v)$ that appear in the input query (or, queries) as $\mathcal{A}(v)$; thus, $\mathcal{A}(v) = \{A_v(e)$: edge e is incident on $v\}$.

The result of Q 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. Similar to the basic sketching method [5, 6], our algorithm [11] constructs an unbiased, bounded-variance probabilistic estimate X_Q for Q using atomic sketches built on the vertices of the join graph $\mathcal{J}(Q)$. More specifically, for each edge $e =< v, w >$ in $\mathcal{J}(Q)$, our algorithm defines a family of four-wise independent random variables $\xi^e = \{\xi^e_i : i = 1, \ldots, |\mathtt{dom}(A_v(e))|\}$, where each $\xi^e_i \in \{-1, +1\}$. The key here is that the equi-join attribute pair $A_v(e)$, $A_w(e)$ associated with edge e shares the same ξ family; on the other hand, distinct edges of $\mathcal{J}(Q)$ use *independently-generated* ξ families (using mutually independent random seeds). The atomic sketch X_v for each vertex v in $\mathcal{J}(Q)$ is built as follows. Let $e_1, \ldots, e_k$ be the edges incident on v and, for $i_1 \in \mathtt{dom}(A_v(e_1)), \ldots, i_k \in \mathtt{dom}(A_v(e_k))$, let $f_v(i_1, \ldots, i_k)$ denote the number of tuples in $R(v)$ that match values $i_1, \ldots, i_k$ in their join attributes. More formally, $f_v(i_1, \ldots, i_k)$ is the number of tuples $t \in R(v)$ such that $t[A_v(e_j)] = i_j$, for $1 \leq j \leq k$ ($t[Aj]$ denotes the value of attribute A in tuple t). Then, the atomic sketch at v is $X_v = \sum_{i_1 \in \mathtt{dom}(A_v(e_1))} \cdots \sum_{i_k \in \mathtt{dom}(A_v(e_k))} f_v(i_1, \ldots, i_k) \prod_{j=1}^{k} \xi^{e_j}_{i_j}$. Finally, the estimate for Q is defined as $X_Q = \prod_v X_v$ (that is, the product of the atomic sketches for all vertices in $\mathcal{J}(Q)$). Note that each atomic sketch X_v is again a randomized linear projection that can be efficiently computed as tuples of $R(v)$ are streaming in; more specifically, X_v is initialized to 0 and, for each tuple t in the $R(v)$ stream, the quantity $\prod_{j=1}^{k} \xi^{e_j}_{t[A_v(e_j)]} \in \{-1, +1\}$ is added to X_v.

Example 1. Consider query Q=`SELECT COUNT FROM` R_1, R_2, R_3 `WHERE` $R_1.A_1 = R_2.A_1$ `AND` $R_2.A_2 = R_3.A_2$. The join graph $\mathcal{J}(Q)$ is depicted in Figure 2, with vertices v_1, v_2, and v_3 corresponding to streams R_1, R_2, and R_3, respectively. Similarly, edges e_1 and e_2 correspond to the equi-join constraints $R_1.A_1 = R_2.A_1$ and $R_2.A_2 = R_3.A_2$, respectively. (Just to illustrate our notation, $R(v_1) = R_1$, $A_{v_2}(e_1) = R_2.A_1$ and $\mathcal{A}(v_2) = \{R_2.A_1, R_2.A_2\}$.) The sketch construction defines two families of four-wise independent random families (one for each edge): $\{\xi^{e_1}_i\}$ and $\{\xi^{e_2}_j\}$. The three atomic sketches X_{v_1}, X_{v_2}, and X_{v_3} (one for each vertex) are defined as: $X_{v_1} = \sum_{i \in \mathtt{dom}(R_1.A_1)} f_{v_1}(i)\xi^{e_1}_i$, $X_{v_2} = \sum_{i \in \mathtt{dom}(R_2.A_1)} \sum_{j \in \mathtt{dom}(R_2.A_2)} f_{v_2}(i, j)\xi^{e_1}_i \xi^{e_2}_j$, and $X_{v_3} = \sum_{j \in \mathtt{dom}(R_3.A_2)} f_{v_3}(j)\xi^{e_3}_j$. The value of random variable $X_Q = X_{v_1} X_{v_2} X_{v_3}$ gives the sketching estimate for the result of Q. □

Our analysis in [11] shows that the random variable X_Q constructed above is an unbiased estimator for Q, and demonstrates the following theorem which generalizes the earlier result of Alon et al. [5] to multi-join queries. (SJ$_v$ $= \sum_{i_1 \in \mathtt{dom}(A_v(e_1))} \cdots \sum_{i_k \in \mathtt{dom}(A_v(e_k))} f_v(i_1, \ldots, i_k)^2$ is the self-join size of $R(v)$.)

