Join Size Estimation Over Data Streams Using Cosine Series

Zhewei Jiang[1], Cheng Luo[1], Wen-Chi Hou[1], Feng Yan[2], Qiang Zhu[3], Chih-Fang Wang[1]

[1]Department of Computer Science, Southern Illinois University, Carbondale IL 62901
{zjiang, cluo, hou, cfw}@cs.siu.edu

[2]Department of Mathematics, Southern Illinois University, Carbondale IL 62901
fyan1@netscape.com

[3]Department of Computer and Information Science, University of Michigan,
Dearborn, MI 48128, USA
qzhu@umich.edu

Abstract

In many applications, data takes the form of a continuous stream rather than a persistent data set. Data stream processing is generally an on-line, one-pass process and is required to be time and space efficient too. In this paper, we develop a framework for estimating join size over the data streams based on the discrete cosine transform (DCT). The DCT generally can provide concise and accurate approximations to data distributions and its coefficients can be updated easily in the presence of insertions and deletions. These features make the DCT suitable for dynamic data stream environments. We have performed analyses and conducted experiments to investigate the applicability of the cosine transform to data streams. The experimental results show that given the same amount of storage space, our method yields more accurate estimates most of the time than the sketch-based methods, which have become the main methods for approximate query processing over data streams. The experimental results have also confirmed that the cosine series can be updated quickly to cope with the rapid flow of data streams.

Keyword: Data Stream, Cosine Series, Query Estimation

1. Introduction

Many applications, such as telephone fraud detection, financial tickers, network monitoring, tele-communications data management, etc., generate data in the form of a continuous stream rather than a persistent data set. Generally, elements of data streams arrive continuously and there is no control over the order in which they arrive. Moreover, a data stream is usually unbounded and there is only one chance to look at it as the data pass by.

To observe, monitor, or summarize the continuous flow of data, queries are posed periodically. These queries are typically referred to as continuous queries because they are issued once and then run continuously [5], unlike the traditional one-time queries that are executed only once. Examples of continuous queries over data streams include a web-based financial search engine that evaluates queries over real-time streaming financial data, an integrated security platform that performs complex stream processing, such as the URL-filtering and join queries over multiple network traffic flows [4]. Continuous query processing generally requires queries to be executed in real-time with limited storage, and thus it must be an on-line, one-pass, and time and space efficient process.

The join operator is probably the most important operator in query processing as it can relate information from different sources (e.g., tables). Some examples of join operations on data streams are finding similar news items from different media sources and finding correlation between phone calls and stock trading [35].

Selectivity estimation or query size estimation plays an important role in query optimization, OLAP, decision support, etc. It has also found its place in data stream processing, such as in the trend analysis, fraud detection, quality and performance monitoring, etc. In this paper, we discuss continuous selectivity estimation over data streams for queries with equi-join operations.

Approximate aggregation query processing has been an important research topic in traditional databases for more than a decade. Various techniques, such as sampling [1, 11, 28], histogram [13, 17, 18, 20], wavelet [6, 7, 27], sketch [2, 3, 32], and discrete cosine transform [21] etc., have been proposed and some of them have been implemented in commercial database systems. In a continuous query processing environment, approximate query results are reported continuously; it poses stern challenges to conventional methods because all these performance measures, such as accuracy, speed, space, and updatability, now become equally important to the methods.

Sampling [14, 15, 22] is a simple and dynamic approach. It can be easily adapted to the continuous data stream environment, but its accuracy for join queries becomes an issue of concern [1, 15] unless the sample size is very large. Histogram provides a concise and efficient way to represent distributions of low-dimensional data. However, as the number of dimensions increases, the space required increases dramatically. The situation is exacerbated by the potentially large domains of attributes in data stream applications. Partition of buckets in the presence of updates can also be very time consuming [19]. The wavelet transform is able to compress a histogram into a small number of coefficients [23] and offers a space efficient (compressed) representation of the data distributions. However, in a dynamic streaming environment, it could require space as large as the size of the data stream itself to calculate the wavelet coefficients [12] and thus not directly applicable to data stream processing. In addition, the wavelet also has a complicated update scheme [24]. Sketch [2, 3, 32] has been proposed for aggregate query estimation recently. Its interesting randomizing algorithm and updatability make it attractive to data stream processing. Sketch has become the basis of many recent works in approximate aggregation query processing over data streams [3, 9, 13, 32]. Although it performs better than the random sampling [3] and histogram [9], large estimation errors are still reported in [9, 13].

In this research, we concentrate on join query size estimation over data streams. While there has been some research on query size estimation over data streams, most of it concentrates on point and range query estimation [6, 11, 13, 25, 27, 28]. Alon et al. [2, 3] uses sketches to estimate the sizes of single equi-join queries, including self-join queries. To the best of our knowledge, only Dobra et al. [9] addresses the same type of query as ours – multi-equi-join query over continuous data stream. Their research is also based on the sketch approach [3].

Our approach is based on the discrete cosine transform. The discrete cosine transform has been used to approximate signals and images of various forms successfully. However, it has not received its due attentions in the field of estimation in databases. In this paper, we develop a framework for join query size estimation using the cosine series. The cosine series also has a straightforward update scheme in the presence of insertions and deletions. We will compare the space requirement, estimation speed, accuracy, and updatability with the sketch approaches. Our experimental results show that the cosine transform yields better estimates, from several times to hundreds of times better, than the sketches most of the time using the same amount of storage space.

The rest of the paper is organized as follows. Section 2 is a brief survey of techniques used in approximation query processing. Section 3 introduces the background of the discrete cosine transform for estimation of aggregation queries. Section 4 details our method on estimating aggregation query with equi-join operators. In section 5, we compare estimation accuracy, speed, space, and updatability of our approach with the sketch methods for single and multiple join queries. Section 6 concludes.

2. Related Work

In this section, we briefly review techniques used in approximate aggregation query processing and discuss their potentials in join size estimation over data streams. Here, a join mainly refers to an equi-join.

There is a long history of using sampling in selectivity estimation on traditional data [14, 15, 22, 28]. In general, with only small samples, accurate estimates can be obtained for point and range queries. Recently, sampling has been used to create synopses for join queries (with foreign key constraints) over data streams [1]. While sampling may adapt itself well to the data streams, the estimation accuracy for join queries is far from satisfactory unless the sample size is very large [1, 15].

Histograms provide a simple way to represent the data distributions and they have had much success in low dimensional selection queries [20, 21, 24, 25]. There have also been some efforts in estimating join sizes using histograms [17, 18]. However, only special types of join predicates, such as k-clique queries [17] and chain/tree queries [18], are considered. It is known that the storage space of histograms can increase dramatically when the number of attributes involved (or the dimensions) increases [4]. This situation is further exacerbated by the usually large domains of attribute values in data stream applications. Dynamic updates of the bucket boundaries can also pose a problem for some types of histograms when used with data streams.

Wavelet Transform has been used to compress histograms into small numbers of coefficients [6, 7, 24, 27]. It has been used for range, point, and range-sum queries [6, 27]. Unfortunately, as the number of dimensions increases, the accuracy degrades drastically unless the number of coefficients used increases significantly. The update of the wavelet coefficients also faces a severe challenge as new data keep flowing in in a data stream environment. Martias et al. [24] has addressed the issue, but their method is still very complicated and inefficient. Recently, Gilbert et al. [12] has shown that wavelet may not be directly applicable to data streams because it could require a space as large as the size of the data stream itself to generate the highest coefficients.

While the above approaches have all shown some deficiencies for data stream processing, Alon et al.'s randomizing technique [2], called sketch, shows some promise. They use a set of independent randomized linear-projection variables, called atomic sketches, to estimate self-join sizes. To construct an atomic sketch, join attribute values are first mapped to {-1, 1} with equal probabilities. Then, an atomic sketch X is derived as the inner product of the frequency vector of the join attribute values with the random vector of {-1, 1} for the join attribute values, that is, $X = \sum_{i \in Dom(A)} f_A(i)\xi_i$, where Dom(A) is the domain of the join attribute A, $f_A(i)$ is the number of the value i of attribute A appearing in the relation, and ξ_i ($\in$ {-1, 1}) is the mapped value of i. It has been shown that the expected value of X^2 is the size of the self-join. To improve the accuracy, Alon et al. [2] uses groups of such independent atomic sketches to estimate the join size. By averaging and selecting the group median, the final estimation is generated. Here, we shall call Alon et al.'s sketch method [2, 3] the basic sketch as it has become the basis of several other methods.

