Dynamic Nonuniform Data Approximation in Databases with Haar Wavelet

Su Chen
Dept. of Computer Science, Rutgers University, Piscataway, NJ, USA
suche@cs.rutgers.edu
Antonio Nucci
Narus Inc., Mountain View, CA, USA
anucci@narus.com

***Abstract*— Data synopsis is a lossy compressed representation of data stored into databases that helps the query optimizer to speed up the query process, e.g. time to retrieve the data from the database. An efficient data synopsis must provide accurate information about the distribution of data to the query optimizer at any point in time. Due to the fact that some data will be queried more often than others, a good data synopsis should consider the use of nonuniform accuracy, e.g. provide better approximation of data that are queried the most. Although, the generation of data synopsis is a critical step to achieve a good approximation of the initial data representation, data synopsis must be updated over time when dealing with time varying data. In this paper, we introduce new Haar wavelet synopses for nonuniform accuracy and time-varying data that can be generated in linear time and space, and updated in sublinear time. We further introduce two linear algorithms, called *2-Step* and *M-Step* for the *Point-wise approximation* problem that clearly outperforms previous algorithms known in literature, and two new algorithm called, *Data Mapping* and *Weight Mapping* for the *Range-sum approximation* problem that, to the best of our knowledge, represent a key research milestone as being the first linear algorithm for arbitrary weights. For both scenarios, we focus not only on the generation of the data synopsis but also on their updates over time. The efficiency of our new data synopses is validated against other linear methods by using both synthetic and real data sets.**

***Index Terms*— Dynamic data synopsis, query optimization, nonuniform lossy compression, point-wise and range-sum approximation, linear complexity**

I. Introduction

How to efficiently query data is a fundamental problem in databases. Estimating the cost of complex queries, reflecting CPU time, memory usage and I/O operations, requires a detailed knowledge of how data are distributed and stored into database tables. In practice, any database system maintains a concise lossy compressed data structure, called *data synopsis*, that approximates the data distribution at any point in time. The query optimizer will use this data summary to decide how a query is executed in order to retrieve the requested data at the minimal cost in terms of overall processing overhead. Besides accuracy, the cost of generating and updating synopses is a major concern, since the cost of optimization may overwhelm its benefits if the cost of synopsis is too large.

According to the characteristics of the environment at which data synopses are applied, data synopses might be classified as (i) *Uniform* vs. *Nonuniform* and (ii) *Static* vs. *Dynamic*.

Data synopses applied to data that require the same quality of approximation are known as "uniform". All data subset will be represented by the same *weight*, that reflects the fact that any data subset must be treated in the exact same way as the others. Scenarios in which at least one data subset is required to be known with a better quality of approximation than others, e.g. for example because it might be invoked in the query process more often than the others, are known as "nonuniform". In this case, the weight associated to each data subset will be different; the larger is the value of its weight the higher is the quality of its approximation for the data subset.

At the same time, data synopses can be defined for "static" or "dynamic" data structures. When data and weights do not change over time, it is said to be "static". In this context, it is important to generate a good data synopsis for the data on hands. When data or weights change over time, the scenario is said to be "dynamic". Dynamic scenarios are challenging to be managed. In this case, data synopses should be well generated and updated over time in order to provide approximated answers with high accuracy and thus provide meaningful information to the query optimizer at any point in time.

There are two types of data synopses that can be used to answer database queries: *Point-wise approximation* and *Range-sum approximation*. The Point-wise approximation is defined for single data point query while the Range-sum approximation is defined for data intervals query, e.g. set of data points. As a consequence, Point-wise approximation can be seen as a special case of Range-sum approximation when the data interval collapses into one single data point.

In this paper, we introduce new wavelet data synopses that can be generated in linear time and updated in sublinear time for dynamic-data and nonuniform weights. Our major contributions are summarized in the following. **First**, we propose two linear algorithms for the Point-wise approximation problem, called *2-step* and *M-step*. Results

This paper is based on "Nonuniform Compression in Databases with Haar Wavelet", by S. Chen and A. Nucci, which appeared in the Proceedings of IEEE Data Compression Conference(DCC), pp. 223-232, Snowbird, USA, March 2007 © 2007 IEEE.

clearly show how the proposed algorithms outperform weighted wavelet algorithm [1] on approximation error on both synthetic and real data sets with only $O(B^3)$ extra running time, where B represents the approximation space. **Second**, we propose two linear algorithms for the Range-sum approximation problem called *Data-mapping* and *Weight-mapping*. To the best of our knowledge, these algorithms represent the first linear algorithms ever proposed in literature for arbitrary weights. For n weights and B compression space, we show that both time and space complexity is $O(n^2 + B^3)$, which is linear to the weight size $O(n^2)$. **Third**, all the algorithms we proposed in this paper are the first ones that focus not only on the generation of the data synopsis but also on its updates overtime. For dynamic data and weights, the time complexity of synopsis updates is $O(\log(n)+B^3)$ for point-wise approximation and $O(n + B^3)$ for range-sum approximation. **Fourth**, we show that when the structure of given weights can be simplified, our algorithm can be tuned accordingly, and the complexity for range-sum synopses generation can be reduced from $O(n^2 + B^3)$ to $O(n + B^3)$.

The remainder of this paper is structured as following. Section II provides the mathematical definition of Point-wise and Range-sum approximations and introduces the related work. In Sections III and IV we present two new algorithms for generating Point-wise approximation, e.g. *2-step* and *M-step*, and two new algorithms for generating Range-sum approximation, e.g. *Data-mapping* and *Weight-mapping*. In section V, we introduce the novel incremental method that can minimize the computational cost when experiencing dynamic data and weights. In section VI we introduce three examples where our algorithm can simplify the inner states, which is used to accelerate the synopses generation, if the weights have patterns inside. As a result, the overall cost can be reduced to sublinear in term of the input weight size. In Section VII we validate the performance achieved by these algorithms using both synthetic and real-data, while Section VIII concludes the paper.

II. Problem Statement and Previous work

Small space synopses in database systems have been studied over decades to improve both the accuracy and the efficiency of the query approximation and query optimization. Traditional data synopses only represent the data that reside in the database, but they lack of considering the characteristic of the data being queried. As a consequence, data queried more often than others are treated exactly in the same way and thus the overall quality of the approximations produced ends to be poor.

Query Feedback systems [2]–[7] have been proposed to address this problem. In these systems, data approximations are corrected by their real values returned from queries, so that the frequently visited data can be more accurate than the others. An example of such feedback mechanism is represented by systems like LEO [7]. However query workload information is not fully explored in these systems, since the data accuracy only reflect how recently the data is queried, but not how often the data is queried.

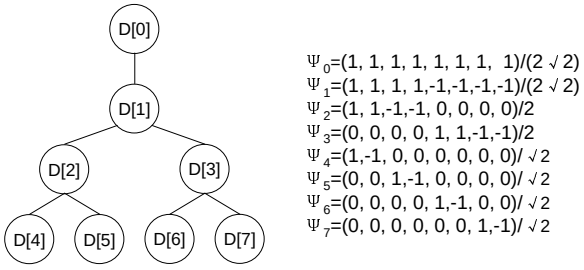


Figure 1. Wavelet transform of signal A

Only recently, real weights that indicate the frequency of data being queried are extracted from workloads and used to generate "usage-oriented" synopses [1], [8], [9]. In this paper, we use weights from workloads instead of query feedback mechanism to quantify the accuracy of our data synopses.