Theorem 2 ([11]). Let Q be a `COUNT` query with n equi-join predicates such that $\mathcal{J}(\mathcal{Q})$ contains no cycles of length > 2. Then, $E[X_Q] = Q$ and using sketching space of $O(\frac{\text{Var}[X_Q]\cdot\log(1/\delta)}{Q^2\cdot\epsilon^2})$, it is possible to approximate Q to within a relative error of ϵ with probability at least $1-\delta$, where $\text{Var}[X_Q] \le 2^{2n}\prod_v \text{SJ}_v$. □

3 Sketch Sharing: Basic Concepts and Problem Formulation

In this section, we turn our attention to sketch-based processing of *multiple* aggregate SQL queries over streams. We introduce the basic idea of sketch sharing and demonstrate how it can improve the effectiveness of the available sketching space and the quality of the resulting approximate answers. We also characterize the class of correct sketch-sharing configurations and formulate the optimization problem of identifying an effective sketch-sharing plan for a given query workload.

3.1 Sketch Sharing

Consider the problem of using sketch synopses for the effective processing of a query workload $\mathcal{Q} = \{Q_1, \ldots, Q_q\}$ comprising multiple (multi-join) `COUNT` aggregate queries. As in [11], we focus on `COUNT` since the extension to other aggregate functions is relatively straightforward; we also assume an attribute-renaming step that ensures that each stream attribute is referenced only once in each of the Q_i's (of course, the same attribute can be used multiple times across the queries in $\mathcal{Q}$). Finally, as in [11], we do not consider single-relation selections, since they can be trivially incorporated in the model by using the selection predicates to define filters for each stream. The sketching of each relation is performed using only the tuples that pass the filter; this is equivalent to introducing virtual relations/streams that are the result of the filtering process and formulating the queries with respect to these relations. This could potentially increase the number of relations and reduce the number of opportunities to share sketches (as described in this section), but would also create opportunities similar to the ones investigated by traditional MQO (e.g., constructing sketches for common filter sub-expressions). In this paper, we focus on the novel problem of sharing sketches and we do not investigate further how such techniques can be used for the case where selection predicates are allowed. As will become apparent in this section, sketch sharing is very different from common sub-expression sharing; traditional MQO techniques do not apply for this problem.

An obvious solution to our multi-query processing problem is to build disjoint join graphs $\mathcal{J}(Q_i)$ for each query $Q_i \in \mathcal{Q}$, and construct independent atomic sketches for the vertices of each $\mathcal{J}(Q_i)$. The atomic sketches for each vertex of $\mathcal{J}(Q_i)$ can then be combined to compute an approximate answer for Q_i as described in [11] (Section 2.2). A key drawback of such a naive solution is that it ignores the fact that a relation R_i may appear in multiple queries in $\mathcal{Q}$. Thus, it should be possible to reduce the overall space requirements by *sharing* atomic-sketch computations among the vertices for stream R_i in the join graphs for the queries in our workload. We illustrate this in the following example.

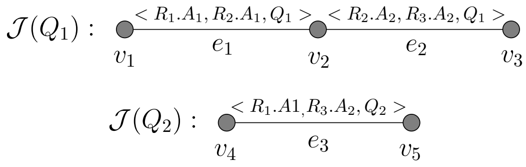

$\mathcal{J}(Q_1)$:
$< R_1.A_1, R_2.A_1, Q_1 >$
$< R_2.A_2, R_3.A_2, Q_1 >$
v_1
e_1
v_2
e_2
v_3
$\mathcal{J}(Q_2)$:
$< R_1.A1, R_3.A_2, Q_2 >$
v_4
e_3
v_5

Fig. 3. Example Workload with Sketch-Sharing Potential.

Example 2. Consider queries Q_1 = SELECT COUNT FROM R_1, R_2, R_3 WHERE $R_1.A_1 = R_2.A_1$ AND $R_2.A_2 = R_3.A_2$ and Q_2 = SELECT COUNT FROM R_1, R_3 WHERE $R_1.A_1 = R_3.A_2$. The naive processing algorithm described above would maintain two disjoint join graphs (Fig. 3) and, to compute a single pair (X_{Q_1}, X_{Q_2}) of sketch-based estimates, it would use three families of random variables (ξ^{e_1}, ξ^{e_2}, and ξ^{e_3}), and a total of five atomic sketches (X_{v_k}, $k = 1, \ldots, 5$).

Instead, suppose that we decide to re-use the atomic sketch X_{v_1} for v_1 also for v_4, both of which essentially correspond to the same attribute of the same stream ($R_1.A_1$). Since for each $i \in \texttt{dom}(R_1.A_1)$, $f_{v_4}(i) = f_{v_1}(i)$, we get $X_{v_4} = X_{v_1} = \sum_{i \in \texttt{dom}(R_1.A_1)} f_{v_4}(i)\xi_i^{e_1}$. Of course, in order to correctly compute a probabilistic estimate of Q_2, we also need to use the same family ξ^{e_1} in the computation of X_{v_5}; that is, $X_{v_5} = \sum_{i \in \texttt{dom}(R_1.A_1)} f_{v_5}(i)\xi_i^{e_1}$. It is easy to see that both final estimates $X_{Q_1} = X_{v_1}X_{v_2}X_{v_3}$ and $X_{Q_1} = X_{v_1}X_{v_5}$ satisfy all the premises of the sketch-based estimation results in [11]. Thus, by simply sharing the atomic sketches for v_1 and v_4, we have reduced the total number of random families used in our multi-query processing algorithm to two (ξ^{e_1} and ξ^{e_2}) and the total number of atomic sketches maintained to four. □

