Iterated DFT Based Techniques for Join Size Estimation *

Kamil Saraç *Ömer Eğecioğlu* *Amr El Abbadi*

Department of Computer Science
University of California Santa Barbara
e-mail: $\{ksarac, omer, amr\}$@cs.ucsb.edu

Abstract

Novel techniques based on the Discrete Fourier Transform are proposed to estimate the size of relations resulting from join operations. For the special case of self join the proposed algorithm gives the exact join size using logarithmic space. A generalization to compute the join of arbitrary relations is then used to develop two tree-based techniques that provide a spectrum of algorithms which interpolate storage requirements versus accuracy of the estimation obtained. Finally, we present experimental results to exhibit the effectiveness of our approach.

1 Introduction

Query optimization is an important issue in Database Management Systems (DBMSs). For a given query, there may be several ways of executing it. Finding all these different access plans and deciding the best one may not be practical for a user. As a result, DBMSs need to perform this complex optimization process by themselves. For this reason, most DBMSs use a component called query optimizer. The aim of query optimizers in relational database systems is to select the most efficient way among all possible ways of executing a query. In practice this process requires approximating the cost of possible execution sequences and selecting the cheapest one in terms of a cost metric, which is usually the cumulative size of intermediate operations. Even though these possible ways result in the same answer, their execution sequence may be different, and hence they may result in different execution costs. A necessary procedure in estimating the execution cost of a query is to estimate the result size of each operation in each possible execution sequence. This information depends on the frequency distribution of the underlying data. In practice, most database systems store summarized information about the frequency distribution (or find it dynamically when needed) and use this information to estimate the result size of various operations and then the cost of the executions. The accuracy of this approximation is crucial to the performance of the database systems. A small error introduced in the execution of a portion of the query can grow exponentially and affect the overall performance considerably [6].

Several techniques (sampling based, parametric and histogram based) have been proposed in the literature for estimating query result sizes. The survey by Mannino, Chu and Sager is a good reference on the early work on database size estimation[10]. These techniques present various trade-offs in terms of storage, precision, run-time overhead, and make varying assumptions about the distribution of the underlying data. Various *sampling* techniques that operate at run time and compute estimates based on random samples of data have been proposed [9, 4]. *Parametric* techniques [15] assume mathematical distributions such as normal, uniform, Poisson, or Zipf. The underlying assumption is that the data follows the characteristics of the used parametric distribution. *Histogram* based techniques [11, 5] divide attribute value domains into buckets and then record bucket size information as a histogram. Ioannidis and Poosala [7] used *serial* histograms which are constructed by grouping attribute values that have similar frequencies. The construction of serial histograms in general is computationally expensive. A practical heuristic approximation is the *end-biased* histogram, where most frequent values are stored exactly and the rest averaged out [7].

In this paper, we propose novel techniques based on the *Discrete Fourier Transform (DFT)* to estimate the size of relations resulting from join operations. The basic idea is the iterated application of DFT to attribute values modulo the phase information which allows a logarithmic size representation of a given vector. The first version of this method called *Self Join using Absolute Value (SJAV)* is exact for self join operations. For arbitrary join operations it provides an upper bound on the size of the join, we refer to this as *Approximation by Absolute Value (AAV)*. The generalizations of AAV, *Tree Approximation Algorithms (TAA)* and *Tree Approximation Algorithms with Truncation (TAAT)* are built upon a binary tree representation of the vectors obtained through the iterated application of the DFT. TAA uses AAV as a subprocedure at the lower levels of the tree, whereas $TAAT$ is obtained by truncating the tree at an appropriate level. Both TAA and $TAAT$ provide a spectrum of algorithms that interpolate storage requirements versus accuracy of the estimates obtained.

This paper is organized as follows. In Section 2 we give the problem statement. In Section 3 we present fundamental properties of the DFT that are used in our algorithms. The algorithms are developed in Sections 4 and 5. We evaluate

*This research was partially supported by the NSF/NASA/ARPA under grant number IRI94-11330.

the performance of the algorithms in Section 6 and conclude the paper in Section 7.

2 Problem Formulation

We first describe the problem of join size estimation in databases. The development and motivation closely follows that of Ioannidis and Poosala [5, 7]. Consider a database with relations $R_0, R_1, \ldots, R_n$. Let a_i, a_{i+1} be attributes of relation R_i. A tree function-free equality-join query Q is defined as

$$\begin{aligned} Q &= (R_0.a_1 = R_1.a_1 \textit{ and } R_1.a_2 = R_2.a_2 \textit{ and} \ldots \textit{and} \\ &\quad R_{n-1}.a_n = R_n.a_n). \end{aligned}$$

The attributes $a_1, a_2, \ldots, a_n$ are called *join attributes*. Relations R_i, $2 \le i \le n-1$ participate in Q with two attributes a_i and a_{i+1} and relations R_0 and R_n with one attribute, a_1 and a_n respectively.

In order to optimize the execution of query Q, the size S of the resulting relation must be estimated. Frequency matrices T_i keep track of the joint-frequency distribution of attributes a_i and a_{i+1} of R_i. An exact expression for S is the matrix product

$$S = T_0 T_1 \ldots T_n, \tag{1}$$

but the storage of the frequency matrices and exact computation of this product is too costly.

We follow the common assumption of *attribute independence* formulated in [2] and adopted by most database systems. According to this assumption, the distribution of attribute values are independent of each other. Therefore, if v_i and h_{i+1} are the frequency distributions of individual attributes a_i and a_{i+1} in R_i, then the frequency matrix T_i can be written as the outer product

$$T_i = \frac{1}{|R_i|}(v_i^T \times h_{i+1})$$

where v_i^T is transpose of v_i and $|R_i|$ represents the number of tuples in R_i. This assumption and associativity enables the calculation of S from

$$(h_1 v_1^T)(h_2 v_2^T) \cdots (h_n v_n^T). \tag{2}$$

Note that $h_i v_i^T = < h_i, v_i >$ is the standard inner product of the vectors $X = h_i$ and $Y = v_i$.