Considerable work is available in literature for the Point-wise data synopses generation when using histogram data synopses [2], [3], [8], [10]–[12]. However, most recently, several papers [1], [13]–[17] remark the effectiveness of using wavelet decomposition in reducing large amount of data to compact sets of wavelet coefficients, termed *wavelet synopses*. Wavelet synopses have been proved to provide fast and reasonably accurate approximate answers to queries. Among wavelet synopses, the Haar wavelet synopsis has been found to be the most interesting one to be used in database due to its simple structure.

In this paper we focus on Haar wavelet synopsis applied to dynamic data and nonuniform weights for both Point-wise and Range-sum approximations. In figure 1, we show an example of wavelet transform of a signal A with length 8. The coefficients D are the *inner product* of the signal A, represented as $\otimes$, and the wavelet vectors $\Psi = \{\Psi_1, ..., \Psi_n\}$ at different level, e.g., $D[i] = A \otimes \Psi_i$.

In the following we provide the mathematical definition of the two problems.

Problem 1 (Point-wise Approximation): Let $A_{1\times n}$ be a generic data vector of dimension n and $\Pi_{1\times n}$ be the weight vector of dimension n that reflects the approximation quality of each data point of A, e.g. to each $A[i]$ is associated a weight $\Pi[i]$. The weight vector is normalized between $[0, 1]$, e.g. $\sum_{i=1}^{n} \Pi[i] = 1$. Let Ψ be the set of n candidate wavelets and B be the allowed number of wavelets to be used for the data synopsis. Let D be the set of coefficients associated to the B chosen wavelets, e.g. $D[i]$ is associated to Ψ_i.

The Point-wise approximation problem can be stated as to identify the optimal subset of B wavelets from Ψ and their associated coefficients D, in order to minimize the weighted Point-wise approximation error defined as:

$$\epsilon^{(P)}(A) = \sum_{i=1}^{n} \Pi[i](A[i] - \widehat{A}[i])^2 \qquad (1)$$

where $\widehat{A}$ represents the wavelet approximation of data set A, e.g. $\widehat{A} = \sum_{i=1}^{B} D[i]\Psi_i$

Problem 2 (Range-sum Approximation): Let $A_{1\times n}$ be a generic data vector of dimension n, and let $A(i,j) = \sum_{k=i}^{j} A[k]$, with $i \leq j$, represent an additive function that operates on all elements of data vector A from $A[i]$ to $A[j]$. Let $\Pi_{n\times n}$ be the weight matrix of dimension $n \times n$ such that $\Pi[i,j] = 0$, $\forall i \geq j$. Each element $\Pi[i,j]$ represents the weight associated to A(i,j). The weight matrix Π is normalized between $[0,1]$, e.g. $\sum_{i=1}^{n}\sum_{j=1}^{n} \Pi[i,j] = 1$.

The Range-sum approximation problem can be stated as to identify the optimal subset of B wavelets from Ψ and their associated coefficients D, in order to minimize the weighted Range-sum approximation error defined as:

$$\epsilon^{(R)}(A) = \sum_{i=1}^{n}\sum_{j=1}^{n} \Pi[i,j](A(i,j) - \widehat{A}(i,j))^2 \qquad (2)$$

where $\widehat{A}(i,j)$ represents a generic linear function of the wavelet approximation of $A(i,j)$, e.g. $\widehat{A}(i,j) = f(\sum_{i=1}^{B} D[i]\Psi_i)$.

Most of the wavelet synopses are generated under the assumption of uniform weights for both Point-wise and Range-sum approximations [13] [14] [15]. For Point-wise approximation, the Parseval's theorem provides a solution that applies to all orthonormal data transforms, i.e., the best approximation is achieved by largest coefficients. For range-sum approximation, [15] presents an optimal solution on Haar wavelet synopsis.

The methods that study the more general case of nonuniform weights scenario result to be either sub-optimal in terms of approximation errors [1] or too expensive in terms of running time [18]. Matias and Urieli in [1] provided the first linear algorithm, called ***Weighted-wavelet*** (W-wav), able to preserve Parseval's orthonormal condition by using a smart combination of wavelets and weights. As a result, their method provides the best synopses on weighted Haar bases, when choosing the largest coefficients as synopses. Unfortunately, this approach provides approximation errors that do not decrease monotonically with respect to the compression space B and the outcome approximation error may not be bounded. In Figure 2 we show with an example of what just stated. Data A is extracted from an exponential distribution. We define approximation error with $B = 0$ as $\epsilon_0 = \sum_i \Pi[i]A[i]^2$, i.e. assume all data values are 0. The approximation error of W-wav with $B = 0$ is $\epsilon_0 = 623.49$. With more compression space $B = 2$, the error increases to 686.55. Their approximation is far from the **ideal approximation** (see Section III), and worse than our ***2-step*** *method* (see Section III) that reduces the error to 306.95 with $B = 2$. This problem exists not only for exponential data sets, but for all data sets that are characterized by only a few very large values.

Guha and Harb in [18] studied this problem for different L_p norm errors. They proved that the best coefficients can be found by searching a bounded region that is specified by the optimal error of L_∞. Because the real value of this optimal error is unknown, the algorithm needs to guess a value to run. The accuracy of the solution is strictly related to the cardinality of the search space. The larger the search space is, the more accurate the result will be at the cost of a higher running time. The time complexity reaches $O(n^3)$ for L_2 error, and the space complexity is super linear to n. As a consequence, the complexity of their approach results to be too high for synopses used for databases.

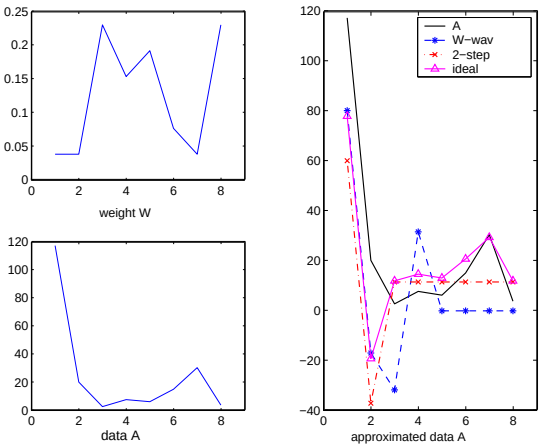


Figure 2. W-wav algorithm is not optimal

In [9], the author studied a special case of this problem where the weights are assumed to be organized into k intervals, and a unique weight value is associated to each interval. The running time is $O(nkB^2\log(n))$. When $k = n$, this case collapses to the Point-wise approximation problem, with a quadratic running time $O(n^2B^2)$ in terms of n.