Gilbert et al. [12] has used the sketch method to estimate point and range query size, while others [3, 9, 32] estimate join size over data streams. Alon et al. [3] first uses sketch to estimate the size of a single equi-join query. To improve the accuracy, Dobra et al. [9] first partitions the underlying join attribute domains and then estimates the join size of each individual sub-domain using the sketch [9]. This approach however requires a priori knowledge of the data distributions (to find a good partition) and relies on the independence assumption of join attributes, which may not be feasible for data stream environment. Ganguly's skimmed sketch [32] skims (extracts) the dense frequencies that are greater than a certain threshold into another distribution. The join size is estimated as the sum of the sub-join sizes of these sub-distributions using the sketch. Better estimation results than the basic sketch [3] are reported. However, extra space, in the order of the attribute domain size, is needed to store the dense frequencies.

Discrete cosine transform (DCT) provides an elegant way to approximate data distributions [21, 29,]. Similar to the wavelet transform, DCT requires only a small amount of space to store the (approximated) data distributions. Another advantage of the method is that its coefficients can be updated easily and dynamically. While (discrete) cosine transform has been successfully applied to range queries [21, 2], to the best of our knowledge, it has not been used for join size estimation.

3. Cosine Series Approximation

In this section, we discuss some properties of discrete cosine transform to lay down the groundwork for estimation of aggregation join queries.

3.1 Attribute Values and Normalization

In general, an attribute can be either numerical or categorical. By mapping each categorical value to a distinct number, we can assume hereafter all attributes are numerical. Let X be the attribute of concern. To apply the discrete cosine transform, a normalization of attribute values is needed first. The attribute values are normalized to a predetermined domain [0, 1] as follows. Let maxX and minX be the maximal and minimal values of attribute X of the data stream, respectively. Then, each value *x* of X is normalized as follows:

$$x^z = \frac{x - \min X}{\max X - \min X} \qquad (3.1)$$

where x^z denotes the normalized value of *x*. For example, an attribute domain of {0, 1, 2, 3, 4} is normalized to {0, 1/4, 2/4, 3/4, 1}. The minimal and maximal values of an attribute can usually be determined based on knowledge of the data. For example, the minimal and maximal values of the attribute “Age” can be reasonably assumed to be 0 and 150, respectively. From now on, we shall assume all attributes are so normalized or they all have domains [0, 1], unless otherwise stated.

3.2 Discrete Cosine Transform

To illustrate the use of the discrete cosine transform, let us first consider a one-dimensional case. Let *N* be the total number of tuples seen so far in the data stream and Dom(X) the domain of attribute X (i.e., [0,1]). The frequency function of the X values is defined as

$$f(x) = \frac{count_x}{N}, \qquad x \in Dom(X)$$

where $count_x$ is the number of elements seen so far in the data stream with the value *x*. The frequency function satisfies the relations: $\sum_{x \in Dom(X)} f(x) = 1$.

Let *n* = |Dom(x)|. By the theory of discrete cosine transform, *f(x)* can be represented as

$$f(x) = \sum_{k=0}^{n-1} \alpha_k \phi_k(x)$$

where $\phi_k(x) = 1$ when $k = 0$; otherwise, $\phi_k(x) = \sqrt{2}\cos k\pi x$. $\alpha_k, k > 0$, are computed by the following formula,

$$\alpha_k = \frac{1}{N}\sum_{i=1}^{N} \phi_k(t_i) = \frac{1}{N}\sum_{j=1}^{n} count_{x_j} \cdot \phi_k(x_j) \qquad (3.2)$$

where $t_i, 1 \le i \le N$, is the X value of the i^{th} element in the data stream, and x_j is the j^{th} X value in Dom(X).

The DCT is known to have an excellent engergy compaction property, where most of the signal information, here the frequency function, tends to be concentrated in a few low-frequency components of the transform [34]. Therefore, the frequency function, in common practice, is approximated by the first *m* coefficient terms, where *m* is a number that is much smaller than the domain size *n*. That is,

$$f(x) \approx \sum_{k=0}^{m-1} \alpha_k \phi_k(x)$$

Example. Consider a one-attribute data stream with 6 tuples {0.33, 0.32, 0.12, 0.66, 0.90, 0.80}. The cosine transformation of this distribution is derived as follows.

Given the cosine series $\phi_k(x), k \ge 0$: $\{1, \sqrt{2}\cos\pi x, \sqrt{2}\cos 2\pi x, ..., \sqrt{2}\cos k\pi x, ...\}$, we derive the respective coefficients as : $a_0 = 1,\ \ a_1 = \frac{1}{6}\sum_{j=1}^{6}\phi_1(t_j) = -0.063,\ \ a_2 = \frac{1}{6}\sum_{j=1}^{6}\phi_2(t_j) = 0.0951,\ \cdots$, where $\phi_k(t_j)$ is $\sqrt{2}\cos k\pi t_j,\ k \ge 1$, and t_j, $1 \le j \le 6$, is the j^th element in the stream. **End of example #**

To apply the transform to the d-dimensional case, the distribution is approximated by its m^d coefficients $a_{k_1,...,k_d}$, $0 \le k_1,...,k_d \le m-1$, as :

$$a_{k_1,..., k_d} = \frac{1}{N}\sum_{i=1}^{N} [\prod_{j=1}^{d} \phi_{k_j}(t_{ij})], \qquad (3.3)$$

where t_{ij} is the j^th attribute of the i^th tuple t_i, $1 \le i \le N$.

As observed from Eq. (3.3), each coefficient $a_{k_1,...,k_d}$ of the transform is just the average of the sum of the products of the basis functions (i.e., $\phi_k(x)$) on the tuples. Therefore, for insertion or deletion of a tuple, we just compute the "contribution" of that tuple to the transform separately and then combine it with the old coefficients. That is, for the arrival of a new tuple $x = (x_1, x_2, ..., x_d)$ to the data stream, which has already had N tuples, $a_{k_1,...,k_d}$ is updated as

$$a_{k_1,...k_d} = \frac{N}{N+1} a_{k_1,...k_{d1}} + \frac{1}{N+1} \prod_{j=1}^{d} \phi_{k_j}(x_j) \tag{3.4}$$

Similarly, to delete a tuple $x = (x_1, x_2, ..., x_d)$ from the data stream, the coefficient is updated as

$$a_{k_1,...k_d} = \frac{N}{N-1} a_{k_{11},...k_{d1}} - \frac{1}{N-1} \prod_{j=1}^{d} \phi_{k_j}(x_j) \tag{3.5}$$

Coefficients can be updated easily and dynamically. Note that the set of coefficients derived by the above incremental update scheme (using Eq. (3.4)) is exactly the same as if we had derived in batch fashion using the Eq. (3.3).

The updates of the coefficients can also be performed in a batch fashion. That is, one can store the frequencies of the newly arrived attribute values in a buffer and then update the coefficients all at once. Note that the time taken to update the coefficients for a batch of newly arrived elements is same as that for each individual tuple. This batch update method can significantly reduce the overheads for updates.

A technique, called the triangular sampling [21], can be used to filter out high frequencies from the m^d coefficients without much information loss. It retains only those coefficients whose indexes satisfy the condition $k_1 + ... + k_d \le m - 1$. The number of coefficients finally stored is $\binom{m+d-1}{d} \le m^d$. Note that the indexes $(k_1, ..., k_d)$ of the coefficients need not be stored because they are uniquely determined for a given m and can be generated automatically. In a d-dimensional case, the ratio of the coefficients stored is $\binom{m+d-1}{d} / m^d \approx \frac{1}{d!}$. That is, we only store around 50%, 17%, and 4% of the m^d coefficients for d=2, 3, and 4, respectively. We incorporate this technique in our implementation.

4. Estimating Size of Join Queries

In this section, we discuss how to estimate the size of a join query using the cosine series. A typical query under consideration may look like "Select Count(*) from $R_1, R_2, …., R_n$ where join-conditions", where the join-conditions have the form of "$R_i.A = R_j.B$ and $R_k.C=R_l.D$ and ….".