3 Preliminaries

The main motivation for our approach is the use of DFT based techniques to estimate each of the factors in (2), and therefore (1). DFT has been used in various estimation problems in database research such as dimension reduction for searching as well as indexing high dimensional data [1, 14].

Consider an N-dimensional real vector $X = (x_1, x_2, \ldots, x_N)$. The *Discrete Fourier Transform (DFT)* of X is the N-dimensional complex vector $\hat{X} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)$ given by $\hat{X} = FX$ where $F = (1/\sqrt{N}) \| \ \omega^{(i-1)(j-1)} \|$ is the $N \times N$ Fourier matrix with $\omega = \cos(2\pi/N) + I \sin(2\pi/N)$, and $I = \sqrt{-1}$. It is well known that the DFT of X can be computed using the Fast Fourier Transform in $O(N \log N)$ arithmetic operations. The fundamental properties of the DFT that we make use of are summarized below [3, 12]. If we start with an N-dimensional nonnegative real vector X with transform $\hat{X}$, then

a) $\hat{x}_1 = (x_1 + x_2 + \cdots + x_N)/\sqrt{N}$ is a nonnegative real number.

b) $\hat{x}_i$ and $\hat{x}_{N-i+2}$ are conjugate complex numbers for $i = 2, 3, \ldots, \lceil \frac{N}{2} \rceil$.

In our presentation, we assume that N is odd for simplicity, although our techniques extend to the case of arbitrary N. If we write $N = 2m+1$, then the coefficient $\hat{x}_1$ is nonnegative real, and the lists $(\hat{x}_2, \hat{x}_3 \ldots, \hat{x}_m)$ and $(\hat{x}_N, \hat{x}_{N-1}, \ldots, \hat{x}_{m+1})$ pair up exactly in conjugate pairs. The most important property of the DFT we use is Parseval's identity [3]:

Theorem *Suppose X and Y are two N-dimensional real vectors. Then $< X, Y > = < \hat{X}, \hat{Y} >$ where $<\ ,\ >$ denotes the standard complex inner product.*

We use Parseval's identity in the following expanded form for real vectors X and Y. Suppose $\hat{X} = \alpha + I\beta$ is the decomposition of $\hat{X}$ into its real and imaginary components. Similarly, write $\hat{Y} = \gamma + I\delta$. Then Parseval's identity implies that

$$< X, Y > \ = \ < \alpha, \gamma > + < \beta, \delta > . \tag{3}$$

4 Development of Iterated DFT based Algorithms

4.1 Self Join using Absolute Values of DFT Coefficients

We start by motivating our approach by using the simple case of a self join, i.e. when a relation is joined with itself on the same attribute. We start with the self join case because it is simple and motivates our general approach. Furthermore, the self join is of particular interest for any application where a dataset is joined with itself on an attribute to find items that share a particular characteristic. This arises in several scientific dataset applications. For example consider a hydrological set of data samples taken over a time period of ocean temperatures. A scientist may be interested in identifying various datasets that share the same temperature. Our approach is particularly efficient for such queries.

Consider a relation R with an attribute a and let X be the frequency vector of a. We will assume for ease of representation that X is N-dimensional where N is of the form of $N = 2^k - 1$ for some integer k. The size S of the result for self join is the inner product of the frequency vector X by itself, i.e. $< X, X >$. Using Parseval's identity in the form (3) with $Y = X$ we have

$$\begin{aligned} < X, X > &= \sum_{i=1}^{N} x_i^2 = < \alpha, \alpha > + < \beta, \beta > \qquad (4) \\ &= \sum_{i=1}^{N} (\alpha_i^2 + \beta_i^2) = \sum_{i=1}^{N} |\hat{x}_i|^2 = < \hat{X}, \hat{X} >, \end{aligned}$$

where $|\hat{x}_i| = \sqrt{\alpha_i^2 + \beta_i^2}$ is the absolute value of $\hat{x}_i$. Hence the inner product of a real vector by itself can be calculated either by summing the squares of its values or by summing the squares of the absolute values of its complex DFT coefficients. However the DFT coefficients $\hat{x}_2, \hat{x}_3, \ldots, \hat{x}_N$ of a real vector occur in complex conjugate pairs for odd N. In particular, let a complex vector $\hat{X} = (\hat{x}_1, \hat{x}_2, \hat{x}_3, \ldots, \hat{x}_{N-1}, \hat{x}_N)$ be the DFT of a real vector X with $N = 2^k - 1$. Then the complex numbers $\hat{x}_2$ and $\hat{x}_N$ are complex conjugate pairs with $x_2 = \alpha_2 + I\beta_2$ and $x_N = \alpha_N + I\beta_N$ where $\beta_2 = -\beta_N$. Similarly $\hat{x}_3$ and $\hat{x}_{N-1}$ and so on are complex conjugate pairs.