Conversely, data synopses generation for the more general case of range-sum approximation has received only little attention from the research community. Most of the work in literature has studied this problem in the context of uniform weights or hierarchical weights [10], [11], [19] ignoring completely the importance of the non-uniform workload scenario. In this paper, we extensively study this problem as well and propose two linear algorithms to generate range-sum synopses in the context of arbitrary weights.

We conclude this section by highlighting the fact that all previous work, for both point-wise and range-sum approximation has been focused only on the generation of the data synopses, but ignored the importance of a more realistic scenario in which both data and weights can change over time and thus the mechanism on how to update the data synopsis over time. We consider this case as well in the paper.

III. Nonuniform Point-wise Approximation Problem

In order to approach this problem, let us define three variables that come to play into the problem (see Figure 3): (i) the data vector A, (ii) the wavelet vectors Ψ and (iii) the weight vector Π. For uniform weights, the data vector A is mapped to the wavelet vector Ψ in

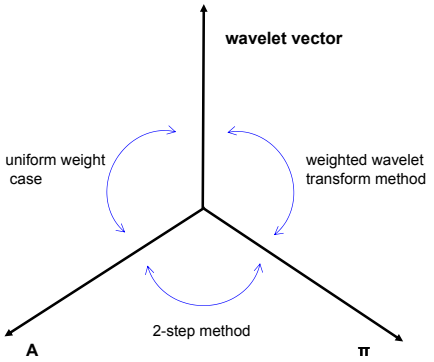


Figure 3. Methods

order to generate data synopses. For nonuniform weights, the mapping is arbitrary. The *W-wav* algorithm combines the weight vector with the wavelet vector by stretching wavelets vertically. The new wavelet basis is thus weight-specific. Data synopsis obtained when considering a specific weight vector might be sub-optimal for a different weight vector. As a result, the wavelet basis has to be recomputed every time a weight change is experienced, leading to large cost in database management. A different approach to solve this problem is to combine the weight vector with the data vector. The intuition behind this approach comes from a good understanding of the error function.

Given the data vector A and the weight vector Π, the error function (Equation 1) can be formulated as:

$$\begin{aligned}\epsilon^{(P)}(A) &= \sum_{i=1}^{n}(\sqrt{\Pi[i]}A[i] - \sqrt{\Pi[i]}\widehat{A}[i])^2 \\ &= \sum_{i=1}^{n}(A_W[i] - \widehat{A}_W[i])^2 \qquad (3)\end{aligned}$$

where $\widehat{A}$ represents the approximation of A, while $A_W[i] = \sqrt{\Pi[i]}A[i]$.

With this simple manipulation, the error function has been reduced to a uniform-weight case. As a result, the best approximation $\widehat{A}^*$ of A can be solved from $\widehat{A}^*[i] = \widehat{A}^*_W[i]/\sqrt{\Pi[i]}$, where the optimal approximation $\widehat{A}^*_W$ of A_W exists according to Parseval's theorem. We refer to this method as ***ideal approximation***. Unfortunately this method is impractical because it requires the knowledge of the weight value $\Pi[i]$ for each point, and thus inadmissible because the compression space B is not large enough to store Π. One might consider to introduce an approximation of Π but this will lead to a strong sub-optimality.

In this paper we propose two algorithms, called *2-step* and *M-step* that are based on the intuition to first select wavelets according to the optimal error ϵ^*, and then optimize their coefficients on $\widehat{A}$.

A. 2-step algorithm

The name of this algorithm comes from its 2-step mechanism adopted to derive the data synopsis (figure 4).

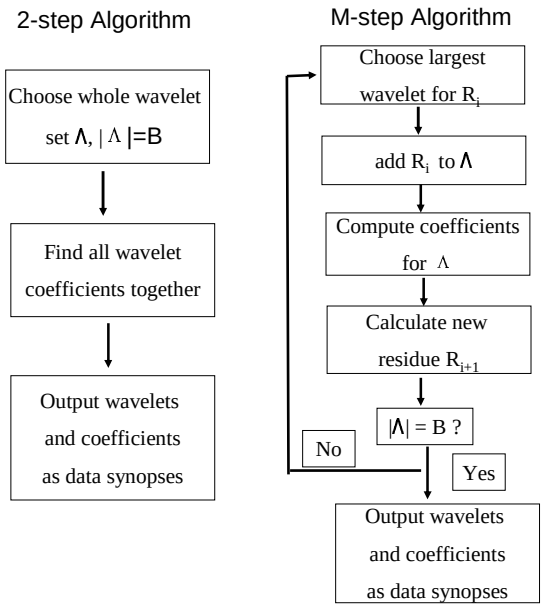


Figure 4. Flow charts of *2-step* and *M-step* algorithms

In *Step A*, we define the weighted data A_W to be a point-wise product of A and $\sqrt{\Pi}$, e.g. $A_W[i] = A[i]\sqrt{\Pi[i]}$. The algorithm selects a set of wavelets with the largest B coefficients from the wavelet transform of A_W.
Step B computes the best coefficients $D[i]$ for the chosen wavelets by solving the partial differential equation of the function described in Equation (1), e.g. $\frac{\partial\epsilon^{(P)}}{\partial D[k]} = 0$, $\forall k \in [1, B]$.

The nice characteristics of the proposed algorithm is related to the fact that its approximation error is bounded by $|\widehat{A}[i] - \widehat{A}^*[i]| = |\widehat{A}_W[i](1 - \frac{1}{\sqrt{\Pi[i]}})|$ after the first iteration, and it will be reduced significantly at the second iteration.

The time and space complexity in Step A is $O(n)$ for generating A_W from A and W, as well as computing its wavelet transform, which is linear to the data size. Thus whether the Step B can be computed in linear time is critical to keep the overall cost low. The following lemma describes why Step B requires only linear time [9].

Lemma 1: For any given wavelet set with size B, the best coefficient set D can be find in $O(n + B^3)$ time.

Proof Let the chosen wavelet set be $\Lambda = \{\Psi_1, ..., \Psi_B\}$. The approximated data R, defined as the product of the chosen wavelets and their coefficients, can be computed as $R = \sum_{i\in\Lambda} D[i]\Psi_i$.

As a consequence, the approximation error ϵ, defined as the L_2 distance between original data and approximated data, can be written as:

$$\begin{aligned}\epsilon &= \sum_i \Pi[i](A[i] - R[i])^2 \\ &= \Pi \otimes ((D[1]\Psi_1 + ... + D[B]\Psi_B - A) \\ &\quad \odot (D[1]\Psi_1 + ... + D[B]\Psi_B - A))\end{aligned}$$

while $\otimes$ is the inner product [1] and $\odot$ is the point-wise product of two vectors [2].

To find the best $D[i]$ that minimizes the error, we solve the following set of equations:

$$\begin{cases} \frac{\partial \epsilon}{\partial D[1]} = 0 \\ ..... \\ \frac{\partial \epsilon}{\partial D[B]} = 0 \end{cases} \Leftrightarrow PD = Q$$

in which, $\frac{\partial \epsilon}{\partial D[i]} = 2\Pi \otimes (\Psi_i \odot (D[1]\Psi_1 + ... + D[B]\Psi_B - A)) = 0$. Notice that, the set of equations above can be written in matrix notation $PD = Q$ where $P[i,j] = \Pi \otimes (\Psi_i \odot \Psi_j)$ and $Q[i] = \Pi \otimes (\Psi_i \odot A)$.