4.1 Definitions and Assumptions

Let intervals $[l_A, r_A]$ and $[l_B, r_B]$ be the domains of $R_i.A$ and $R_j.B$, respectively, before normalization, and minAB = $\min(l_A, l_B)$, maxAB = $\max(r_A, r_B)$. By defining the frequency of a value falling outside of its domain to be 0, we can assume both attributes, i.e., $R_i.A$ and $R_j.B$, have the same domain [minAB, maxAB] (with frequencies equal to 0 for those values falling outside of their original domains). Normalization is then performed on the domain [minAB, maxAB], converting it to the interval (0, 1). Hereafter, we assume all pairs of join attribute have the same domains and are normalized to (0, 1).

4.2 Join Size Estimation

Consider a query with an equi-join like "Select COUNT(*) from R$_1$, R$_2$ where R$_1$.A = R$_2$.B". Discussions on queries with multiple equi-joins follow naturally. Let *n* be the domain size for both attributes A and B, $\{a_k\}$ and $\{b_k\}$ be the DCT coefficients of R$_1$.A and R$_2$.B, respectively. The join size, denoted as J, is computed as:

$$J = \sum_{k=0}^{n-1} count_{A_{v_i}} count_{B_{v_i}} . \tag{4.1}$$

where $count_{A_{v_i}}$ and $count_{B_{v_i}}$ are the numbers of tuples whose values are v$_i$ in R$_1$.A and R$_2$.B, respectively. On the other hand, by Parseval's identity [33],

$$\sum_{k=0}^{n-1} \frac{count_{A_{v_i}}}{N_1} \cdot \frac{count_{B_{v_i}}}{N_2} = \sum_{k=0}^{n-1} \frac{a_k}{\sqrt{n}} \cdot \frac{b_k}{\sqrt{n}} \tag{4.2}$$

Consequently, the join size is obtained as

$$J = \frac{N_1 N_2}{n} \sum_{k=0}^{n-1} a_k \cdot b_k \tag{4.3}$$

By using only the first *m* coefficients, J is estimated as:

$$Est = \frac{N_1 N_2}{n} \sum_{k=0}^{m-1} a_k b_k \tag{4.4}$$

As shown by Eq. (4.4), the join size estimate can be derived by adding up the products of corresponding coefficients. The formula for queries with multiple joins is the similar by adding up the products of the corresponding coefficients on the same dimensions.

4.3 Error Analysis

In this section, we give a brief discussion on the number of coefficients needed to guarantee the relative error to be smaller than a threshold e.

We assume both relations have the same size N, for simplicity. Let *n* be the size of join attribute domains. As shown in Eq. (4.4), the join size estimate *Est* of the two relations is calculated as

$$Est = \frac{N^2}{n} \sum_{k=0}^{m-1} a_k b_k = \frac{N^2}{n} + \frac{N^2}{n} \sum_{k=1}^{m-1} a_k b_k . \tag{4.5}$$

As shown in Eq (4.3), we only need the first *n* terms to compute the actual join size J. That is,

$$J = \frac{N^2}{n} \sum_{k=0}^{n-1} a_k b_k = \frac{N^2}{n} + \frac{N^2}{n} \sum_{k=1}^{n-1} a_k b_k . \tag{4.6}$$

We know that $a_0 = 1$, and from Eq. (3.2), $a_k = \frac{1}{N} \sum_{i=1}^{n} count_{v_i} \sqrt{2} \cos k\pi v_i , k \geq 1.$

Since $-1 \leq \cos k\pi v_i \leq 1,$ we derive $-\sqrt{2} \leq a_k \leq \sqrt{2}.$ similarly, $b_0 = 1$ and $-\sqrt{2} \leq b_k \leq \sqrt{2}$. Using the bounds $-\sqrt{2} \leq a_k , b_k \leq \sqrt{2}$, we obtain

$$| J - Est | = | \frac{N^2}{n} \sum_{k=m}^{n-1} a_k b_k | \leq \frac{2N^2(n-m)}{n} . \tag{4.7}$$

The relative error is defined as

$$relative\_error = \frac{| J - Est |}{J} \leq \frac{2N^2(n-m)}{Jn} \tag{4.8}$$

by assuming J > 0. To guarantee the relative error to be smaller than or equal to a given number *e*, from Eq. (4.8), we derive $n - \frac{eJn}{2N^2} \le m$. Consequently, the space requirement to guarantee an error *e* is:

$$m = n - \left\lfloor \frac{eJn}{2N^2} \right\rfloor \tag{4.9}$$

As a simple comparison, the basic sketch [3] has a best case space bound $\Omega(N^2/J)$ and worst case bound $O(N^4/J^2)$[32]. By boosting the basic sketch's worst case bound to $O(N^2/J)$,the skimmed sketch [32] has a space bound of $\Theta(N^2/J)$. However, this bound is valid only when the join size is greater than a sanity bound of $N^{3/2}$ or NlogN [3]. When the join size is small, the required space could be much greater than the bounds given above. Moreover, the skimmed sketch uses extra space to store extracted frequencies; this extra space is in the order of *n* (i.e., O(n)). In general, it is very difficult or impossible to derive tighter bounds for our approach as well as for other approaches because of the diversities of the frequency functions, which are further complicated by the join operations. However, there are some interesting properties that may shed some light on the comparisons. That is, the best and worst cases of our approach happen to be, respectively, the worst and best cases of the sketches'. We discuss these situations in the following.

4.3.1 Best Case Error

The cosine transform approximates smooth distributions better. Therefore, the best case, in terms of estimation accuracy, happens when the join attribute values are uniformly distributed regardless of the attribute domain size *n* is. The cosine coefficients $a_0=1$ and $b_0=1$. As for a_k and b_k, $1 \le k \le m$, they can be derived as:

$$a_k = \frac{1}{N}\sum_{i=1}^{n} count_{V_i} \cdot \sqrt{2} \cdot \cos k\pi_{V_i} = \frac{\sqrt{2}}{n}\sum_{i=1}^{n}\cos\frac{k\pi(2i-1)}{2n} = 0 \tag{4.10}$$

Similarly, $b_k = 0$. Thus, the join size estimate *Est* is

$$Est = \frac{N^2}{n}\sum_{k=0}^{m-1} a_k b_k = \frac{N^2}{n} \cdot a_0^2 \cdot b_0^2 = \frac{N^2}{n} = J \tag{4.11}$$

That is, using only the first term of the transforms, i.e., $a_0=b_0=1$, is already enough to represent the uniform distributions and gives no-error join size estimation.

On the other hand, the sketch methods have their worst case here. They require at least $O(\frac{N^2}{J}) = O(\frac{N^2}{N^2/n}) = O(n)$ space, which makes them not better than the brute-force method.

4.3.2. Worst Case Error

The worst case (of DCT) happens when all the tuples in a data stream have the same and sole join attribute value. Let v_{j_1} and v_{j_2} be the sole join attribute values in the two data streams, respectively. For simplicity, we shall consider only the case where $j_1=j_2=j$. That is, $count_{v_j} = N$, and $count_{v_i} = 0$ for $i \ne j$. Since $J=N^2$ in this case, by Eq. (4.9), the number of coefficients needed to guarantee the relative error is smaller than or equal to e is :

$$m = n - floor(\frac{en}{2}). \tag{4.12}$$

On the other hand, the sketch methods can obtain the exact join size ($J=N^2$) because there is only one value in the attributes. The sketch methods have their lower bound O(1) space here.

From the discussion above, we derive that $\Omega(1)$ and $O(n - floor(en/2))$ are our lower and upper space bounds, respectively.

As observed each method has its strengths and weaknesses. No single method is best for all distributions. Therefore, in the next section we will perform extensive experiments to see how they react to different types of data and which method is likely to cope with more types of data, especially real-life (like) data.