Since conjugate numbers have the same absolute value, $|\hat{x}_i|^2 = |\hat{x}_{N-i+2}|^2$ for $2 \leq i \leq (N+1)/2$. Therefore combining with (4),

$$\sum_{i=1}^{N} x_i^2 = \hat{x}_1^2 + 2\sum_{i=2}^{\frac{N+1}{2}} |\hat{x}_i|^2. \tag{5}$$

This reduces the calculation of the desired sum on the left of (5) to the calculation of $\sum_{i=2}^{\frac{N+1}{2}} |\hat{x}_i|^2$. This latter sum is the inner product of the $(N-1)/2$-dimensional vector $X_1 = (|\hat{x}_2|, |\hat{x}_3|, \ldots, |\hat{x}_{(N+1)/2}|)$ by itself. Since this vector is also real, we can take *its* DFT and iterate this process by using Parseval's identity and (5) at each step. When $N = 2^k - 1$, the next vector to be considered has $(N-1)/2 = 2^{k-1} - 1$ elements, the one after that $2^{k-2} - 1$, and so on. Therefore this process ends in $k = \log(N+1) - 1$ iterations. Let $X_0 = X$ and define X_j to be the real vector $X_j = (x_{j1}, x_{j2}, \ldots, x_{jN_j})$ of length $N_j = 2^{k-j} - 1$ obtained after the j-th iteration, where the j-th iteration consists of taking the DFT of the N_{j-1}-st dimensional real vector X_{j-1}, and consequently setting $X_j = (|\hat{x}_{j-1,2}|, |\hat{x}_{j-1,3}|, \ldots, |\hat{x}_{j-1,N_j+1}|)$.

Let $\hat{x}_{01} = \hat{x}_1$, and let $\hat{x}_{j1}$ be the first element of the DFT $\hat{X}_j$ of the vector X_j. By iterating (5), we obtain the sequence of nonnegative real numbers $\hat{x}_{01}, \hat{x}_{11}, \ldots, \hat{x}_{k-1,1}$ with the property that

$$< X, X >= \hat{x}_{01}^2 + 2[\hat{x}_{11}^2 + 2[\hat{x}_{21}^2 + \cdots]\cdots]] = \sum_{j=0}^{k-1} 2^j \hat{x}_{j1}^2 \,. \tag{6}$$

Hence the self join of a relation on an attribute a with N values can be calculated using the $k = \log(N+1)$ representative values computed. We call this algorithm *Self Join using Absolute Value (SJAV)*, and denote the sequence of k values obtained by this method from X by $SJAV(X)$.

Example 1 *Consider the vector X = (54.34, 79.7, 25.88, 97.13, 10.74, 37.52, 66.94) with $< X, X >= 25413.9$. The application of $SJAV$ on X is given in Figure 1. The resulting values $SJAV(X)$ that are stored are $\hat{x}_{01} = 140.7$, $\hat{x}_{11} = 49.33$, $\hat{x}_{21} = 13.69$. By using these representative values for*

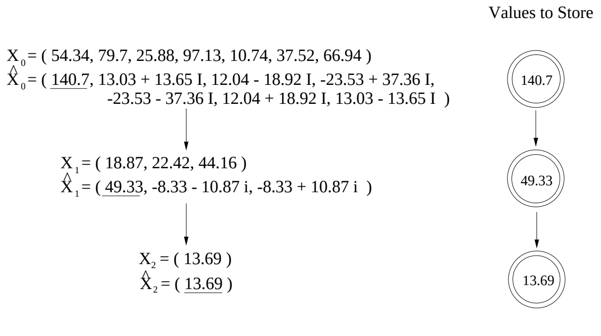


Figure 1: Example 1: The tree structure of $SJAV$.

X, and using (6), we calculate

$$\begin{aligned} < X, X > &= \hat{x}_{01}^2 + 2\hat{x}_{11}^2 + 2^2\hat{x}_{21}^2 \\ &= 140.7^2 + 2 * 49.33^2 + 4 * 13.69^2 = 25413.9 \end{aligned}$$

The self join algorithm using absolute values calculates the exact size of the self join. This is quite significant since all prior algorithms can only provide an approximation to the size of the resulting set. Consider again our hydrology example with ocean temperature. If we assume that the range of water temperature is say from -10^0C to 110^0C, then only 7 values need to be maintained by the query optimizer to calculate the exact self join sizes. If the domain of the self join attribute is 10^6, only 20 values need to be maintained by the query optimizer.

4.2 Approximating the Size of Arbitrary Joins

Unfortunately there is no direct equivalent to equation (4) as a way of representing the inner product of two different vectors. Suppose X and Y are two real vectors with DFTs $\hat{X} = \alpha + I\beta$ and $\hat{Y} = \gamma + I\delta$. Thus $\hat{x}_i = \alpha_i + I\beta_i$ and $\hat{y}_i = \gamma_i + I\delta_i$ for $i = 1, 2, \ldots, N$. Even though $< X, Y > = < \hat{X}, \hat{Y} >$ by Parseval's theorem, the expression (3) for the inner product can be expanded in the frequency domain as

$$< \hat{X}, \hat{Y} >= \sum_{i=1}^{N} \alpha_i \gamma_i + \sum_{i=1}^{N} \beta_i \delta_i,$$

which is not necessarily equal to

$$\sum_{i=1}^{N} |\hat{x}_i||\hat{y}_i| = \sum_{i=1}^{N} \sqrt{\alpha_i^2 + \beta_i^2}\sqrt{\gamma_i^2 + \delta_i^2}. \tag{7}$$

However by the Cauchy-Schwarz inequality the sum in (7) is an upper bound to the actual inner product $< \hat{X}, \hat{Y} >$ and therefore

$$\sum_{i=1}^{N} x_i y_i \leq \sum_{i=1}^{N} |\hat{x}_i||\hat{y}_i|.$$

At this point we will use the right hand side of the above inequality as an approximation to the left hand side and call this as *Approximation by Absolute Values (AAV)*. Now we can use the logarithmic representation obtained above by $SJAV$ as follows: Suppose $SJAV$ produces the sequences $SJAV(X) = \hat{x}_{01}, \hat{x}_{11}, \ldots, \hat{x}_{k-1,1}$ for X and $SJAV(Y) = \hat{y}_{01}, \hat{y}_{11}, \ldots, \hat{y}_{k-1,1}$ for Y. Then

$$\sum_{i=1}^{N} |\hat{x}_i||\hat{y}_i| = \sum_{j=0}^{k-1} 2^j \hat{x}_{j1}\hat{y}_{j1}.$$

This approximation to $< X, Y >$ using $SJAV$ is exact when $X = Y$. When Y is close in the N-dimensional space to a constant multiple of X, we expect the error in the approximation to be small as a consequence of the equality condition in the Cauchy-Schwarz inequality. As the two vectors become more distant in this sense, the error will increase. This crude approximation to $< X, Y >$ has some remarkable asymptotic properties which we mention in Section 6 where we present our experimental results. What we refer to as the *Approximation by Absolute Value (AAV)* algorithm is outlined in Figure 2.