Since the time for solving $PD = Q$ is at most $O(B^3)$, the dominate part of the time complexity ends to be the generation of the matrices P and Q. In the following, we propose an efficient approach to generate those matrices in $O(n)$ based on the intuition that the computation of Ψ_i and Ψ_j collapses into three simple and standard cases, as shown in the following:

case 1: If Ψ_i and Ψ_j do not overlap,

$\Psi_i \odot \Psi_j = 0$

case 2: If Ψ_i and Ψ_j are the same, i.e., $i = j$,

$\Psi_i \odot \Psi_i = \{\frac{1}{l}, ..., \frac{1}{l}\}$, where l is the length of Ψ_i's **Support Interval**, which is the non-zero parts of Ψ_i.

case 3: If Ψ_i covers Ψ_j,

$\Psi_i \odot \Psi_j = \pm c_i \Psi_j$, c_i is a constant that normalize Ψ_i, named **Normalization Factor**, e.g., in figure 1, $c_7 = \sqrt{2}$.

As a consequence, $P[i,j]$ is either 0 (case 1), or the average value of Π in Ψ_i's in the non-zero interval (case 2), or the wavelet transform coefficient of Π scaled by $+c_i$ or $-c_i$, where the sign depends on the relative position of Ψ_j and Ψ_i (case 3). For both case 2 and case 3, $P[i,j]$ can be computed directly from the wavelet transform of Π. Since we can compute all values for $P[i,j]$ directly from one wavelet transform of Π, we need only $O(n)$ time to generate the entire matrix P.

Same observation hold true for the computation of the matrix Q. Indeed, all $Q[i]$ values can be generated in $O(n)$ as $Q[i] = \Pi \otimes (\Psi_i \odot A) = (\Pi \odot A) \otimes \Psi_i$, which is the coefficient of wavelet transform of $\Pi \odot A$.

In the following, we prove that the $PD = Q$ is indeed solvable. First, let's consider the case that not all coefficients of Π are 0 in any of the chosen wavelet Ψ_i's support interval, $i = 1..B$.

Then matrix P can be decomposed as the product of the matrix T and its transpose.

$$P = TT' = \begin{pmatrix} \sqrt{\Pi} \odot \Psi_1 \\ ... \\ \sqrt{\Pi} \odot \Psi_B \end{pmatrix} \left(\sqrt{\Pi'} \odot \Psi'_1, ..., \sqrt{\Pi'} \odot \Psi'_B \right)$$

Since $\Psi_1, ..., \Psi_B$ are linear independent, and $\sqrt{\Pi} \odot \Psi_i \neq \vec{0}$,[3] the $rank(T) = rank(T') = B$. This means that $rank(P) = B$ and thus the P matrix is non-singular. Therefore $PD = Q$ is solvable.

Second, let's consider a more general case where all Π are 0 for the entire Ψ_i's support interval. To prove this case, we set all the $D[i] = 0$, and thus reduce the matrix P to be equal to $(B-k) \times (B-k)$, where k is the number of such Ψ_is. Therefore, this equation is solvable.

Now let's consider the computational time. In the worst case, we have B equations with B variables, and thus we can solve all $D[i]$ in $O(B^3)$ time. By adding the time required to generate the matrices P and Q and solve the system $PD = Q$, the total time complexity to compute the coefficients for a given wavelets is equal to $O(n+B^3)$. □

In summary, the time complexity of the *2-step* algorithm is $O(n)$ for the computation of the wavelet transform and $O(n + B^3)$ for the computation of the best coefficients. The space complexity is $O(n)$ for wavelet transform, and $O(B^2)$ for the coefficients selection. We want to remark to the reader that the $O(n)$ space can be reused in the second step since the algorithm requires to keep the indexes of the B chosen wavelets only. As a consequence, the total time required for execution is expressed by $O(n + B^3)$ while the space required is $O(max\{n, B^2\})$. Furthermore, its complexity becomes linear to the data size n when $B \ll n$, e.g. typical of standard database applications.

B. M-step Algorithm

The *M-step* algorithm represents an improvement of the *2-step* algorithm based on the observation that the error obtained by the *2-step* algorithm can be further reduced down by selecting new wavelets at each iterations (figure 4). The selection of the new wavelets is carried out in order to minimize the difference between the approximated data and the original data, termed ***residue*** and denoted as R_i where i represents the ith-iteration. The algorithm starts by setting the initial residue $R_0 = A_W$, and the chosen wavelet set to be the empty set. At each iteration i, the algorithm computes the wavelet transform for the current residue R_i, chooses the wavelet Ψ_k with the largest coefficient and that does not belong to the chosen set. The new wavelet Ψ_k is added to the chosen set. The algorithm computes the new coefficient vector D for all chosen wavelets by solving $\frac{\partial \epsilon^{(P)}}{\partial D[k]} = 0$. At this point, the new residue R_{i+1} can be computed as $R_{i+1} = A_W - \sum_{k=1}^{i} D[k]\Psi_k$, and the algorithm is ready to enter into the next iteration $i + 1$. The $M - step$ algorithm continues until the cardinality of chosen wavelet set is equal to B.

At each step of this algorithm, every data point of the residue can change due to the new added wavelet. As a result, the algorithm needs to recompute the wavelet transform for every R_i. After B steps, the running time

[1] $z = X_{1\times n} \otimes Y_{1\times n} \Leftrightarrow z = \sum_{i=1}^{n} X[i]Y[i]$

[2] $Z_{1\times n} = X_{1\times n} \odot Y_{1\times n} \Leftrightarrow Z[i] = X[i]Y[i]$

[3] This statement is based on the assumption that at least one of $\Pi[j] \neq 0$ if $j \in \Psi_i$'s support interval

adds up to $O(nB+B^4)$. The storage space is $O(\max\{n+B, B^2\})$, because the algorithm needs to remember up to B chosen coefficients.

A variation of the M-step algorithm is to choose I wavelets together at each step. This reduces the running time to $O(nB/I + B^4/I)$ at the cost of a degradation in accuracy. When $I = B$, this algorithm collapses to the *2-step* algorithm.

A general comment about the two algorithms proposed in this section is related to their property of having the approximation error decreasing monotonously. If a "bad" wavelet is chosen at any iteration, the next step can always ignore its negative effects by setting its coefficient to 0. In Section VII we show how the two algorithms are able to efficiently reduce their estimation errors compared to *W-wav* algorithm.

IV. Nonuniform Range-sums approximation problem

The same idea presented in the previous section for the Point-wise approximation, can be easily generalized in the case of the Range-sum queries. For a data set A with length n, there are $\frac{n(n+1)}{2}$ number of intervals and weights. A **Naive** method to solve the Range-sum approximation problem is to generate a new data set $A^{(new)}$, in which every data point $A(i,j)$ represents an interval extracted from the original data A and computed as $A^{(new)}(i,j) = \sum_{k=i}^{j} A[k]$. As a consequence, it is straightforward to write the error function of the new data $A^{(New)}$ as $\epsilon^{(P)}(A^{(New)}) = \sum_{i,j} \Pi[i,j](A^{(New)} - \widehat{A^{(New)}})^2$, and thus find the wavelet synopsis using methods in point-wise approximation case.