5. Experimental Results

In this section, we report the experimental results of estimating join queries over data streams. Alon et al. [3] proposed the seminal sketch method, which we have called the basic sketch method in this paper. It has become the foundation of many other works [3, 9, 12, 32]. Dobra et al. [9] partition the attribute domains and apply the sketch to sub-domains of the attributes; it requires a priori knowledge of the data distributions and an independence assumption of the join attributes. Since this approach is essentially the basic sketch approach applied to individual sub-domains (with additional assumptions) we will not include it in the comparisons as the results of the basic sketch approach are directly applicable to sub-domains of Dobra's. Ganguly et al. [32] skims the dense frequencies from the original data distributions. It then uses the sketch to estimate the join size of the non-dense frequency portions. Improved accuracy has been reported but extra storage space for the extracted dense frequencies is required, which could be as large as the attribute domain size.

In this paper, we shall compare our method with Alon's basic sketch and Ganguly's skimmed sketch as none of these requires a priori knowledge of data distributions or an independence assumption of join attributes. We have implemented the sketches and the cosine transform in C++. The experiments were run on a PC with a 1,400 MHZ Pentium IV CPU and 1GB memory. Experiments are performed on synthetic data as well as on real-life data.

5.1 Query Description

The following is an example of an equi-join query, specifically, a three-join query, in the experiments:

Select COUNT (*) from R_1, R_2, R_3 , R_4

Where R_1.A = R_2.A and R_2.B = R_3.B and R_3.C = R_4.C

Tuples are read one after another to simulate the arrival of items in the data stream. Cosine coefficients and atomic sketches are updated whenever a tuple arrives. To compute the average errors of each case, each query is executed 200 times, of which each is executed with a different set of relation instances

We compare the accuracies of the methods by using the same amount of space. The space is used to store atomic sketches or the coefficients of our cosine series. For simplicity, instead of using bytes, we use the number of coefficients or atomic sketches to specify the size of storage space. We have adopted the commonly used average relative error as the performance measure. The relative error is defined as |Act(ual) – Est(imate)| / Act(ual). Each result is the average of 200 queries. We have also recorded the estimation time and update speed.

5.2 Experiments on Synthetic Data

5.2.1 Synthetic Data

We generate two types of synthetic data for the experiments. The first type consists of datasets of different characteristics. They are used to explore the strengths and weaknesses of the methods. The second type consists of real-life like datasets, generated by the data generator used in [9, 27]. They are used to assess the potentials of these methods in the real world applications.

In the first type of synthetic data, we use the Zipfian distributions [30] to generate frequencies of attribute values. The Zipfianly distributed frequencies, with a parameter z (or zipf), are generated by the formula: $f_z(i) = \frac{1}{i^z} / \sum_{j=1}^{n} \frac{1}{j^z}$, where $f_z(i)$ is i^{th} frequency value, $1 \le i \le n$. The z values of 0.5 and 1 roughly represent a slightly skewed and a skewed distribution, respectively. We will study how skew can affect the performance of the methods.

Besides the skew, correlations between the join attributes can also influence the performance of the methods. We generate data with different correlations to study their strengths and weaknesses. Independent attributes are generated by using different random mappings (from frequencies to attribute values), while correlated attributes are generated using the same mapping (for positively correlated attributes) or "inverted" mapping (for negatively correlated attributes). Here, positively correlated attributes, say, A and B, refer to roughly the situation where if the frequency of a value x in A is greater than the frequency of another value y in A, then the frequency of x in B is also greater than y in B. Negatively correlated attributes refer to the opposite situation. The smoothness of frequency functions can affect the performance too. We will investigate its effects by comparing their performance on smooth distributions with random distributions. Smooth distributions are generated by orderly mapping frequencies to attribute values.

For the second type of synthetic experiments, we generate real-life like data to assess their potentials for real-life applications. It is argued that real-life data are often correlated and sparsely clustered [9, 27]. Vitter [27] implemented a synthetic data generator to generate relations with such properties, and Dobra [9] extended it to generating correlated joint attributes between relations. Here, we will also use their methods to generate test datasets for our experiments.

Two types of synthetic experiments are performed. The first type aims to find out the strengths and weaknesses of the methods by observing their performance on different types of data. The second type is to assess their potentials for real-world applications by using real-life like data.

5.2.2 Results of Synthetic Data

5.2.2.1 Synthetic Data Type I

Two relations, R_1 and R_2, are generated, each with 10^7 (N) tuples. Each relation has an attribute with a domain size of 10^5 (n). These figures are the same as the experimental setting in related papers [9, 27, 32]. The frequencies of the attribute values in the two relations follow the Zipfian distributions with zipf values 0.5 and 1.0, respectively. Correlations and smoothness are instilled in attribute values using different mappings, as mentioned earlier. It should be mentioned that the skimmed sketch uses additional O(n) space to store extracted dense frequencies. The additional space, which is not reported in the figures presented here, from thousands to 10^5, is much larger than the largest number of DCT's coefficients or atomic sketches used in the experiments. Readers are advised to note the hidden space consumed by the skimmed sketch when interpreting the results.

Figures 1 to 4 show how the methods perform with respect to different types of data, from positively correlated to negatively correlated data. As observed from Figure 1, sketches perform better than the cosine method. Actually, the positively correlated case is a generalization of the self-join case for which the sketch was shown to be most suitable [3]. Recall also from the discussion of Section 4 that sketch's best case happens when all the join attribute values are the same, which is the extreme case of the positive correlations. However, as the positive correlations weaken, their performance degrades and our approach performs better as shown in Figures 2, 3 and 4. For example, with 500 coefficients (or atomic sketches), which is only 0.5% of the attribute domain size ($n=10^5$), the relative errors of the skimmed and basic sketches are 2.7 and 8.3 times greater than the cosine method in the weak positively correlated case (Figure 2), 24.4 and 49.8 times in the independent case (Figure 3) and 3.0 and 8.9 times in the negatively correlated case (Figure 4).

The data set used in Figure 2 is obtained by permuting only 10% of the frequencies of R_2 in Figure 1. The permutation introduces some randomness and weakens the positive correlations. Notice how large difference randomness has made in these two figures. Note that in Figure 1 no single tuple in the data set violates the positive correlation. The way to permute the frequencies also may affect the estimation performance.

Let us now examine the impact of the smoothness of distribution functions on the performance by comparing Figures 1 and 5. The data used in these two figures are basically identical, except that the frequency functions of R_1 and R_2 in Figure 1 are rough (due to the random mapping between frequencies and attribute values) while they are smooth in Figure 5 (due to the orderly mapping between frequencies and attribute values). The two relations are positively correlated just like in Figure 1. As observed, smoothness plays in DCT's favour. The cosine method has improved its performance a lot here, as compared to Figure 1, on this strongly correlated dataset due to the smoothness. For example, with 500 coefficients, the cosine method yields an average error of 56.24% in Figure 4, down from 96.58% in Figure 1. As expected, smoothness has no effect on the sketches since sketches do not approximate distributions.

Let us now examine the effects of skew by comparing Figures 3 and 6. When the distributions become skewer, that is, the zipf value of R_2 changes from 1.0 (in Figure 3) to 1.5 (in Figure 6) with R_1 remaining the same, all methods suffer from performance degradation. For example, with 500 coefficients, the relative errors of the cosine, skimmed sketch, and basic sketch increase from 9.98%, 92.40%, and 333.09% (in Figure 3) to 24.21%, 158.76%, and 837.85% (in Figure 6), respectively. The skew does not seem to play particularly in favour of any method. But still the errors of skimmed and basic sketches are 7.5 and 39.5 times greater than ours, respectively.

As a short summary of this qualitative study, we observe that the sketch methods are suitable for strong positively correlated data, while our approach is more suitable for from weak positively correlated, random, to negatively correlated data. In addition, our approach can also benefit from the smoothness of distributions functions, which often exhibits in the real-life data, such as the distributions of ages and salaries of employees in a company.