5 Improving AAV: Tree Based Algorithms

AAV is a technique that gives us an upper bound for the inner product of two real vectors, which is exact for self join operations. Now we use AAV as a subroutine to develop more accurate approximation by means of a tree based

AAV Algorithm:

1. Calculate $SJAV(X)$ for all N-dimensional frequency vectors X by iterated DFT.
   Each $SJAV(X)$ is a sequence of $\log(N+1)$ real numbers.

2. Approximate $< X, Y >$ by

$$\sum_{j=0}^{k-1} 2^j \hat{x}_{j1} \hat{y}_{j1}$$

   where $SJAV(X) = \hat{x}_{01}, \hat{x}_{11}, \ldots, \hat{x}_{k-1,1}$ and $SJAV(Y) = \hat{y}_{01}, \hat{y}_{11}, \ldots, \hat{y}_{k-1,1}$.

Figure 2: Outline of the AAV algorithm.

structure. This results in a sequence of methods called *Tree Approximation Algorithms* (TAA_l), one for every l ranging from 0 to $k-1$. This approach exploits the form of Parseval's identity given in equation (3) for N-dimensional real vectors X and Y. The main idea is to keep the real and the imaginary parts of the transformed vector $\hat{X}$ *exactly* (i.e. without losing the phase information by taking the absolute value as in AAV). This results in two real vectors α and β, each N-dimensional where $\hat{X} = \alpha + I\beta$. Since $\hat{x}_i$ and $\hat{x}_{N-i+2}$ are conjugate complex numbers for $i = 2, 3, \ldots, (N+1)/2$, the numbers in the list $\alpha_2, \alpha_3, \ldots, \alpha_N$ appear in pairs. Similarly the numbers in $\beta_2, \beta_3, \ldots, \beta_N$ appear in pairs. Consider the vectors $h(\alpha) = (\alpha_2, \alpha_3, \ldots, \alpha_{(N+1)/2})$ and $h(\beta) = (\beta_2, \beta_3, \ldots, \beta_{(N+1)/2})$. We invoke two instances of the AAV algorithm: one on $h(\alpha)$ corresponding to the real part of $\hat{X}$ (left subtree), and one on $h(\beta)$ corresponding to the imaginary part of $\hat{X}$ (right subtree). Thus these two real vectors $h(\alpha)$ and $h(\beta)$ are treated as raw inputs to the AAV algorithm, producing two lists $AAV(h(\alpha))$ and $AAV(h(\beta))$ consisting of $k-1$ nonnegative real numbers each, where $N = 2^k - 1$. We demonstrate this process on an example.

Example 2 *Consider the vector X of the previous example, $N = 2^3 - 1 = 7$. The application of TAA_1 in which we now use AAV at level $l = 1$ results in the tree structure given in Figure 3.*

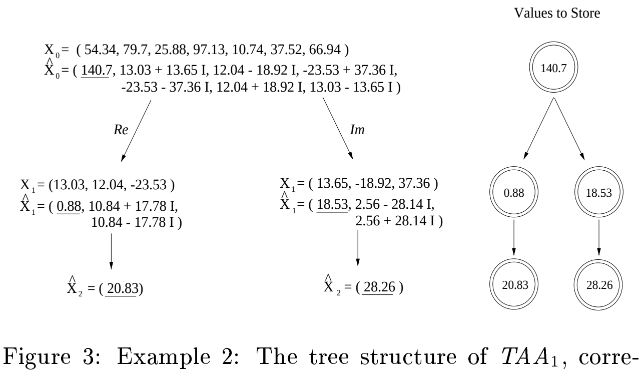


Figure 3: Example 2: The tree structure of TAA_1, corresponding to level $l = 1$.

For this example we need to store two lists $AAV(h(\alpha))$ and $AAV(h(\beta))$ of length 2 each, plus the number $\hat{x}_{01}$ that is computed at the root of the tree. The structure of the total list to be stored for X is itself a tree which is depicted on the right side of Figure 3. Since the calculation at the root of the tree in Figure 3 is exact and the application of the approximate AAV algorithm starts at level $l = 1$ of the tree, this algorithm is denoted by TAA_1. Note that AAV can be viewed as the special case TAA_0, starting the application at level 0. We see that this immediately generalizes to arbitrary level l, and results in an algorithm TAA_l for $l = 0, 1, \ldots, k-1$ in which the data is exact up to level l and the AAV algorithm is applied to the vectors at the next level. As a boundary case, we set $AAV(X) = X$ if X is a vector of length 1.