Although the idea is simple, the complexity of this approach is high because the length of $A^{(New)}$ is $O(n^2)$ and thus largely exceeds the synopses space B.

Because the complexity is determined by the number of intervals constituting the original data A and the weights associated to each interval, in this Section we propose two new algorithms, named *Data-mapping* and *Weight-mapping*, that generate new data vectors of size n. To the best of our knowledge, these algorithms are the first linear algorithms for arbitrary weights proposed in literature as now.

A. Data-mapping algorithm

The ***Data-mapping*** algorithm transforms the original Range-sum approximation problem into a simpler Point-wise approximation problem by introducing a simple data and weight transformation. Given the original data A, we define a new data $A^{(DM)}$ obtained as the partial sum of A, e.g. $A^{(DM)}[i] = \sum_{k=1}^{i} A[k]$ with $A^{(DM)}[0] = 0$.

By using the new data $A^{(DM)}$, we can re-write $A(i,j)$ as the difference between $A^{(DM)}[j]$ and $A^{(DM)}[i-1]$, i.e., $A(i,j) = A^{(DM)}[j] - A^{(DM)}[i-1]$. As a consequence, equation (2) can be looked as a function of new data $A^{(DM)}$.

$$\begin{aligned}\epsilon^R &= \sum_{i,j} \Pi_{i,j}[(A^{(DM)}[j] - A^{(DM)}[i-1]) \\ &\quad -(\widehat{A^{(DM)}[j]} - \widehat{A^{(DM)}[i-1]}]^2\end{aligned}$$

Thus we can obtain the expression of the new weights associated to the new data vector $A^{(DM)}$ as follows:

$$\Pi^{(DM)}[i] = \sum_{k=1}^{i} \sqrt{\Pi[k,i]} + \sum_{k=i+1}^{n} \sqrt{\Pi[i+1,k]} \qquad (4)$$

In order to minimize the error function and thus obtain the best wavelet coefficients D, we calculate the derivative equation of the error function regard to D and set it to 0, e.g. $\frac{\partial \epsilon^{(R)}(A^{(DM)})}{\partial D} = 0$. As a consequence, the problem ends into solving a set of B linear equations of the form for the general wavelet coefficient vector $D[k]$ (Equation 5).

If the B linear equations are solved one by one, the complexity of the procedure ends up to be $O(n^2B^2+B^3)$. In order to reduce the time complexity of this step, we propose to organize the Equations (5) in matrix notation of the form $PD = Q$, where $P[k,l] = \sum_{i,j} \Pi[i,j](\Psi_k[j] - \Psi_k[i])(\Psi_l[j] - \Psi_l[i])$ and $Q[k] = \sum_{i,j} \Pi[i,j](A^{(DM)}[j] - A^{(DM)}[i-1])(\Psi_k[j] - \Psi_k[i-1])$.

Then we construct a prefix sum table for $\Pi[i,j]$ that helps to compute the weights for any interval (i,j) in constant time, so that the overall complexity can be reduced to $O(n^2 + B^3)$. In the following paragraphs, we introduce the method that can reduce the cost from polynomial to linear.

We start with the evaluation of the cost to generate the matrices P and Q.

First, the matrix P can be computed in $O(B^2)$ due to two major observations:

(i) There are only a constant number of intervals, i.e., less than 25 intervals, in which $P[k,l] \neq 0$, since there are only 5 non-zero intervals for $\Psi_k[j] - \Psi_k[i-1]$ and $\Psi_l[j] - \Psi_l[i-1]$ (figure 5).

(ii) In these intervals, $P[k,l]$ can be computed in constant time from $\sum_{i\in I, j\in J} \Pi_{i,j}$, since normalization factor c_k and c_l are constants specified by only Ψ_k and Ψ_l.

(iii) The computation of $\sum_{i\in I, j\in J} \Pi_{i,j}$ can be carried out in constant time with the help of a prefix sum table. The prefix sum table for $\Pi_{i,j}$ can be easily generated by adding every row of the table $\Pi_{i,j}$ to its next to generate $\sum_{i\in[1,k]} \Pi_{i,j}$ for each j, then adding every column to its next right column to generate $\sum_{i\in[1,k], j\in[1,l]} \Pi_{i,j}$ (see Figure 6 for more details). At the end of this process, the data is very well organized so that the computation of the $\sum_{i\in I, j\in J} \Pi_{i,j}$ for any interval $I = [i1, i2]$ and $J = [j1, j2]$ needs only 3 operations: $\Pi_{[j1,j2]}^{[i1,i2]} = (\Pi_{[1,j2]}^{[1,i2]} - \Pi_{[1,j1-1]}^{[1,i2]}) - (\Pi_{[1,j2]}^{[1,i1-1]} - \Pi_{[1,j1-1]}^{[1,i1-1]})$,where Π_J^I represent $\sum_{i\in I, j\in J} \Pi_{i,j}$.

Similarly, the matrix Q can be computed in $O(B)$ time after constructing a prefix sum table of similar form for $\Pi[i,j]A(i,j)$.

$$
\begin{aligned}
& D[1]\sum_{i,j}\Pi[i,j](\Psi_1[j]-\Psi_1[i-1])(\Psi_k[j]-\Psi_k[i-1])+...+ \\
& D[B]\sum_{i,j}\Pi[i,j](\Psi_B[j]-\Psi_B[i-1])(\Psi_k[j]-\Psi_k[i-1]) \\
= \quad & \sum_{i,j}\Pi[i,j](A^{(DM)}[j]-A^{(DM)}[i-1])(\Psi_k[j]-\Psi_k[i-1]) \qquad (5)
\end{aligned}
$$

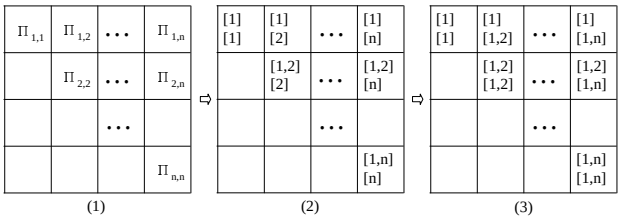
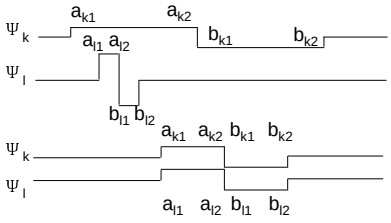


Figure 6. Example of generation of a prefix sum table for $P[k,l]$



If $i-1<a_{k1}$,
1:if $i\le j\le a_{k2}$, $\Psi_k[j]-\Psi_k[i-1]=\frac{1}{c_k}$
2:if $b_{k1}\le j\le b_{k2}$, $\Psi_k[j]-\Psi_k[i-1]=-\frac{1}{c_k}$
else $\Psi_k[j]-\Psi_k[i-1]=0$,
since $\Psi_k[j]=\Psi_k[i-1]=0$.
If $a_{k1}\le i-1\le a_{k2}$,
3:if $b_{k1}\le j\le b_{k2}$, $\Psi_k[j]-\Psi_k[i-1]=-\frac{2}{c_k}$
4:if $b_{k2}<j$, $\Psi_k[j]-\Psi_k[i-1]=-\frac{1}{c_k}$
else $\Psi_k[j]-\Psi_k[i-1]=0$,
since $\Psi_k[j]=\Psi_k[i-1]=\frac{1}{c_k}$.
If $b_{k1}\le i-1\le b_{k2}$,
5:if $b_{k2}<j$, $\Psi_k[j]-\Psi_k[i-1]=\frac{1}{c_k}$
else $\Psi_k[j]-\Psi_k[i-1]=0$,
since $\Psi_k[j]=\Psi_k[i]=-\frac{1}{c_k}$.