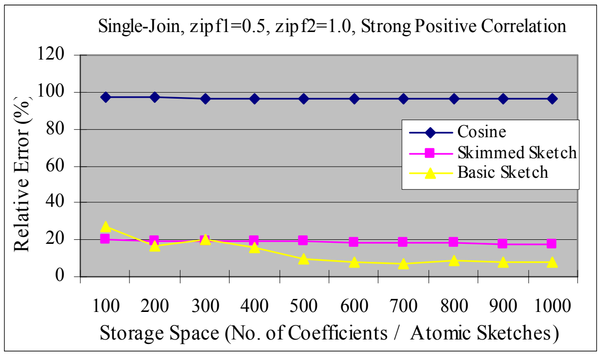


Figure 1. Strong Positively Correlated Attributes

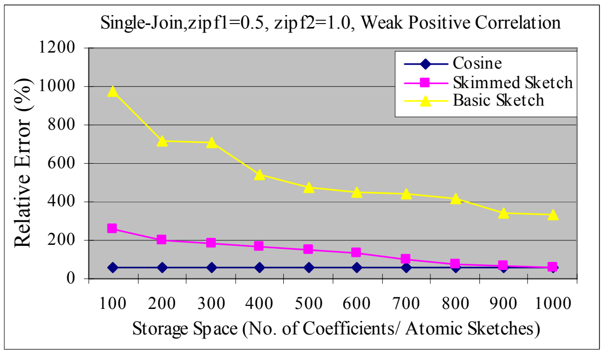
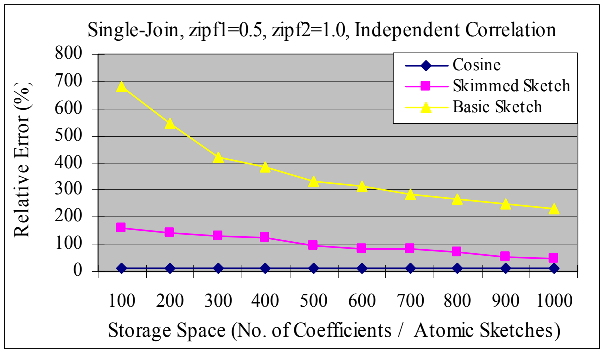


Figure 2. Weak Positively Correlated Attributes



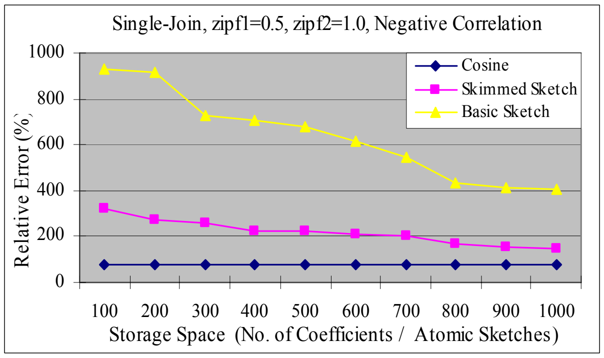


Figure 3. Independent Join Attributes

Figure 4. Negatively Correlated Attributes

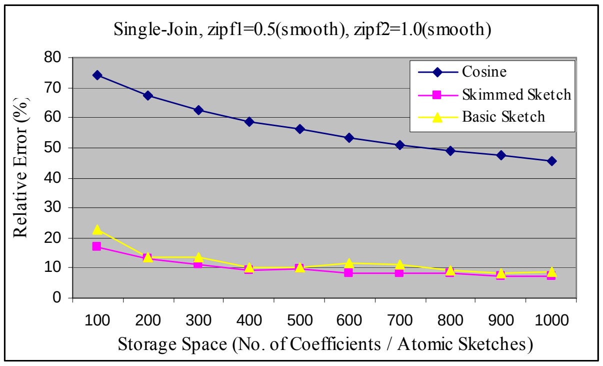


Figure 5. Strong Positively Correlated Attributes with Smooth Distributions

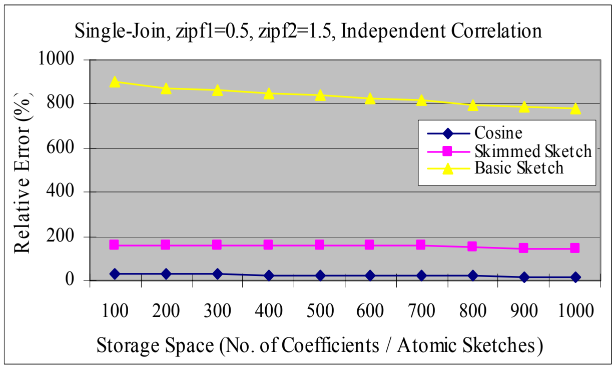


Figure 6. Independent Join Attributes with Skewer distributions

5.2.2.2 Synthetic Data Type II

The purpose of this experiment is to assess the potentials of these methods in real-life applications. We implemented the data generator proposed by Vitter, et al. [27] and extended by Dobra, et al. [9] to generate real-life like data. The data are clustered and positively correlated. The datasets are generated by distributing tuples across and within the randomly picked rectangular regions (clusters) in the multi-dimensional attribute space of a relation. We chose the same parameter setting as in [9]: skew across regions (z_{inter}) =1.0 and skew within each region (z_{intra}) =0.0-0.5; number of regions=10 and 50 (the later is in addition to Dobra's [9]); size of each domain=1024; size of each relation=10^7, volume of each region =1,000 – 2,000 and perturbation parameter p=0.5 - 1.0.

Figure 7 and 8 show the results of single-join queries over clustered and correlated datasets with different numbers of clusters. Again, the cosine method outperforms the sketch methods. For example, with 500 coefficients in Figure 7, our method generates an average error of 0.60% while the errors of skimmed sketch and basic sketch method are 7.98% and 8.24%, respectively, which are 13.2 and 13.6 times greater than ours. Figure 8 shows a similar result as the number of clusters increases to 50. The superiority of our method in these experiments is mainly due to the not extremely strong positive correlations (as compared to that in Figure 1) exhibited in the data although the clusters are still very positively correlated. Randomness sets in when the centers of the clusters are selected randomly within their respective shrunk regions in the correlated relations. Clustered data could also make the distribution curves a little smoother than a completely random distribution.

Similar results are observed in the two-join query cases, as shown in Figures 9 and 10. Please note that the attributes space is 1024^2 =10^6 in the two-join cases and thus more coefficients are needed to capture the distribution information. Suffering from the large attribute space, all methods degrades compared to the single-join case. For example, in the two-join queries, with 1,000 coefficients, which is only 0.1% of the attribute space, the errors of the cosine method are 26.27% and 12.65% for 10 and 50 clusters, respectively, while the errors are only 0.28% and 0.96% in the single-join cases. Our method again performs much better than the sketch methods in both cases. Their errors are as large as 142.46% (skimmed sketch) and 147.56% (basic sketch) for the 10-cluster dataset, which are 5.4 and 5.6 times greater than ours. For the 50-cluster dataset, our result is 12.65%, while theirs are 139.89 % and 180.37% for the skimmed and basic sketches, respectively; theirs are 11.1 and 14.3 times larger than ours. The ratios do not differ much from those of 10-cluster set.

The experimental results of three-join queries are shown in Figures 11 and 12. Since too few resulting tuples could generate large estimation relative errors, the attribute domain sizes are reduced to 400, instead of 1,024 as in the previous experiments. Figure 11 shows the results of the three-dimensional 10-cluster dataset. With 1,000 coefficients, the cosine yields an average error of 86.26% and the error decreases to 9.03% when 20,000 coefficients are used. As for the skimmed and basic sketches, the errors are too large to be useful for coefficients less than 10,000. Even with 20,000 coefficients, their relative errors are still 2.2 and 3.0 times larger than ours. Similar results are also observed in Figure 12 for the 50-cluster dataset.