Let $\mathcal{J}(\mathcal{Q})$ denote the collection of all join graphs in workload $\mathcal{Q}$, i.e., all $\mathcal{J}(Q_i)$ for $Q_i \in \mathcal{Q}$. Sharing sketches between the vertices of $\mathcal{J}(\mathcal{Q})$ can be seen as a transformation of $\mathcal{J}(\mathcal{Q})$ that essentially *coalesces* vertices belonging to different join graphs in $\mathcal{J}(\mathcal{Q})$. (We also use $\mathcal{J}(\mathcal{Q})$ to denote the transformed multi-query join graph.) Of course, as shown in Example 2, vertices $v \in \mathcal{J}(Q_i)$ and $w \in \mathcal{J}(Q_j)$ can be coalesced in this manner *only if* $R(v) = R(w)$ (i.e., they correspond to the same data stream) and $\mathcal{A}(v) = \mathcal{A}(w)$ (i.e., both Q_i and Q_j use exactly the same attributes of that stream). Such vertex coalescing implies that a vertex v in $\mathcal{J}(\mathcal{Q})$ can have edges from multiple different queries incident on it; we denote the set of all these queries as $Q(v)$, i.e., $Q(v) = \{Q(e) : \text{edge } e \text{ is incident on } v\}$. Figure 4(a) pictorially depicts the coalescing of vertices v_1 and v_4 as discussed in Example 2. Note that, by our coalescing rule, for each vertex v, all queries in $Q(v)$ are guaranteed to use exactly the same set of attributes of $R(v)$, namely $\mathcal{A}(v)$; furthermore, by our attribute-renaming step, each query in $Q(v)$ uses each attribute in $\mathcal{A}(v)$ exactly once. This makes it possible to share an atomic sketch built for the coalesced vertices v across all queries in $Q(v)$ but, as we will see shortly, cannot guarantee the correctness of the resulting sketch-based estimates.

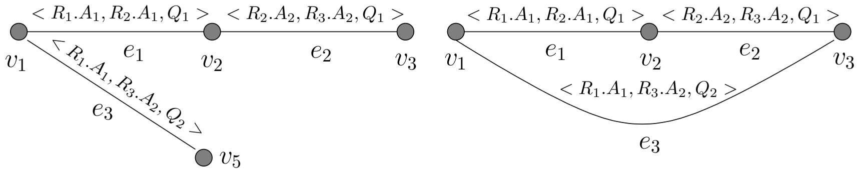
Estimation with Sketch Sharing. Consider a multi-query join graph $\mathcal{J}(\mathcal{Q})$, possibly containing coalesced vertices (as described above). Our goal here is to build *atomic sketches corresponding to individual vertices* of $\mathcal{J}(\mathcal{Q})$ that can then be used for obtaining sketch-based estimates for *all* the queries in our workload $\mathcal{Q}$. Specifically, consider a query $Q \in \mathcal{Q}$, and let $V(Q)$ denote the (sub)set of vertices in $\mathcal{J}(\mathcal{Q})$ attached to a join-predicate edge corresponding to Q; that is, $V(Q) = \{v :$ edge e is incident on v and $Q(e) = Q\}$. Our goal is to construct an unbiased probabilistic estimate X_Q for Q using the atomic sketches built for vertices in $V(Q)$.

The atomic sketch for a vertex v of $\mathcal{J}(\mathcal{Q})$ is constructed as follows. As before, each edge $e \in \mathcal{J}(\mathcal{Q})$ is associated with a family ξ^e of four-wise independent $\{-1, +1\}$ random variables. The difference here, however, is that edges attached to node v for the *same attribute* of $R(v)$ share the *same* ξ family since the *same* sketch of $R(v)$ corresponding to vertex v is used to estimate *all* queries in $Q(v)$; this, of course, implies that the number of *distinct* ξ families for all edges incident on v is exactly $|\mathcal{A}(v)|$ (each family corresponding to a distinct attribute of $R(v)$). Furthermore, all distinct ξ families in $\mathcal{J}(\mathcal{Q})$ are generated independently (using mutually independent seeds). For example, in Figure 4(a), since $A_{v_1}(e_1) = A_{v_1}(e_3) = R_1.A_1$, edges e_1 and e_3 share the same ξ family (i.e., $\xi^{e_3} = \xi^{e_1}$); on the other hand, ξ^{e_1} and ξ^{e_2} are distinct and independent. Assuming $\mathcal{A} = \{A_1, \ldots, A_k\}$ and letting $\xi^1, \ldots, \xi^k$ denote the k corresponding distinct ξ families attached to v, the atomic sketch X_v for node v is simply defined as $X_v = \sum_{(i_1,\ldots,i_k) \in A_1 \times \cdots \times A_k} f_v(i_1, \ldots, i_k) \prod_{j=1}^{k} \xi^j_{i_j}$ (again, a randomized linear projection). The final sketch-based estimate for query Q is the product of the atomic sketches over all vertices in $V(Q)$, i.e., $X_Q = \prod_{v \in V(Q)} X_v$.