Example 3 *Continuing with the vector X of the previous examples, the application of TAA_2 in which we use AAV after level $l = 2$ gives the tree structure of Figure 4.*

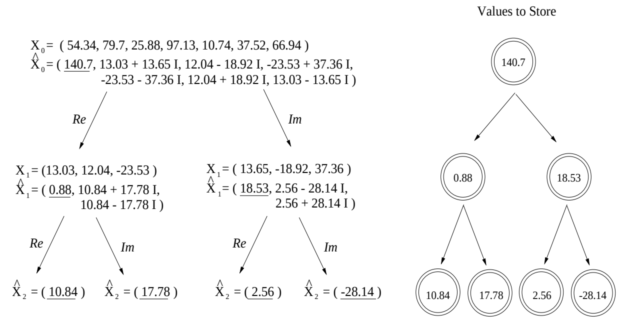


Figure 4: Example 3: The tree structure of TAA_2, corresponding to level $l = 2$.

The generation of the representative data in the form of a tree for a real vector X using algorithm TAA_l can be described recursively in Figure (5).

TAA data generation Algorithm

1. Suppose $N = 2^k - 1$, X is an N-dimensional real vector, and $\hat{X} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N) = \alpha + I\beta$.

2. If $l = 0$, then $TAA_l = AAV$, and the tree T of values computed as $AAV(X)$ is a linear list of length $\log(N+1)$, where N is the length of X.

3. If $l > 0$, then T has two principal subtrees T_{Re} and T_{Im}. The value at the root of T is $\hat{x}_1$. The subtrees T_{Re} and T_{Im} are constructed by applying algorithm TAA_{l-1} to the vectors $h(\alpha)$ and $h(\beta)$, respectively.

Figure 5: TAA representative tree generation algorithm.

5.1 Tree Approximation Algorithms and the TAA Product

In the above, we presented the TAA_l technique to extract a new set of representative data for a vector. In this process, by delaying the application of AAV i.e. increasing the value of l, we will be storing more exact data and hence the approximation based on this data will result in more accurate estimations. In this subsection we present the approximation algorithm that uses the representative data values found in the above process.

Definition 1 *Suppose T_1 and T_2 are two trees of height H and the same shape where the data field in each node stores a real numerical value. Let $Vec_l(T_1)$ denote the vector of values in the nodes at level l of T_1 ordered from left to right. Similarly, define the vector $Vec_l(T_2)$ for T_2. The TAA product of T_1 and T_2 is defined as*

$$TAA(T_1, T_2) = \sum_{l=0}^{H-1} 2^l < Vec_l(T_1), Vec_l(T_2) > \qquad (8)$$

In other words we first imagine T_1 and T_2 lined up so that corresponding nodes are on top of one another. We then multiply the numerical values in the paired nodes, further multiply this number by 2**(level of the node), and then add up the resulting numbers from each node.

Remark 1 *In terms of the TAA product, the approximation given by AAV to $< X, Y >$ is simply $TAA(T_X, T_Y)$ where T_X and T_Y are the trees (chains) of values $AAV(X)$ and $AAV(Y)$ respectively, produced by algorithm AAV.*

TAA algorithm at level l for the approximation of result sizes of join operations can be described succinctly in terms of the TAA product. This is shown in Figure (6).

TAA Algorithm for Level l:

1. Calculate the representative trees T_X for all N-dimensional frequency vectors X by the TAA_l data representation generation algorithm given in Figure 5.
2. Approximate $< X, Y >$ by the TAA product

$$TAA(T_X, T_Y) = \sum_{j=0}^{k-1} 2^j < Vec_j(T_X), Vec_j(T_Y) > .$$

Figure 6: Outline of the TAA algorithm.

Theorem 1 *Suppose X and Y are N-dimensional real vectors where $N = 2^k - 1$. Let T_X^l and T_Y^l denote the trees that are generated by TAA_l for the representation of X and Y respectively, $l = 0, 1, \ldots, k-1$. Then*

1. *Each approximation $TAA(T_X^l, T_Y^l)$ computed as given in (8) is an upper bound to the inner product $< X, Y >$.*
   Furthermore $TAA(T_X^0, T_Y^0) \geq TAA(T_X^1, T_Y^1) \geq \cdots \geq TAA(T_X^{k-1}, T_Y^{k-1})$, and $TAA(T_X^{k-1}, T_Y^{k-1}) =< X, Y >$ is exact.
2. *The number of elements stored in the representative tree T_X^l generated by TAA_l is $2^l(k - l + 1) - 1$.*

We give an example for the calculation of the approximation to $< X, Y >$ using the TAA product of the trees T_X and T_Y for $l = 2$.

Example 4 *Take X as in the previous examples. The tree T_X produced by algorithm TAA_2 for X is as given on the right hand side of Figure 4. Take Y=(69.97, 82.28, 49.67, 36.22, 29.81, 95.85, 51.74). The application of TAA_2 on Y produces the tree T_Y given in Figure 7. The approximation to the inner product $< X, Y >$ using the formulation in (8) on T_X and T_Y is calculated as follows:*

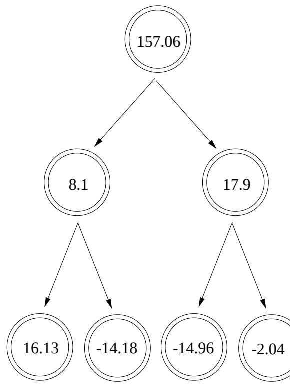


Figure 7: The tree T_Y: Application of TAA_2 to vector Y.

$l = 2$*:* $Vec_2(T_X) = (10.84, 17.78, 2.56, -28.14)$,
$Vec_2(T_Y) = (16.13, -14.18, -14.96, -2.04)$,
contribution to (8): $2^2 * (10.84 * 16.13 - 17.78 * 14.18 - 2.56 * 14.96 + 28.14 * 2.04)$

$l = 1$*:* $Vec_1(T_X) = (0.88, 18.53)$,
$Vec_1(T_Y) = (8.1, 17.9)$,
contribution to (8): $2^1 * (0.88 * 8.1 + 18.53 * 17.9)$

$l = 0$*:* $Vec_0(T_X) = (140.7)$, $Vec_0(T_Y) = (157.06)$,
contribution to (8): $2^0 * 140.7 * 157.06$

Summing the contributions from each level, we find $TAA(T_X, T_Y) = 22544$. Since we are using TAA_2 and $2 = k - 1$, this approximation is exact, i.e. $< X, Y >= 22544$.