Figure 5. 5 intervals in which $\Psi_k[j]-\Psi_k[i-1]\neq 0$ (c_k is the normalization factor for Ψ_k)

As a result, the time required to generate the matrix P and Q is only $O(B^2)$ with the help of their prefix sum tables, which need $O(\frac{n(n+1)}{2})$ to construct. Therefore the total cost is successfully reduced from $O(n^2B^2+B^3)$ to $O(n^2+B^3)$, including the $O(B^3)$ time for solving equation $PD=Q$.

B. Weight-mapping algorithm

In some scenario, for example, when monitoring the trends of data changes through its approximation, one may prefer to approximate the original data A instead of its prefix sum data $\widehat{A^{(DM)}}$. In this case, we need to find the new weights that are associated with A, since the original weights $\Pi[i,j]$ are given for interval $[i,j]$.

Similar to the data-mapping scenario, the new weights can be derived from the error function (Equation 2) by substituting $A(i,j)$ with $\sum_{i\le k\le j}A[k]$, i.e., $\Pi^{(WM)}[k]=\sum_{i\in[1,k],j\in[k,n]}\sqrt{\Pi[i,j]}$

Now the new error function $PD=Q$, composed by equations of the type $\frac{\partial\epsilon^{(R)}}{\partial D[k]}=0$, can be expanded as the following (Equation 6):

$$
\begin{aligned}
& D[1]\sum_{i,j}\Pi[i,j]\Psi_k(i,j)\Psi_1(i,j)+ \\
& D[2]\sum_{i,j}\Pi[i,j]\Psi_k(i,j)\Psi_2(i,j) \\
& +...+ \\
& D[B]\sum_{i,j}\Pi[i,j]\Psi_k(i,j)\Psi_B(i,j) \\
= \quad & \sum_{i,j}\Pi[i,j]\Psi_k(i,j)A(i,j) \qquad (6)
\end{aligned}
$$

As for the weight mapping algorithm, there is a constant number of intervals, e.g. in this case 13 intervals, in which $\Psi_k(i,j)\Psi_l(i,j)\neq 0$. In these intervals, if we compute the prefix table for each of the $i\Pi[i,j], j\Pi[i,j], i^2\Pi[i,j], j^2\Pi[i,j], ij\Pi[i,j]$, and $\Pi[i,j]A(i,j)$, then the matrices P and Q can be generated in $O(B^3)$ time. As a result, the total cost for the Weight-mapping algorithm is still $O(n^2+B^3)$, due to the prefix table construction time $O(n^2)$ and equation solving time $O(B^3)$.

We conclude this section by providing the flow-charts of the Data-mapping and Weight-mapping algorithms shown in Figure 7. Major strengths of these two algorithms are their capability to severely reduce the complexity of the original problem. They only require $O(n^2)$ time and space to compute the new data set and $O(n^2+B^3)$ time and $O(n^2+B^2)$ space to compute wavelets and coefficients. As a consequence, the total running time of those algorithms is bounded by $O(n^2+B^3)$, while the

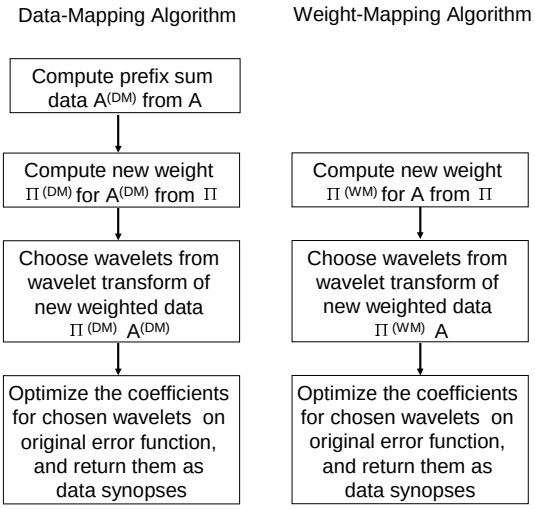


Figure 7. Data-mapping and Weight-mapping algorithm

total space is bounded by $O(n^2 + B^2)$.

V. Tracking dynamic changes of weights and data

In the previous sections III and IV we introduced the algorithms to generate the data synopses. As we discussed in the introduction, in databases, data and weights might change over time. In this section we discuss how to keep the data synopses accurate over time when dealing with dynamic data and weights.

For Point-wise approximation, when a data point or a weight value changes, only up to $\log(n)$ wavelets need to change. By comparing them with the chosen B wavelets, the new wavelets can be found in $max(log(n), B)$ time. Thus, the updating time is bounded by $O(\log(n) + B^3)$ with $O(B^3)$ time required to find the new coefficients.

For Range-sum queries, the dominant part of the cost is associated to the prefix sum table construction time, that requires $O(n^2)$. In the following we propose an incremental *multi-step* method to keep the overall cost below $O(n + B^3)$ while requiring an extra $O(n)$ space to store updates. We describe the new method in the following sections for the data-mapping and the weight-mapping algorithms for both the cases of dynamic data and weights.

A. Data change in data-mapping algorithm

For the data mapping algorithm, a change of a single data point may cause the change of up to n values in the prefix sum data $A^{(DM)}$, depending on the location of the data point.

First, the method needs to recompute the wavelet transform of $A^{(DM)}$, which requires $O(n)$ time. Now, since $A^{(DM)}$ is not involved in the generation of the matrix P, the data change does not require any re-computation of the matrix P (see Equation (5)). However this is not the case for the vector Q.

As a second step, we need to re-compute the vector Q. It is straightforward to notice that if we update the prefix sum table for Q, i.e. prefix sum table of $\Pi[i,j]A(i,j)$, and compute Q directly from the new table, the cost of its re-computation will be $O(n^2)$, due to the fact that all $O(n^2)$ entries $\begin{smallmatrix}[1,j]\\[1,i]\end{smallmatrix}$ with $i \geq t$ or $j \geq t$ will be affected by such update.