Correctness of Sketch-Sharing Configurations. The X_Q estimate construction described above can be viewed as simply "extracting" the join (sub)graph $\mathcal{J}(Q)$ for query Q from the multi-query graph $\mathcal{J}(\mathcal{Q})$, and constructing a sketch-based estimate for Q as described in Section 2.2. This is because, if we were to only retain in $\mathcal{J}(\mathcal{Q})$ vertices and edges associated with Q, then the resulting subgraph is identical to $\mathcal{J}(Q)$. Furthermore, our vertex coalescing (which completely determines the sketches to be shared) guarantees that Q references exactly the attributes $\mathcal{A}(v)$ of $R(v)$ for each $v \in V(Q)$, so the atomic sketch X_v can be utilized.

There is, however, an important complication that our vertex-coalescing rule still needs to address, to ensure that the atomic sketches for vertices of $\mathcal{J}(\mathcal{Q})$ provide unbiased query estimates with variance bounded as described in Theorem 2. Given an estimate X_Q for query Q (constructed as above), unbiasedness and the bounds on Var$[X_Q]$ given in Theorem 2 depend crucially on the assumption that the ξ families used for the edges in $\mathcal{J}(Q)$ are distinct and independent. This means that simply coalescing vertices in $\mathcal{J}(\mathcal{Q})$ that use the same set of stream attributes is insufficient. The problem here is that the constraint that all edges for the same attribute incident on a vertex v share the same ξ family may (by transitivity) force edges for the same query Q to share identical ξ families. The following example illustrates this situation.

Example 3. Consider the multi-query join graph $\mathcal{J}(\mathcal{Q})$ in Figure 4(b) for queries Q_1 and Q_2 in Example 3. ($\mathcal{J}(\mathcal{Q})$ is obtained as a result of coalescing vertex pairs v_1, v_4 and v_3, v_5 in Fig, 3.) Since $A_{v_1}(e_1) = A_{v_1}(e_3) = R_1.A_1$ and $A_{v_3}(e_2) = A_{v_3}(e_3) = R_3.A_2$, we get the constraints $\xi^{e_3} = \xi^{e_1}$ and $\xi^{e_3} = \xi^{e_2}$. By transitivity, we have $\xi^{e_1} = \xi^{e_2} = \xi^{e_3}$, i.e., all three edges of the multi-query graph share the same ξ family. This, in turn, implies that the same ξ family is used on both edges of query Q_1; that is, instead of being independent, the pseudo-random families used on the two edges of Q_1 are perfectly correlated! It is not hard to see that, in this situation, the expectation and


< R_1.A_1, R_2.A_1, Q_1 >
< R_2.A_2, R_3.A_2, Q_1 >
v_1
e_1
v_2
e_2
v_3
< R_1.A_1, R_3.A_2, Q_2 >
e_3
v_5
< R_1.A_1, R_2.A_1, Q_1 >
< R_2.A_2, R_3.A_2, Q_1 >
v_1
e_1
v_2
e_2
v_3
< R_1.A_1, R_3.A_2, Q_2 >
e_3

(a) Well-formed join graph

(b) Join graph that is not well-formed

Fig. 4. Multi-Query Join Graphs $\mathcal{J}(\mathcal{Q})$ for Example 2.

variance derivations for X_{Q_1} will fail to produce the results of Theorem 2, since many of the zero cross-product terms in the analysis of [5, 11] will fail to vanish. □

As is clear from the above example, the key problem is that constraints requiring ξ families for certain edges incident on each vertex of $\mathcal{J}(\mathcal{Q})$ to be identical, can transitively ripple through the graph, forcing much larger sets of edges to share the same ξ family. We formalize this fact using the following notion of (transitive) *ξ-equivalence* among edges of a multi-query graph $\mathcal{J}(\mathcal{Q})$.

Definition 1. *Two edges e_1 and e_2 in $\mathcal{J}(\mathcal{Q})$ are said to be ξ-equivalent if either (1) e_1 and e_2 are incident on a common vertex v, and $A_v(e_1) = A_v(e_2)$; or (2) there exists an edge e_3 such that e_1 and e_3 are ξ-equivalent, and e_2 and e_3 are ξ-equivalent.*

Intuitively, the classes of the ξ-equivalence relation represent exactly the sets of edges in the multi-query join graph $\mathcal{J}(\mathcal{Q})$ that need to share the same ξ family; that is, for any pair of ξ-equivalent edges e_1 and e_2, it is the case that $\xi^{e_1} = \xi^{e_2}$. Since, for estimate correctness, we require that all the edges associated with a query have distinct and independent ξ families, our sketch-sharing algorithms only consider multi-query join graphs that are *well-formed*, as defined below.

Definition 2. *A multi-query join graph $\mathcal{J}(\mathcal{Q})$ is well-formed iff, for every pair of ξ-equivalent edges e_1 and e_2 in $\mathcal{J}(\mathcal{Q})$, the queries containing e_1 and e_2 are distinct, i.e., $Q(e_1) \neq Q(e_2)$.*

It is not hard to prove that the well-formedness condition described above is actually necessary and sufficient for individual sketch-based query estimates that are unbiased and obey the variance bounds of Theorem 2. Thus, our shared-sketch estimation process over well-formed multi-query graphs can readily apply the single-query results of [5, 11] for each individual query in our workload.