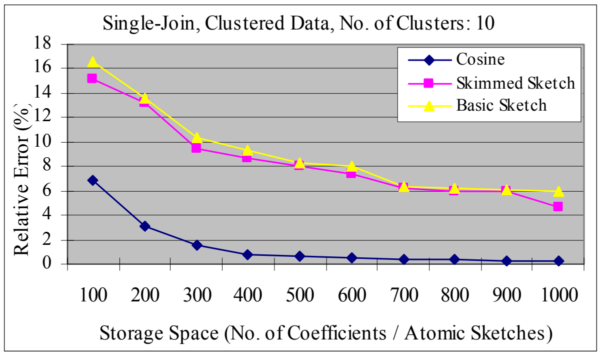


Figure 7. Single-Join Query, Cluster Data, No. of Clusters: 10

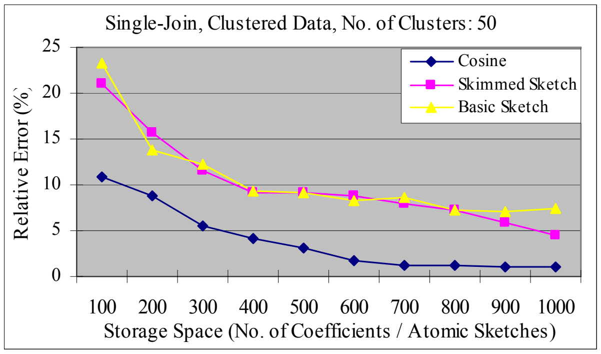


Figure 8. Single-Join Query, Cluster Data, No. of Clusters: 50

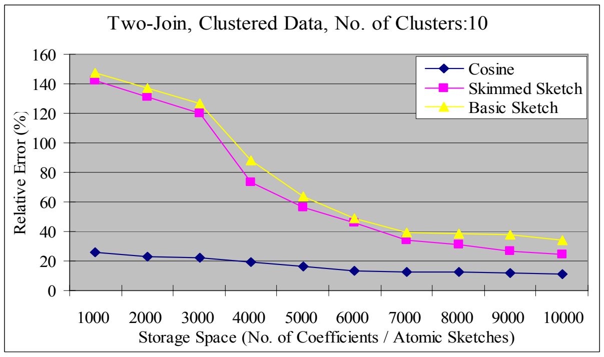


Figure 9. Two-Join Query, Cluster Data, No. of Clusters: 10

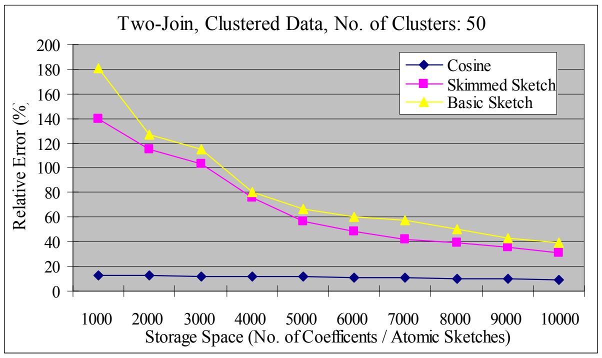


Figure 10. Two-Join Query, Cluster Data, No. of Clusters: 50

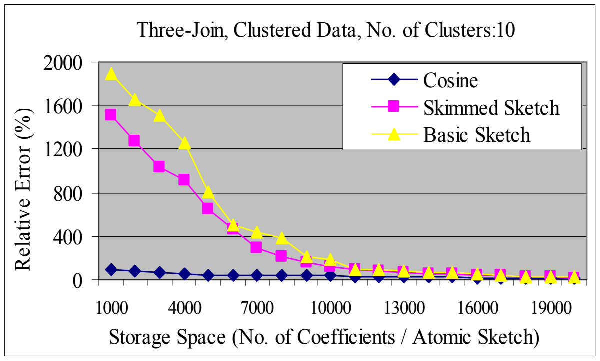


Figure 11. Three-Join Query, Cluster Data, No. of Clusters: 10

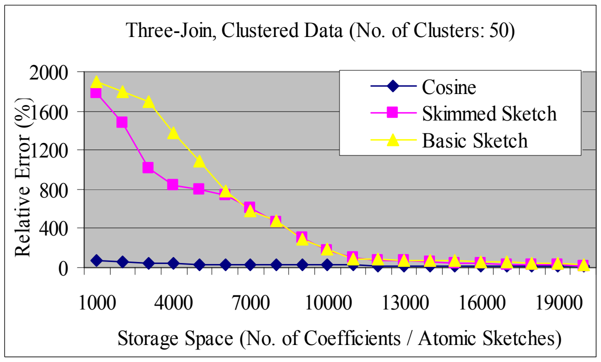


Figure 12. Three-Join Query, Cluster Data, No. of Clusters: 50

5.3 Experiments on Real Data

5.3.1 Real Datasets

Three real-life datasets - the Current Population Survey (also used in [9]), the Income and Program Participation Survey from the Bureau of Census, and the Internet Traffic Archive, are used for the experiments. The Current Population Survey (also denoted as real dataset I), containing 133,696, 143,598, and 135,872 tuples for the January, February, and March 2004, respectively, are used for the experiments. The two attributes selected for the experiments are Age and Education, whose ranges are [1, 99] and [1, 46], respectively.

For the Income and Program Participation Survey (also denoted as real dataset II), we use the attributes SSUSEQ (Sequence Number of Sample Unit), WHFNWGT (Weight for household reference person), and THEARN (Total household earned income) for the years 2001 and 2004. Their ranges are [1, 50,000], [1, 9,999] and [1, 1,500], respectively. The 2001 data has 361,046 tuples and 2004 data 441,849 tuples.

The Internet Traffic Archive (also denoted as real dataset III) contains the traces of internet network traffic. We use the DEC-PKT traces that record four hours' worth of the wide-area traffic between the Digital Equipment Corporation and the rest of the world for our study. Records of the TCP and UDP packets in the first three hours are used in our experiments. The sizes of the TCP files are 94MB, 113MB and 128MB, and size of the UDP files are 21.4MB, 21.4MB and 26.9MB respectively. Two attributes, the source and destination hosts are used as the join attributes. Their ranges are [0, 2394] in TCP files and [0, 7327] in UDP files.

5.3.2 Results on Real Data

The experimental results of single-joins on the real dataset I are shown in Figure 13. The join attribute is "Age". All methods give good estimation, as shown in Figure 13. For example, with only 20 coefficients or atomic sketches, the relative errors of the cosine, skimmed, and basic sketches, are already as low as 4.71%, 8.08%, and 16.05%, respectively. Two reasons contribute to this high accuracy are the small attribute domain (0-99) and large number of resulting tuples (0.26 billion). In general, the smaller the domain size, the better the approximation.

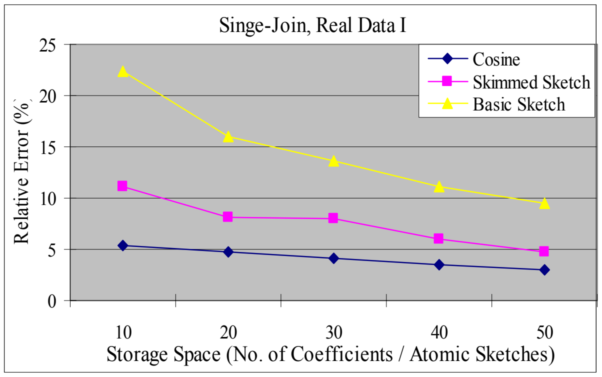


Figure 13. Single-Join Query, Real Data I

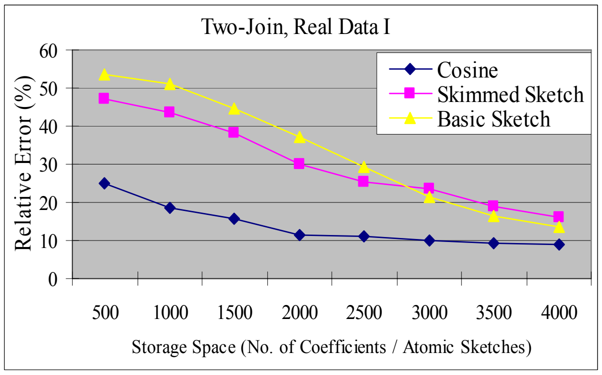


Figure. 14. Two-Join Query, Real Data I

For two-join queries, the resulting size is 1.02×10^8. As shown in Figure 14, the relative error of ours is less than 15% with only 1,500 coefficients while the relative errors of the skimmed and basic sketches are as large as 38.1% and 44.81%. We attribute the superiority of the cosine method to (1) the not extremely strong positive correlations like that in Figure 1, where there is absolutely no tuple violating the positive correlation, and (2) some smoothness in the distribution curves (as compared to the totally rugged curves). In fact, the positive correlations between the two "Age" attributes and the two "Education" attributes are rather strong, but the cosine method still outperforms the sketches methods.