Next, we show that by considering some inner properties of the vector Q, the method can further reduce its re-computation time to $O(B)$ in case of a single data point change, or more generally, to $O(B + x)$ for x changes. Recall that there is only a constant number of intervals in which $Q[k] \neq 0$. Let $\{I, J\}$ be these non-zero intervals, and $v_{I,J}$ be one of the $Q[k]$ values specific for the interval I, J, i.e. $Q[k]$ can be seen as the sum of $v_{I,J}$s over all intervals $\{I, J\}$s, i.e., $Q[k] = \sum_{I,J} v_{I,J}$.

Now, the new $Q'[k]$ can be computed from the old vector $Q[k]$ in constant time. To prove that, let's assume that the difference between the new data value $A'[t]$ and the old data value $A[t]$ is $\delta_t^A = A'[t] - A[t]$.

We start from the update in one of its non-zero interval $I = [i1, i2], J = [j1, j2]$. Because only when $t \in I, J$, the new update will be reflected in the partial sum data, that is when $i \leq j < t$ or $t < i \leq j$, $A'(i,j) = A(i,j)$, and when $i \leq t \leq j$, $A'(i,j) = A(i,j) + \delta_t^A$.

Therefore we can separate the new value $v'_{I,J}$ into two parts: one includes $A'[t]$, i.e., $i \in I \cap [1,t], j \in J \cap [t,n]$, and the other does not, i.e., $i \in I \cap [1,t), j \in J \cap [1,t)$ and $i \in I \cap (t,n], j \in J \cap (t,n]$.

Thus, $v'_{I,J}$ can be computed from $v_{I,J}$ in $O(1)$ time (Equation (7)) with the help of the prefix sum table of $\Pi[i,j]$. As a consequence $Q'[k]$ can be computed from $Q[k]$ in constant time, since the number of I, Js is only a small constant. Therefore the total update time for Q is $O(B)$.

More generally, when there are x number of changes in the data, the total updating time of Q is $O(B + x)$ obtained by adding the changes to Q one by one.

When $x < n$, we compute $Q[k]$ using the original table, then update it to $Q'[k]$: The update time contains the following.
(1) Recompute wavelet coefficients: $O(n)$.
(2) Choose B wavelets and compute P and Q: $O(B^2)$.
(3) Update Q to Q' for all δ_t^A: $O(B + x)$.
(4) Solve new equation $PD = Q'$: $O(B^3)$.

When the changes reach n, we recompute the prefix sum tables, and run the algorithm on the new tables, which takes $O(n^2 + B^3)$.

So the amortized cost is $O(n + B^3)$, because $O(\frac{1}{n}[\sum_{x=1}^{n-1}(n + B^3 + x) + (n^2 + B^3)]) = O(n + B^3)$

B. Data change in weight mapping algorithm

The only difference between the weight mapping and the data mapping resides in the first step of the method. When $A[t]$ changes, the time involved in finding new wavelets is $O(\log(n))$, instead of $O(n)$. In the second step, only the vector Q's prefix table changes with $A[t]$, since $P[k,l] = \sum_{i,j} \Pi[i,j]\Psi_k(i,j)\Psi_l(i,j)$, and $Q[k] = \sum_{i,j} \Pi[i,j]\Psi_k(i,j)A(i,j)$. The amortized cost is still $O(n + B^3)$, due to

$$
\begin{aligned}
v'_{I,J} &= \sum_{\substack{i\in I\cap[1,t)\\ j\in J\cap[1,t)}} \Pi[i,j]A(i,j) + \sum_{\substack{i\in I\cap(t,n]\\ j\in J\cap(t,n]}} \Pi[i,j]A(i,j) + \sum_{\substack{i\in I\cap[1,t]\\ j\in J\cap[t,n]}} \Pi[i,j]A'(i,j) \\
&= \sum_{\substack{i\in I\cap[1,t)\\ j\in J\cap[1,t)}} \Pi[i,j]A(i,j) + \sum_{\substack{i\in I\cap(t,n]\\ j\in J\cap(t,n]}} \Pi[i,j]A(i,j) + \sum_{\substack{i\in I\cap[1,t]\\ j\in J\cap[t,n]}} \Pi[i,j]A(i,j) + \sum_{\substack{i\in I\cap[1,t]\\ j\in J\cap[t,n]}} \Pi[i,j]\delta_t^A \\
&= \sum_{\substack{i\in I\\ j\in J}} \Pi[i,j]A(i,j) + (min(i2,t)-i1)*(j2-max(j1,t))\delta_t^A \sum_{i,j}\Pi[i,j] \\
&= v_{I,J} + (min(i2,t)-i1)*(j2-max(j1,t))\delta_t^A \sum_{i,j}\Pi[i,j] \qquad (7)
\end{aligned}
$$

$O(\frac{1}{n}[\sum_{x=1}^{n-1}(\log(n)+B^3+x)+(n^2+B^3)]) = O(n+B^3)$.

If a weight value $\Pi[i,j]$ changes to $\Pi'[i,j]$, where $\sum_{i,j}\Pi'[i,j] = \sum_{i,j}\Pi[i,j] + \delta_{i,j}^{\Pi}$, both $P[k,l]$ and $Q[k]$ need to be updated.

By applying the same method in data updates, we can compute $P[k,l]$ and $Q[k]$ from the original prefix sum tables, and add $\delta_{i,j}^{\Pi}$ or $\delta_{i,j}^{\Pi}A(i,j)$ to compute $P'[k,l]$ and $Q'[k]$.

C. Weight change in data mapping algorithm

In the data mapping algorithm, a single update in the original weights Π leads to at most two changes in $\Pi^{(DM)}$ (Equation (4)). Thus only $O(\log(n))$ wavelet coefficients need to be re-calculated. Because of these new coefficients, the Equation $PD = Q$ has to be re-generated, which requires $O(B^2)$ time. Similar to the case of the data changes, for x changes in weights, the update time from $PD = Q$ to $P'D = Q'$ is $O(B^2 + x)$ for $P[k,l]$ and $O(B + x)$ for $Q[k]$. The time required to solve the equation $PD = Q$ is then $O(B^3)$. Therefore, the amortized time for x updates is $O(\frac{1}{n}[\sum_{x=1}^{n-1}(\log(n) + B^3 + x) + (n^2 + B^3)]) = O(n + B^3)$.

D. Weight change in weight mapping algorithm

The only difference between the data-mapping and the weight mapping regards the fact that a single update in weights may cause up to n changes in weighted data $\Pi^{(WM)} \odot A$. Thus, we need $O(n)$ wavelet transform time, and the amortized cost is still $O(n + B^3)$, because $O(\frac{1}{n}[\sum_{x=1}^{n-1}(n + B^3 + x) + (n^2 + B^3)]) = O(n + B^3)$

In summary, our incremental method exploits the previous computed values residing in the database to compute the new values, avoiding the new values to be calculated from scratch. Therefore, this method can successfully reduce the update cost for both data-mapping and weight-mapping from linear to sub-linear in time.

VI. Special Cases for Range-Sum Weights

In the previous sections, we discussed the most general cases of weights, in which each point of the weight may differ from the other ones. However, there are some special scenarios in which weights may have some nice structures that, if fully exploited, might help to further reduce the running time. In this section we discuss three of them: *Uniform Weights*, *Uniform Length Weights* and *Hierarchical Sum Weights*.