3.2 Problem Formulation

Given a large workload $\mathcal{Q}$ of complex queries, there can obviously be a large number of well-formed join graphs for $\mathcal{Q}$, and all of them can potentially be used to provide approximate sketch-based answers to queries in $\mathcal{Q}$. At the same time, since the key resource constraint in a data-streaming environment is imposed by the amount of memory available to the query processor, our objective is to compute approximate answers to queries in $\mathcal{Q}$ that are as accurate as possible given a fixed amount of memory M for the sketch synopses. Thus, in the remainder of this paper, we focus on the problem of computing (1) a well-formed join graph $\mathcal{J}(\mathcal{Q})$ for $\mathcal{Q}$, and (2) an allotment of the M units of space to the vertices of $\mathcal{J}(\mathcal{Q})$ (for maintaining iid copies of atomic sketches), such that an appropriate aggregate error metric (e.g., average or maximum error) for all queries in $\mathcal{Q}$ is minimized.

More formally, let m_v denote the sketching space allocated to vertex v (i.e., number of iid copies of X_v). Also, let M_Q denote the number of iid copies built for the query estimate X_Q. Since $X_Q = \prod_{v \in V(Q)} X_v$, it is easy to see that M_Q is actually constrained by the *minimum* number of iid atomic sketches constructed for each of the nodes in $V(Q)$; that is, $M_Q = \min_{v \in V(Q)}\{m_v\}$. By Theorem 2, this implies that the (square) error for query Q is equal to W_Q/M_Q, where $W_Q = \frac{8\text{Var}[X_Q]}{E[X_Q]^2}$ is a constant for each query Q (assuming a fixed confidence parameter δ). Our sketch-sharing optimization problem can then be formally stated as follows.

Problem Statement. Given a query workload $\mathcal{Q} = \{Q_1, \ldots, Q_q\}$ and an amount of sketching memory M, compute a multi-query graph $\mathcal{J}(\mathcal{Q})$ and a space allotment $\{m_v : \text{ for each node } v \text{ in } \mathcal{J}(\mathcal{Q})\}$ such that one of the following two error metrics is minimized:

- Average query error in $\mathcal{Q}$ = $\sum_{Q \in \mathcal{Q}} \frac{W_Q}{M_Q}$.
- Maximum query error in $\mathcal{Q}$ = $\max_{Q \in \mathcal{Q}}\{\frac{W_Q}{M_Q}\}$.

subject to the constraints: (1) $\mathcal{J}(\mathcal{Q})$ is well-formed; (2) $\sum_v m_v \leq M$ (i.e., the space constraint is satisfied); and, (3) For all vertices v in $\mathcal{J}(\mathcal{Q})$, for all queries $Q \in Q(v)$, $M_Q \leq m_v$. □

The above problem statement assumes that the "weight" W_Q for each query $Q \in \mathcal{Q}$ is known. Clearly, if coarse statistics in the form of histograms for the stream relations are available (e.g., based on historical information or coarse a-priori knowledge of data distributions), then estimates for $E[X_Q]$ and Var$[X_Q]$ (and, consequently, W_Q) can be obtained by estimating join and self-join sizes using these histograms [11]. In the event that no prior information is available, we can simply set each $W_Q = 1$; unfortunately, even for this simple case, our optimization problem is intractable (see Section 4).

In the following section, we first consider the sub-problem of optimally allocating sketching space (such that query errors are minimized) to the vertices of a *given*, well-formed join graph $\mathcal{J}(\mathcal{Q})$. Subsequently, in Section 5, we consider the general optimization problem where we also seek to determine the best well-formed multi-query graph for the given workload $\mathcal{Q}$. Since most of these questions turn out to be $\mathcal{NP}$-hard, we propose novel heuristic algorithms for determining good solutions in practice. Our algorithm for the overall problem (Section 5) is actually an iterative procedure that uses the space-allocation algorithms of Section 4 as subroutines in the search for a good sketch-sharing plan.

4 Space Allocation Problem

In this section, we consider the problem of allocating space optimally given a well-formed join graph $J = \mathcal{J}(\mathcal{Q})$ such that the average or maximum error is minimized.

4.1 Minimizing the Average Error

The problem of allocating space to sketches in a way that minimizes the average error turns out to be $\mathcal{NP}$-hard even when $W_Q = 1$. Given the intractability of the problem, we look for an approximation based on its continuous relaxation, i.e., we allow the M_Q's and m_v's to be continuous. The continuous version of the problem is a convex optimization problem, which can be solved exactly in polynomial time using, for example, interior point methods [15]. We can then show that a near-optimal integer solution is obtained by rounding down (to integers) the optimal continuous values of the M_Q's and m_v's.

Since standard methods for solving convex optimization problems tend to be slow in practice, we developed a novel specialized solution for the problem at hand. Our solution, which we believe has applications to a much wider class of problems than the optimal space allocation problem outlined in this paper, is based on a novel usage of the Kuhn-Tucker optimality conditions (KT-conditions). We rewrite the problem using the KT conditions, and then we solve the problem through repeated application of a specific Max-Flow formulation of the constraints. Due to space limitations, we omit a detailed description of the algorithm and the analysis of its correctness; details can be found in the full version of this paper [13]. Our results are summarized in the following theorem:

Theorem 3. There is an algorithm that computes the optimal solution to the average-error continuous convex optimization problem in at most $O(\min\{|\mathcal{Q}|, |J|\} \cdot (|\mathcal{Q}| + |J|)^3)$ steps. Furthermore, rounding this optimal continuous solution results in an integer solution that is guaranteed to be within a factor of $(1 + \frac{2|J|}{M})$ of the optimal integer solution. $\square$

4.2 Minimizing the Maximum Error

It can be easily shown (see [13] for details) that the problem of minimizing the maximum error can be solved in time $O(|J| \log |J|)$ by the following greedy algorithm: (1) take each m_v proportional to $\max_{Q \in Q(v)} W_Q$), (2) round down each m_v component to the nearest integer, and (3) take the remaining space $s \leq |J|$ and allocate one extra unit of space to each of the nodes with the s smallest values for quantity $m_v / \max_{Q \in Q(v)} W_Q$.