Note that as indicated in part 2. of Theorem 1, the number of elements to be stored when the family of algorithms TAA is used varies from $k = \log(N + 1)$ for $l = 0$, to N for $l = k - 1$. Thus it is not surprising that the resulting size of a join operation can be exactly calculated using the full tree obtained for $l = k - 1$. This is because the number of elements stored as T_X^{k-1} for the representation of each vector in this case is N, and there are no savings in terms of space. Figure 8 shows the improvement in the approximation as the level l increases in the TAA approach. Fifty pairs of vectors X, Y of dimension $N = 2^{10} - 1$ were generated with uniformly random real entries between 0 and 1. TAA_l was run for levels $l = 0, 1, \ldots, 9$. The number of elements stored for the corresponding algorithms is given on the horizontal axis. Vertical axis is the average relative error of the corresponding approximation.

5.2 Tree Approximation Algorithms with Truncation

In the TAA algorithms we stored the first DFT coefficients of vectors at each level exactly up to level l, and subsequently used the AAV algorithm to get an approximation for the vectors from that level on. From Theorem 1 we know that AAV gives an upper bound for the inner product, and consequently the algorithms TAA_l are all upper bounds.

An alternative approach is to remove the application of AAV after level l in TAA_l, and approximate the result size by using only the first DFT coefficients of vectors in the tree up to level l. From this level on, instead of applying the AAV to the rest of the vectors, we approximate them by zero. In this way the resulting approximation is no longer an upper bound, as the nonnegative quantities contributed in the application of AAV down the tree are eliminated. The resulting family of algorithms indexed by level l is called *Tree*

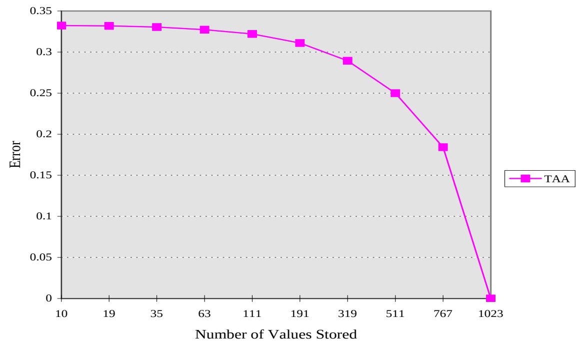


Figure 8: The approximation error of TAA_l as a function of level l.

Approximation Algorithms with Truncation (TAAT). To apply $TAAT_l$, we go down l levels in the tree and store the $2^{l+1}-1$ values of the partial tree generated. Thus $TAAT$ is in general more efficient in terms of storage than the TAA. The algorithm for using $TAAT_l$ to compute an approximation to $< X, Y >$ is similar to that of TAA. The approximation is given by the TAA product of the trees T_X^l and T_Y^l as defined in (8), but the summation is only up to level l instead of $k-1$, i.e. the approximation is given by

$$\sum_{j=0}^{l} 2^j < Vec_j(T_X), Vec_j(T_Y) > .$$

6 Performance Evaluation

In this section we experimentally evaluate the errors due to our approximation techniques. For the purpose of comparison, we compare our algorithms with the *end-biased v-optimal histograms (EB)* of Ioannidis and Poosala [7]. This approach is practical in terms of computation time and gives very good approximations [7, 13] (unlike the *serial* method, which gives even better approximations, but is computationally impractical for large vector sizes [7]).

The first set of experiments are designed for estimating a worse case upper bound on the size of a relation resulting from a join operation. This case is referred to as the *maximal result size* [5]. Given two frequency vectors X and Y, of the attributes involved in a join operation, the maximal result size can be obtained by sorting the two frequency vectors and assuming that the frequency values at the corresponding entries in the sorted vectors correspond to the same attribute values. AAV uses absolute values for approximating the size of a join operation. Since the absolute values represent an upper bound on the actual values, AAV is a good candidate for the maximal result case. The experiments were run for vectors of sizes 2^k-1 for $k = 5, 6, \dots, 12$, and hence the number of values stored per vector (or number of buckets in the standard terminology [7]) ranged from 5 to 12. In the figures, the x-axis refers to the number of values stored, while the y-axis refers to the relative errors as a percentage of the exact size. We used synthetic data from two different data distributions, namely *Random Data* (generated from a uniform distribution) and *Zipf Data*. For the former, we generated frequency vectors using a random number generator that gives uniformly distributed random numbers within the range [0,1]. For the latter, we used Zipf distributions with different z parameter values.

For random data we ran 50 experiments and took the average values–for AAV the variance for all values was negligible, and for EB the variance ranged from .0025 to 0. As shown in Figure 9(a) AAV performs much better than EB and its accuracy improves as the size of the vector increases. Since the error due to EB is always negative, the figures plot the graphs with the error sign. AAV, on the other hand, always gives a positive error since it gives an upper bound for the worse case. This is more appropriate for upper bound approximations. For the Zipf data, we generated frequency vectors by fixing z_1 for the first vector X to 1.0 and varied z_2 for the second vector Y from 0.0 to 2.0. Figure 9(b) shows that as before AAV always gives an upper bound on the error for the maximal result case. A common observation is that as predicted, whenever the data distribution of the two vectors are close to each other, i.e., z_1 is close to z_2, the error using AAV becomes smaller, and whenever the value of z_2 increases, the approximation is worse. EB performs best when the data distribution is quite skewed, i.e., for large values of z. In contrast, when z is in the range 0 to 1, AAV's performance is superior with relatively small errors.