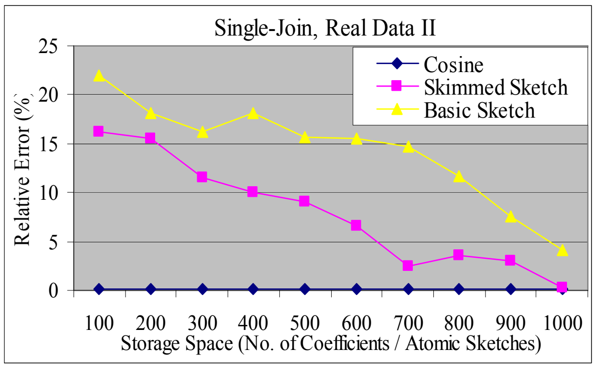


Figure 15. Single-Join Query, Real Data II

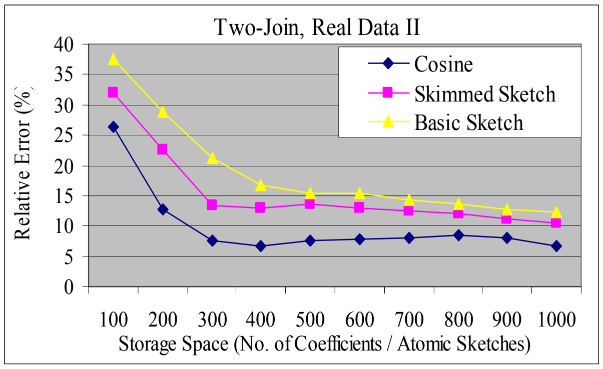
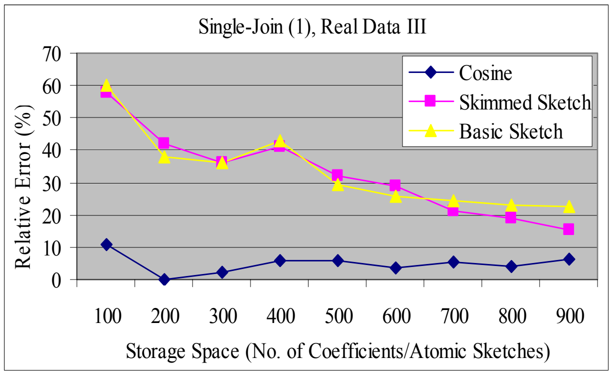


Figure 16. Two-Join Query, Real Data

Figures 15 and 16 show the results on the real dataset II (the Income and Program Participation Survey data). The single-join query (Figure 15) is performed on the SSUSEQ attribute, which has a rather large domain [1, 50,000]. Our method achieves high accuracy with just a few coefficients while the two sketch methods could not provide satisfactory estimation until using 10 times more coefficients. With 100 coefficients, the relative errors of the cosine, skimmed sketch, and basic sketch are 0.12%, 16.23%, and 22.12%, respectively. The skimmed and basic sketches generate 136 and 185 times larger error than our method. When the number of coefficients increases to 1,000, the relative errors of the cosine, skimmed sketch, and basic sketch decrease to 0.07%, 0.29%, and 4.06%, respectively. Cosine provides much better estimation all the time. Figure 16 shows the results of two-join queries on attributes WHFNWGT and THEARN. Again, the cosine method outperforms the skimmed and basic sketch methods. With 1,000 coefficients, the relative error of cosine is 6.6% while the relative error of the skimmed and basic sketch is 10.5% and 12.3%.

The experimental results on real dataset III (Internet Traffic Archive) are shown in Figures 17 to 20. The single-join is performed on the source (Figure 17) and destination (Figure 18) host attributes of the TCP datasets. In Figure 17, with 100 coefficients, the cosine has an error of 10.79% while the skimmed and basic sketches' errors are 57.6% and 60.1%, respectively. With 900 coefficients, the relative error of the cosine is 6.10% while 15.3% and 22.6% for the skimmed and basic sketches, respectively.

The results of two-join experiments are shown in Figures 19 (for the TCP files) and 20 (for the UDP files). The performance of the skimmed and basic sketches is again much worse than our method. In Figure 19, with 1,500 coefficients, our method generates an error of 0.57% while the skimmed and basic sketch still has errors as large as 66.04% and 93.72%, respectively. A similar result is observed in Figure 20 for the two join experiment on the UDP files.



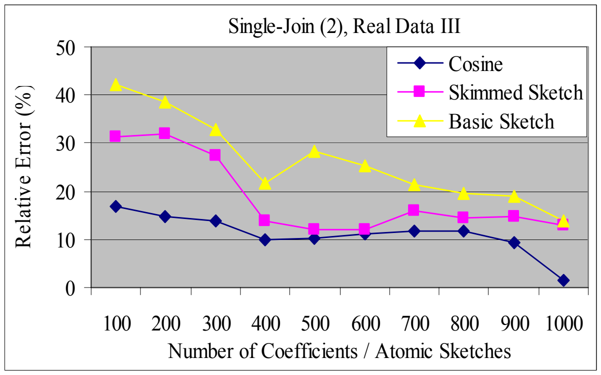


Figure 17. Single-Join Query (1), Real Data III

Figure 18. Single-Join Query (2), Real Data III

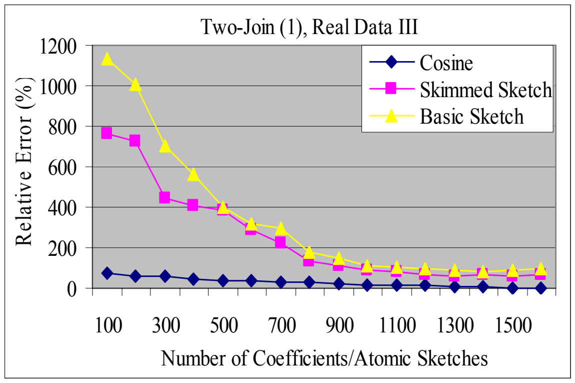


Figure 19. Two-Join Query (1), Real Data III

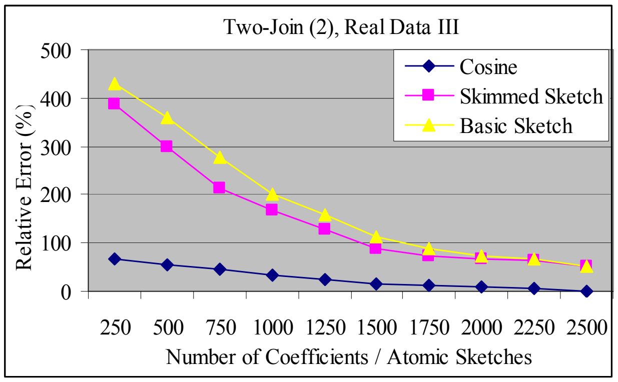


Figure 20. Two-Join Query (2), Real Data III

5.4 Computation Speed

When a tuple arrives, we immediately update the coefficients, following Eq. (3.4). On the average, it takes 0.32 μs to update one coefficient. So, even for the case with 10,000 coefficients, it takes only 3.2 ms to do the job. To estimate join sizes, we follow Eq. (4.6). On the average, it takes about 0.4 ms to derive an estimate from 10,000 coefficients. As for the sketch methods, to update 10,000 atomic sketches, it takes about 1.0 ms, which is faster than ours; this is due to simpler computations involved in updating atomic sketches. But to derive an estimate from 10,000 atomic sketches, it would take 1.6 ms, as compared to our 0.4 ms, because they need to find the median of a large number of group means.

It is worth mentioning that all the methods can update their coefficients in a batch fashion. That is, updates to the distributions can be stored aside and then applied to the coefficients or atomic sketches all in once. As a result, there should not be any problem for all these methods to cope with the fast on-line one-pass data streams.

6. Conclusions and Future Work

In this paper, we discuss approximate aggregation query processing over data streams with limited storage space. Specifically, we concentrate on the estimation of aggregation queries with equi-joins. We use cosine series to approximate the data distributions of the data streams and then use them to estimate the size of equi-join queries. Experimental results have shown that our approach produces much more accurate estimates than sketches for most cases. We have also demonstrated that our approach can be updated dynamically and quickly. The proposed method is well suited for on-line approximate aggregation equi-join queries over continuous data streams. Our method can also be applied to non-equal-joins, range, and point queries.

References

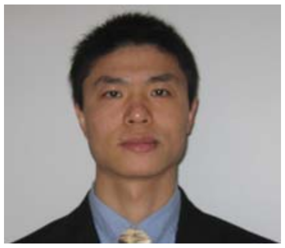
[1] S. Acharya, P. Gibbons, V. Poosala, and S. Ramaswamy. “Join synopses for approximate query answering”, In *SIGMOD. ACM Press*, 1999, pp275-286.