5 Computing a Well-formed Join Graph

The optimization problem we are trying to solve is: find a well-formed graph $\mathcal{J}(\mathcal{Q})$ and the optimal space allocation to the vertices of $\mathcal{J}(\mathcal{Q})$ such that the average or maximum error is minimized. If we take $W_Q = 1$ for all queries and minimize the maximum error, this optimization problem reduces to the problem of finding a well-formed join graph $\mathcal{J}(\mathcal{Q})$ with the minimum number of vertices; this problem is $\mathcal{NP}$-hard (see [13] for the proof) which makes the initial optimization problem $\mathcal{NP}$-hard as well.

In order to provide an acceptable solution in practice we designed a greedy heuristic, that we call `CoalesceJoinGraphs`, for computing a well-formed join graph with small error. The Algorithm `CoalesceJoinGraphs` iteratively merges pair of vertices in J that causes the largest decrease in error, until the error cannot be reduced any further by coalescing vertices. It uses the algorithm to compute the average (Section 4.1) or maximum error (Section 4.2) for a join graph as a subroutine, which we denote by `ComputeSpace`, at every step. Also, in order to ensure that graph J always stays well-formed, J is initially set to be equal to the set of all the individual join graphs for queries in $\mathcal{Q}$. In each subsequent iteration, only vertices for identical relations that have the same attribute sets and preserve the well-formedness of J are coalesced. Well-formedness testing essentially involves partitioning the edges of J' into equivalence classes, each class consisting of ξ-equivalent edges, and then verifying that no equivalence class contains multiple edges from the same join query; it can be carried out very efficiently, in time proportional to the number of edges in J'. The Algorithm `CoalesceJoinGraphs` needs to make at most $O(N^3)$ calls to `ComputeSpace`, where N is the total number of vertices in all the join graphs $\mathcal{J}(Q)$ for the queries, and this determines its time complexity.

6 Experimental Study

In this section, we present the results of an experimental study of our sketch-sharing algorithms for processing multiple `COUNT` queries in a streaming environment. Our experiments consider a wide range of `COUNT` queries on a common schema and set of equi-join constraints and with synthetically generated data sets. The reason we use synthetic data sets is that these enable us to measure the effectiveness of our sketch sharing techniques for a variety of different data distributions and parameter settings. The main findings of our study can be summarized as follows.

• **Effectiveness of Sketch Sharing.** Our experiments with various realistic workloads indicate that, in practice, sharing sketches among queries can significantly reduce the number of sketches needed to compute estimates. This, in turn, results in better utilization of the available memory, and much higher accuracy for returned query answers. For instance, for our first query set (its description is provided latter in this section), the number of vertices in the final coalesced join graph returned by our sketch-sharing algorithms decreases from 34 (with no sharing) to 16. Further, even with $W_Q = 1$ (for all queries Q), compared to naive solutions which involve no sketch sharing, our sketch-sharing solutions deliver improvements in accuracy ranging from a factor of 2 to 4 for a wide range of multi-query workloads.

• **Benefits of Intelligent Space Allocation.** The errors in the approximate query answers computed by our sketch-sharing algorithms are smaller if approximate weight information $W_Q = \frac{8\text{Var}[X]}{E[X]^2}$ for queries is available. Even with weight estimates based on coarse statistics on the underlying data distribution (e.g., histograms), accuracy improvements of up to a factor of 2 can be obtained compared with using uniform weights for all queries.

Thus, our experimental results validate the thesis of this paper that sketch sharing can significantly improve the accuracy of aggregate queries over data streams, and that a careful allocation of available space to sketches is important in practice.

6.1 Experimental Testbed and Methodology

Algorithms for Answering Multiple Aggregate Queries. We compare the error performance of the following two sketching methods for evaluating query answers.

• *No sketch sharing.* This is the naive sketching technique from Section 2.2 in which we maintain separate sketches for each individual query join graph $\mathcal{J}(Q)$. Thus, there is no sharing of sketching space between the queries in the workload, and independent atomic sketches are constructed for each relation, query pair such that the relation appears in the query.

• *Sketch sharing.* In this case, atomic sketches for relations are reused as much as possible across queries in the workload for the purpose of computing approximate answers. Algorithms described in Sections 4 and 5 are used to compute the well-formed join graph for the query set and sketching space allocation to vertices of the join graph (and queries) such that either the average-error or maximum-error metric is optimized. There are two solutions that we explore in our study, based on whether prior (approximate) information on join and self-join sizes is available to our algorithms to make more informed decisions on memory allocation for sketches.