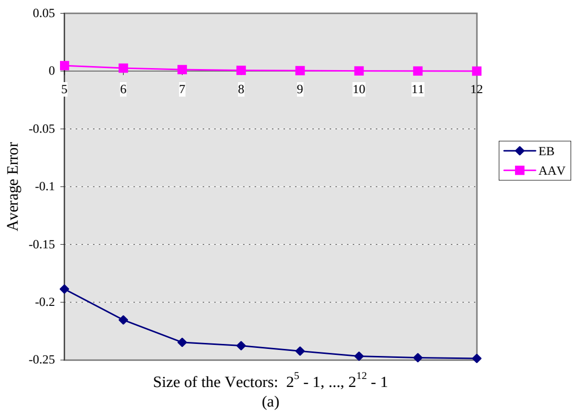
Finally, we compared the two techniques on multiple relation join queries ranging from 2 to 10 relations joined. For random data, we ran 50 experiments and show the average error; for Zipf data, we randomly selected z values for all vectors in the range [0,1]. The vector sizes were fixed to 2^8-1. Figure 10(a) shows the results of this experiment for Random data, and Figure 10(b) for Zipf data. Again, the approximation error due to AAV is significantly less than for EB for maximal result size approximation for multiple join operations.

Above we have seen that AAV performs well for the maximal result size case. In order to evaluate the performance of TAA we applied it to approximate the size of the relation resulting from a join operation for the average case. We ran experiments on random data with frequency vectors of length 2^k-1 for $k = 5, 6, \dots, 10$. Interestingly, we observe that when the $AAV(= TAA_0)$ algorithm is applied, the ratio of the approximation to the exact result is asymptotically $4/3$ as N tends to infinity[1]. Consequently for large N, we expect $3/4$ of the result obtained by AAV to be a better approximation on the average, although not necessarily an upper bound any longer. Figure 11 (a) compares the average error of this approximation to EB and shows that the errors are quite close, although EB results in less error. However, TAA is still an upper bound on the size of the join and hence can be considered a more reliable approximation, since it never underestimates the size of the join. This may be particularly important if these estimates are used for time or space allocation (an underestimate of size may cause an application to run out of memory).

In the case of $TAAT$ (Figure 11(b)), we ran 50 experiments on random data with frequency vector of length 2^10-1 and observed that the errors incurred by the two methods are very close and interleaved. This is in accordance with the expectation that since $TAAT$ ignores the nonnegative contribution of AAV, its approximation should be closer to the exact value.

The algorithms TAA_l are not suitable for unimodal distributions with sharp peaks. In the Zipf distribution for

[1] Let $[0,1]^N$ denote the unit cube in N-dimensional Euclidean space. This asymptotic behavior of AAV would follow from the convergence of the integral $\iint_{X,Y\in[0,1]^N} \frac{<AAV(X), AAV(Y)>}{<X,Y>} dXdY \rightarrow 1.333\ldots$



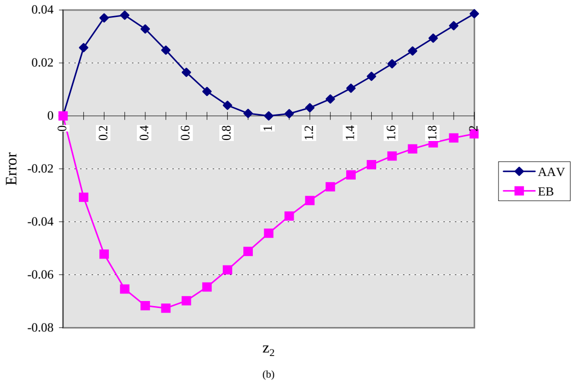
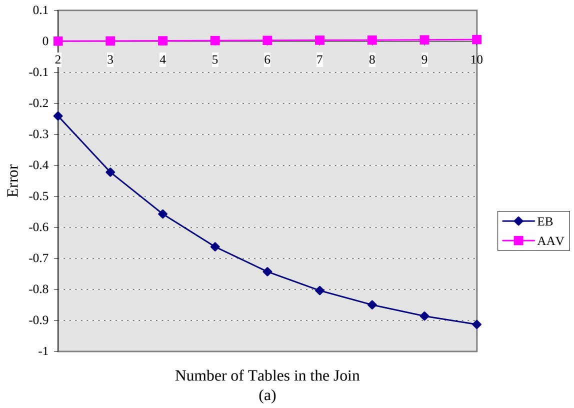


Figure 9: Average error for maximal result size. (a) Random Data. (b) Zipf Data



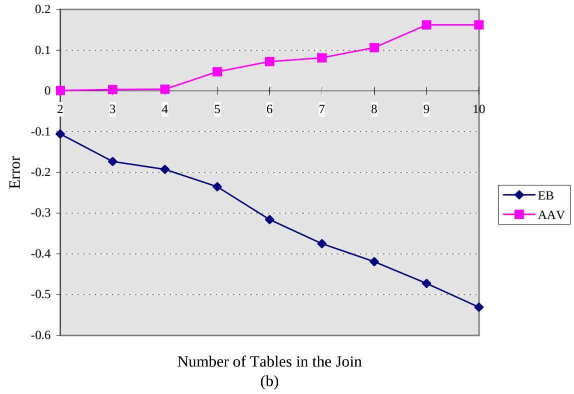
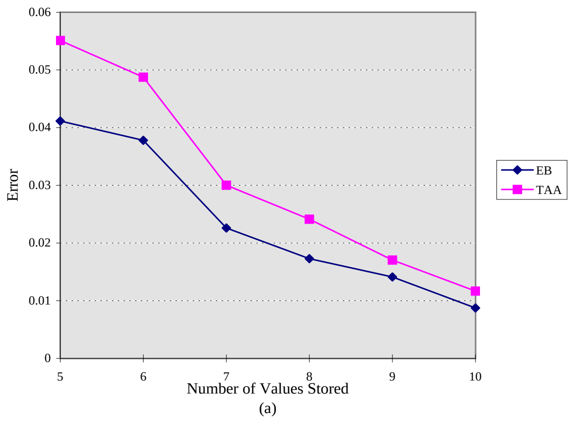


Figure 10: Multiple joins case. (a) Random data. (b) Zipf data.