[2] N. Alon, Y. Matias and M.Szegedy. “The Space Complexity of Approximation the Frequency Moments”, In *Proc of 28th Annual ACM Symp.on the Theory of Computing*, May 1996, pp 20-29.

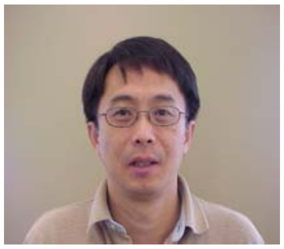
[3] N. Alon, P.B Gibbons, Y. Matias and M.Szegedy. "Tracking Join and Self-join Sizes in Limited Storage", In *proc of the 18th ACM SIGACT-SIGMOD-SIGART Symp. on the Principles of Database Systems*, May 1999, pp.10-20.
[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. "Models and Issues in Data Stream Systems". *21st ACM SIGACTSIGMOD-SIGART Symposium on Principles of Database Systems*, Madison, June 2002, pp 1-16.
[5] S. Babu and J. Widom. "Continuous queries over data streams". *SIGMOD Record*, 2001, 30(3): pp109-120.
[6] A. Bulut and A. K. Singh. "SWAT: Hierarchical stream summarization in large networks". In *IEEE 19th International Conference on Data Engineering*, Bangalore, India, Mar 2003 pp303-314
[7] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. "Approximate query processing using wavelets". In *Proceedings of 26th International Conference on VLDB*, Cairo, Egypt, 2001, pp111-122.
[8] C.Cui, An Introduction to Wavelets, Academic Press, 1992.
[9] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. "Processing complex aggregate queries over data stream", In *ACM-SIGMOD,* Madison, Wisconsin, June 2002, pp61-72.
[10] E. Dudewicz and S. Mishra, Modern Mathematical Statistics, John Wiley & Sons. Inc, 1988.
[11] P. Gibbons and Y. Matias. "New sampling-based summary statistics for improving approximate query answers". In *ACM SIGMOD 1998*, Seattle, Washington , June 1998, pp331-342.
[12] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan and M. Strauss, "Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries", In *Proc. of VLDB*, 2001, Roma, Italy, Sep, 2001 , pp79-88.
[13] S. Guha, N. Koudas, and K. Shim. "Data-streams and histograms". In *Proc. ACM Symp. on the Theory of Computing (STOC)*, 2001, Hersonissos, Greece, 2001, pp471-475
[14] P. Haas, J. Naughton, S. Seshadri, L. Stokes, "Sampling-based estimation of the number of distinct values of an attribute", *Proc. of VLDB*, Zurich, Switzerland, September, 1995, pp311-322.
[15] W-C. Hou, G. Ozsoyoglu and B. Taneja, "Statistical Estimators for Relational Algebra Expressions", *Proceedings of the 7th ACM Symposium on Principles of Database Systems*, Austin TX, March 1988, pp276-287.
[16] P. Hall, "Orthogonal Series Distribution Function Estimation with Applications", *Journal of the Royal Statistical Society*, Series B, Vol.45, No.1 pp81-88, 1983
[17] Y. Ioannidis and S. Christodoulakis. "Optimal Histograms for Limiting Worst-Case Error Propagation in the Size of Join Results". *ACM Transactions on Database Systems,*December 1993, Vol. 18, No. 4, 709-748.
[18] Y. E. Ioannidis and V. Poosala. "Balancing Histogram Optimality and Practicality for Query Result Size Estimation". In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, San Jose, California, 1995, pp233-244.
[19] Y. E. Ioannidis and V. Poosala "Histogram-Based Approximation of Set-Valued Query Answers", *Proc. of 25th VLDB Conference*, pp174-185
[20] N. Koudas, S. Muthukrishnan and D. Srivastava. **"**Optimal Histograms for Hierarchical Range Queries (Extended Abstract) (2000)", *Proc. of the 19th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems,* Dallas, Texas, United States, 2000, pp196 - 204
[21] J-H. Lee, D-H. Kim and C-W Chung, "Multi-dimensional Selectivity Estimation Using Compressed Histogram Information", *SIGMOD 1999*, pp205-214.
[22] R. Lipton, J. Naughton and D. Schneider, "Practical Selecting Estimation through Adaptive Sampling", *ACM SIGMOD 1990*, Atlantic City, NJ, May 1990, pp1-11.
[23] Y. Matias, J. S. Vitter, and M. Wang. "Wavelet-Based Histograms for Selectivity Estimation", In *Proceedings of the ACM SIGMOD Conference 1998*, Seattle, Washington , June 1998, pp 448-459.
[24] Y. Matias, J. Scott Vitter and M. Wang, "Dynamic Maintenance of Wavelet-Based Histograms". *Proc 26th VLDB Conference 2000*, Cairo, Egypt, 2000, pp101- 110
[25] V. Poosala and Y.E.Ioannnis, "Selectivity Estimation Without Attribute Values Independence Assumption", *Proc 23rd VLDB Conference, Athens,Greece, 1997*, pp 486- 495

[26] TPC benchmark D, (decision support, 1995)
[27] J.S. Vitter and M. Wang. "Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets." *In SIGMOD*, 1999, pp193- 204
[28] Y-L Wu, D. Agrawal and A. E. Abbadi, "Applying the Golden Rule of Sampling for Query Estimation", In *ACM SIGMOD 2001,* Santa Barbara, California, May 2001*,* pp 449- 460
[29] F. Yan, W-C. Hou, Q. Zhu, "Selectivity Estimation Using Orthogonal Series", *8th International Conference on Database Systems for Advanced Applications (DASFAA),* Kyoto, Japan, March, 2003, pp 157-164.
[30] G.K. Zipf, *Human Behavior and the Principle of Least Effort* (Addison-Wesley, Reading, MA, 1949)
[31] M. Pinsky, "*Introduction to Fourier Analysis and Wavelet*", Brooks/Cole Thomson Learning, Inc., 2002
[32] S. Ganguly, M. Garofalakis, R. Rastogi, "Processing Data-Stream Join Aggregates Using Skimmed Sketches", *Proc of EDBT*, Heraklion-Crete, Greece, March 2004, pp. 569-586
[33] E. Issacson and H. B. Keller, *Analysis of Numerical Methods Theorem 3*, Dover Publications, 1994, P. 238
[34] W. L. Briggs and V. E. Henson, *DFT : an owner's manual for the discrete Fourier transform*, Philadelphia : Society for Industrial and Applied Mathematics Published, 1995.
[35] *http://www-static.cc.gatech.edu/projects/disl/specialProjects/StreamJoins.htm*

Zhewei Jiang is currently a Ph.D student in the Computer Science Department at Southern Illinois University, Carbondale, IL, USA. Her interests are in databases and data mining.



Cheng Luo is currently a Ph.D student in the Computer Science Department at Southern Illinois University, Carbondale, IL, USA. His interests are in databases and data mining



Wen-Chi Hou received the MS and PhD degrees in computer science and engineering from Case Western Reserve University, Cleveland Ohio, in 1985 and 1989, respectively. He is presently an associated professor of computer science at Southern Illinois University at Carbondale. His interests include statistical databases, mobile databases, XML databases, and data streams.

Feng Yan received his MS in computer science and PhD in Mathematics from Southern Illinois University at Carbondale in 1999. He is currently a senior quantitative analyst at FPL Energy. His interests are in statistical databases, data mining, optimization and stochastic calculus and its applications.

Qiang Zhu received his PhD degree in computer science at the University of Waterloo, Canada, in 1995. He is currently an associate professor of computer and information science at The University of Michigan, Dearborn, MI, USA. He is also an IBM CAS Faculty Fellow at the IBM Toronto Laboratory. His research interests include database query optimization, data streams, multidimensional indexing, self-managing database systems, data mining, and Web data management.

Chih-Fang Wang received his PhD in computer engineer at University of Florida in 1998. His current research interests include sequential, parallel and distributed algorithms, data structures, high performance computing, optical networks, quantum computing, DNA computing, bioinformatics, wireless/mobile security, data mining, and mobile agents and secure mobile agent platforms.