- No prior information. The weights for all join queries in the workload are set to 1, and this is the input to our sketch-sharing algorithms.
- Prior information is available. The ratio $\frac{8\text{Var}(X)}{E[X]^2)}$ is estimated for each workload query, and is used as the query weight when determining the memory to be allocated to each query. We use coarse one-dimensional histograms for each relational attribute to estimate join and self-join sizes required for weight computation. Each histogram is given 200 buckets, and the frequency distribution for multi-attribute relations is approximated from the individual attribute histograms by applying the attribute value independence assumption.

Query Workload. The query workloads used to evaluate the effectiveness of sketch sharing consist of collections of JOIN-COUNT queries roughly based on the schema and queries in TPC-H benchmark; this allows us to simulate a somewhat realistic scenario in which sketches can be shared. The schema consists of six relations with one to three join attributes. Three workloads have been defined on this schema. Workload 1, inspired by the TPC-H workload, consists of twelve queries. Workload 2 extends the first workload with seventeen random queries. Workload 3 contains seven queries from the workload 1 that contain one or two join constraints together with a query from workload 3 that contains three join constraints. The full details, which we omit due to lack of space, can be found in [13].

Data Set. We used the synthetic data generator from [16] to generate all the relations in our experiments. 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

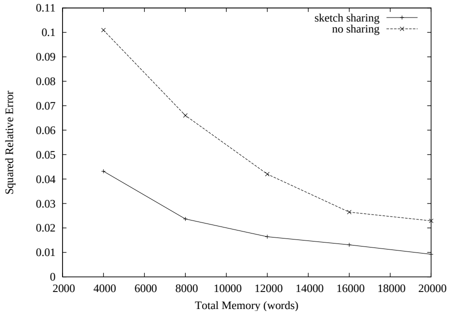


Fig. 5. Average error (workload 1)

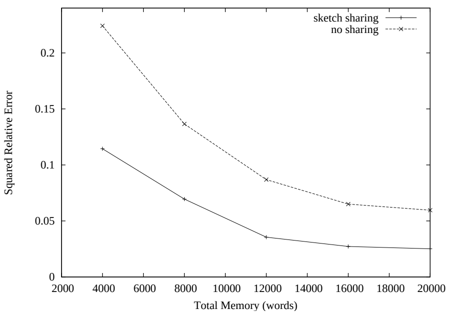


Fig. 6. Maximum error (workload 1)

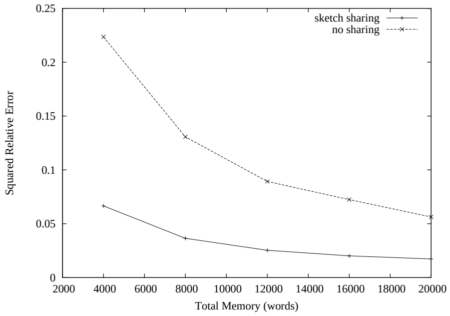


Fig. 7. Average error (workload 2)

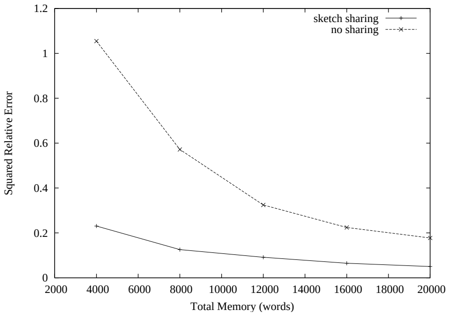


Fig. 8. Maximum error (workload 2)

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.

Answer-Quality Metrics. In our experiments we use the square of the absolute relative error ($\frac{(actual-approx)^2}{actual^2}$) in the aggregate value as a measure of the accuracy of the approximate answer for a single query. For a given query workload, we consider both the average-error and maximum-error metrics, which correspond to averaging over all the query errors and taking the maximum from among the query errors, respectively. We repeat each experiment 100 times, and use the average value for the errors across the iterations as the final error in our plots.

6.2 Experimental Results

Results: Sketch Sharing. Figures 5 through 8 depict the average and maximum errors for query workloads 1 and 2 as the sketching space is increased from 2K to 20K words. From the graphs, it is clear that with sketch sharing, the accuracy of query estimates improves. For instance, with workload 1, errors are generally a factor of two smaller with sketch sharing. The improvements due to sketch sharing are even greater for workload 2

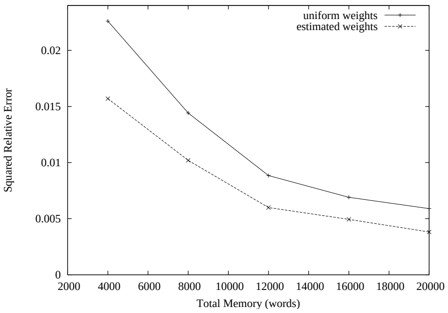


Fig. 9. Average error (workload 3)

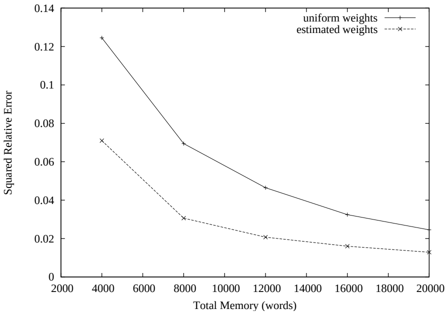


Fig. 10. Maximum error (workload 3)

where due to the larger number of queries, the degree of sharing is higher. The improvements can be attributed to our sketch-sharing algorithms which drive down the number of join graph vertices from 34 (with no sharing) to 16 for workload 1, and from 82 to 25 for workload 2. Consequently, more sketching space can be allocated to each vertex, and hence the accuracy is better with sketch sharing compared to no sharing. Further, observe that in most cases, errors are less than 10% for sketch sharing, and as would be expected, the accuracy of estimates gets better as more space is made available to store sketches.