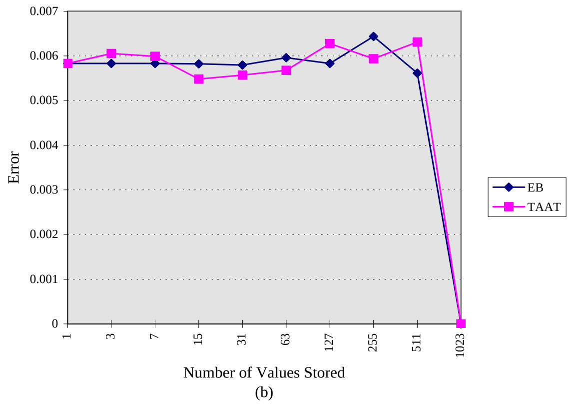


Figure 11: (a) Average error for TAA. (b) Average error for TAAT.

example, there are only a few values which are quite high relative to the rest. This behavior makes it especially suitable for EB since these large values are kept in separate buckets and the immaterial part averaged out. The DFT on the other hand is insensitive to the position of high values in the time domain.

7 Conclusion

We developed iterated DFT-based algorithms for estimating the size of relations resulting from join operations. DFT has previously been used in database systems for reducing the dimensionality of high-dimensional data, thus making it more suitable for current index structures. In this paper, we used several properties of DFT to reduce the size of the transformed frequency vector, while still providing good approximations for the size of join operations. The resulting algorithms present a spectrum of tradeoffs between storage requirements and accuracy. For the case of self join, we developed an exact algorithm, $SJAV$, for size calculation using only logarithmic space. This is particularly important for several scientific applications that require self join. For the more general case of joining different relations, we developed AAV. TAA is an iterative, tree-based algorithm where exact calculation is used up to a certain level, and then AAV is used for approximation. TAA interpolates in terms of accuracy and storage requirements between AAV at one extreme to exact calculation on the other, while space requirements increase from logarithmic to linear. Finally, $TAAT$ was developed by truncating the nonnegative contributions of AAV, and thus providing better size approximations. Our preliminary experimental results support our conclusions. In particular, both AAV and TAA provide fairly accurate maximal size estimates for a variety of dataset distributions. This is especially the case for dataset that are most highly skewed (Zipf with $0 \leq z \leq 1.5$). For the average case estimates, TAA provides a reasonable approximation while still guaranteeing that the estimate is an upper bound. $TAAT$, on the other hand, provides comparable approximations to prior algorithms for uniformly distributed data.

Finally, we should point out that our methods are particularly suitable for applications that require a mix of self join and arbitrary join operations. For self joins the calculations are exact and as our previous experiments demonstrates, provide good estimates for the general case.

References

[1] Agrawal R., Faloutsos C., Swami A.: Efficient Similarity Search In Sequence Databases. FODO, 1993.

[2] Christodoulakis S.: Implications of Certain Assumptions in Database Performance Evaluation ACM Transactions on Database Systems, 1984.

[3] Davis P. J.: Interpolation and Approximation, pg 192. Dover Publ. N.Y. 1963.

[4] Haas P. J. and Swami A. N.: Sampling-Based Selectivity Estimation for Joins Using Augmented Frequency Value Statistics. The International conference on Data Engineering, 1995.

[5] Ioannidis Y.: Universality of Serial Histograms. Proceedings of the 19th Conference on Very Large Databases, Dublin, 1993.

[6] Ioannidis Y., Christodoulakis S.: On the propagation of errors in the size of join results. Proc. of the 1991 ACM-SIGMOD Conf. Denver CO, May 1991.

[7] Ioannidis Y., Poosala V.: Balancing histogram Optimality and Practicality for Query Result Size Estimation. SIGMOD'95, San Jose, CA USA, June 1995

[8] Korn F., Jagadish H. V., Faloutsos C.: Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences. SIGMOD'97, AZ, USA, May 1997

[9] Lipton R. J., Naughton J. F., and Schneider D. A.: Practical Selectivity Estimation through Adaptive Sampling. Proceedings of ACM-SIGMOD, 1990.

[10] Mannino M. V., Chu P., Sager T. : Statistical Profile Estimation in Database Systems. ACM Computing Surveys, 20(3):192-221, Sept 1988.

[11] Muralikrishma M, and DeWitt D.: Equi-depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries. Proceedings of the ACM-SIGMOD Conference on Management of Data, 1988.

[12] Oppenheim A. V., Schafer R. W.: Discrete-time Signal Processing. Prentice Hall Signal Processing Series, 1989.

[13] Poosala V.: Histogram-based Estimation Techniques in Database Systems. Ph.D. Thesis, University of Wisconsin Madison, 1997.

[14] Rafiei D., Mendelzon A.: Similarity-Based Queries for Time Series Data. Proceedings of the ACM-SIGMOD Conference on Management of Data, 1997.

[15] Selinger P., Astrahan M., Chamberlin D., Lorie R., Price T.: Access Path Selection in a Relational Database Management System. Proceedings of the ACM-SIGMOD Conference on Management of Data, 1979.