Results: Intelligent Space Allocation. We plot in Figures 9 and 10, the average and maximum error graphs for two versions of our sketch-sharing algorithms, one that is supplied uniform query weights, and another with estimated weights computed using coarse histogram statistics. We considered query workload 3 for this experiment since workloads 2 and 3 have queries with large weights that access all the underlying relations. These queries tend to dominate in the space allocation procedures, causing the final result to be very similar to the uniform query weights case, which is not happening for query workload 3. Thus, with intelligent space allocation, even with coarse statistics on the data distribution, we are able to get accuracy improvements of up to a factor of 2 by using query weight information.

7 Concluding Remarks

In this paper, we investigated the problem of processing *multiple* aggregate SQL queries over data streams concurrently. We proved correctness conditions for multi-query sketch sharing, and we developed solutions to the optimization problem of determining sketch-sharing configurations that are optimal under average and maximum error metrics for a given amount of space. We proved that the problem of optimally allocating space to sketches such that query estimation errors are minimized is $\mathcal{NP}$-hard. As a result, for a given multi-query workload, we developed a mix of near-optimal solutions (for space allocation) and heuristics to compute the final set of sketches that result in small errors. We conducted an experimental study with realistic query workloads; our findings indicate that (1) Compared to a naive solution that does not share sketches among queries, our sketch-sharing solutions deliver improvements in accuracy ranging from a factor of 2 to 4, and (2) The use of prior information about queries (e.g., obtained from coarse histograms), increases the effectiveness of our memory allocation algorithms, and can cause errors to decrease by factors of up to 2.

Acknowledgements. This work was partially funded by NSF CAREER Award 0133481, NSF ITR Grant 0205452, by NSF Grant IIS-0084762, and by the KD-D Initiative. Any opinions, findings, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the sponsors.

References

1. Gilbert, A.C., Kotidis, Y., Muthukrishnan, S., Strauss, M.J.: “How to Summarize the Universe: Dynamic Maintenance of Quantiles”. In: VLDB 2002, Hong Kong, China (2002)
2. Bar-Yossef, Z., Jayram, T., Kumar, R., Sivakumar, D., Trevisan, L.: “Counting distinct elements in a data stream”. In: RANDOM’02, Cambridge, Massachusetts (2002)
3. Gibbons, P.B., Tirthapura, S.: “Estimating Simple Functions on the Union of Data Streams”. In: SPAA 2001, Crete Island, Greece (2001)
4. Manku, G.S., Motwani, R.: “Approximate Frequency Counts over Data Streams”. In: VLDB 2002, Hong Kong, China (2002)
5. Alon, N., Gibbons, P.B., Matias, Y., Szegedy, M.: “Tracking Join and Self-Join Sizes in Limited Storage”. In: PODS 2001, Philadeplphia, Pennsylvania (1999)
6. Alon, N., Matias, Y., Szegedy, M.: “The Space Complexity of Approximating the Frequency Moments”. In: STOC 1996, Philadelphia, Pennsylvania (1996) 20–29
7. Indyk, P.: “Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computation”. In: FOCS 2000, Redondo Beach, California (2000) 189–197
8. Gilbert, A.C., Kotidis, Y., Muthukrishnan, S., Strauss, M.J.: “Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries”. In: VLDB 2000, Roma, Italy (2000)
9. Thaper, N., Guha, S., Indyk, P., Koudas, N.: “Dynamic Multidimensional Histograms”. In: SIGMOD 2002, Madison, Wisconsin (2002)
10. Garofalakis, M., Gehrke, J., Rastogi, R.: “Querying and Mining Data Streams: You Only Get One Look”. Tutorial at VLDB 2002, Hong Kong, China (2002)
11. Dobra, A., Garofalakis, M., Gehrke, J., Rastogi, R.: “Processing Complex Aggregate Queries over Data Streams”. In: SIGMOD 2002, Madison, Wisconsin (2002) 61–72
12. Sellis, T.K.: “Multiple-Query Optimization”. ACM Transactions on Database Systems **13** (1988) 23–52
13. Dobra, A., Garofalakis, M., Gehrke, J., Rastogi, R.: Sketch-based multi-query processing over data streams. (Manuscript available at: `www.cise.ufl.edu/˜adobra/papers/sketch-mqo.pdf`)
14. Motwani, R., Raghavan, P.: “Randomized Algorithms”. Cambridge University Press (1995)
15. Stefanov, S.M.: Separable Programming. Volume 53 of Applied Optimization. Kluwer Academic Publishers (2001)
16. Vitter, J.S., Wang, M.: “Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets”. In: SIGMOD 1999, Philadelphia, Pennsylvania (1999)