Uniform weights This case, also known as "unweighted range-sum problem", assume that there is only one unique weight for all intervals, i.e. $\forall i,j,k,l$, $\Pi[i,j] = \Pi[k,l]$.

Uniform length weights This case assume that the weights are same if their interval lengths are the same, i.e. $\Pi[i,j] = \Pi[k,l]$, if $j-i = l-k$, and $\Pi[i,j] = \Pi[k,l]+h$, if $j-i = l-k+1$, where h is the unit weight difference. In this context, we use $\Pi_{|l|}$ to represent the weight for an interval with length l, such that $\Pi_l = \Pi_1 + (l-1)*h$.

Hierarchical sum weights This case assume the existence of a unique weight $\Pi_{i,i}$ for each i. All other weights can be derived from them, i.e. $\Pi_{i,j} = \sum_{k=i}^{j}\Pi_{k,k}$.

Recall that in section IV, after we constructed the prefix tables, the computational costs for generating P, Q and solving $PD = Q$ were completely independent from n. The dominant part of the overall cost $O(n^2 + B^3)$ turned out to be the table construction time $O(n^2)$.

For these special cases, where weights for different points and intervals are not totally independent, we will show that we can further reduce down the table size from $O(n^2)$ to $O(n)$ by exploiting the existence of an inner dependency between weights and intervals.

Before starting, we introduce a simple but important lemma that we will use later to analyze the cost reduction.

Lemma 2: Given a generic vector $V = \{V[1], V[2], ..., V[n]\}$, the sum of any arithmetic series of its entries, i.e. $f(k,d,i,j) = kV[i] + (k+d)V[i+1] + ... + (k+(j-i)d)V[j]$, can be computed in $O(1)$ time after $O(n)$ preprocessing time.

Proof We generate the following two prefix sum tables from vector V in $O(n)$ time.

$$
\begin{aligned}
S_1^V &= \{V[1], V[1]+V[2], ..., V[1]+...+V[n]\} \\
S_2^V &= \{V[1], V[1]+2V[2], ..., V[1]+...+nV[n]\}
\end{aligned}
$$

Then any $f(k,d,i,j)$ can be computed in 9 operations as in Equation (8).

$$
\begin{aligned}
& f(k,d,i,j) \\
= \;& kV[i] + ... + (k+(j-i)d)V[j] \\
= \;& kV(i,j) + d(V[i+1] + ... + (j-i)V[j]) \\
= \;& k(S_1^V[j] - S_1^V[i-1]) + d(S_2^V[j] - S_2^V[i]) \\
& -id(S_1^V[j] - S_1^V[i]) \qquad (8)
\end{aligned}
$$

□

For all these special cases, the prefix sum tables of size $O(n^2)$ can be reduced to a vector of size $O(n)$, which is similar to S_1^V, S_2^V. Furthermore, with the help of these new prefix vectors the entries of both matrix P and vector Q can still be computed in $O(1)$ time. Running time for all other parts of the algorithm remains same, except the table construction time is reduced from $O(n^2)$ to $O(n)$. So the total running time is reduced to $O(n + B^3)$.

In this section, we discuss only the prefix vector for Q, i.e., the prefix vector of $\sum_{i\in I, j\in J} \Pi[i,j]A(i,j)$. The prefix vector for P (prefix vector of $\sum_{i\in I, j\in J} \Pi_{i,j}$) can be considered as a special case of the previous one when $A(i,j) = 1$. In the following we use the uniform weight case as an example to show how to reduce the prefix table size, and summarize the results for all three cases in table II.

We also want to remark to the reader that we need to consider only two cases of overlapping intervals, e.g., $I \cap J = \phi$ and $I = J$, to further simplify our computation, because if $I \cap J \neq \phi$, and $I \neq J$, then $j1 \leq i2$. This interval can be divided into 3 parts: $I1 = [i1, j1 - 1], J1 = [j1, j2]$, $I2 = [j1, i2], J2 = [j1, i2]$ and $I3 = [j1, i2], J3 = [i2 + 1, j2]$. In these new intervals $I1 \cap J1 = \phi$, $I2 = J2$ and $I3 \cap J3 = \phi$. This division creates 3 subintervals, each of them can be solved by using the prefix vectors defined in the table I.

Example: prefix table reduction for uniform weights

In the uniform weight case, $\Pi_{i,j}$ is a constant for all intervals. We define $S[i] = \sum_{k=1}^{i} A[k]$.

$$
\begin{aligned}
& \sum_{i\in I, j\in J} A(i,j) = \sum_{i\in I, j\in J} (S[j] - S[i-1]) \\
= \;& (i2 - i1 + 1)(S[j1] + ... + S[j2]) \\
- \;& (j2 - j1 + 1)(S[i1-1] + S[i2-1]) \\
= \;& (i2 - i1 + 1)(S3[j2] - S3[j1-1]) \\
- \;& (j2 - j1 + 1)(S3[i2-1] - S3[i1-2]) \qquad (9)
\end{aligned}
$$

where interval $I = [i1, i2], J = [j1, j2]$, and $S3$ is defined in table I.

As a result, if we pre-compute a prefix sum vector $S3$, the prefix sum $\sum_{i\in I, j\in J} A(i,j)$ for any interval I, J can be computed in constant time as in equation (9). Therefore $\sum_{i\in I, j\in J} \Pi[i,j]A(i,j) = \Pi[i,j] \sum_{i\in I, j\in J} A(i,j)$ can be computed in $O(1)$ time from $\sum_{i\in I, j\in J} A(i,j)$ since $\Pi[i,j]$ is a constant.

In this section, we have shown that our algorithm can be easily applied to different special weights, with a dramatic complexity reduction. This is because our algorithms capture the properties of the real complex component, i.e., the $O(n^2)$ weights, through its prefix sum table. When the weights can be simplified, the cost of our algorithm can be simplified accordingly.

TABLE I.
PREFIX SUM TABLES

$S_1 = \{\Pi[1,1], \Pi[1,1] + \Pi[2,2], ..., \Pi[1,1] + ... + \Pi[n,n]\}$
$S_2 = \{\Pi[1,1], \Pi[1,1] + 2\Pi[2,2], ..., \Pi[1,1] + ... + n\Pi[n,n]\}$
$S_3 = \{S[1], S[1] + S[2], ..., S[1] + ... + S[n]\}$
$S_4 = \{S_1[1]S[1], S_1[2]S[2], ..., S_1[n]S[n]\}$
$S_5 = \{S_1[0]S[1], S_1[1]S[2], ..., S_1[n-1]S[n]\}$
$S_6 = \{S_2[1]S[1], S_2[2]S[2], ..., S_2[n]S[n]\}$
$S_7 = \{\Pi_{1,1}S_3[1], 2\Pi_{2,2}S_3[2], ..., n\Pi_{n,n}S_3[n]\}$
$S_8 = \{\Pi_{1,1}S_3[1], \Pi_{2,2}S_3[2], ..., \Pi_{n,n}S_3[n]\}$
$S_9 = \{S_3[1], S_3[1] + S_3[2], ..., S_3[1] + ... + S_3[n]\}$
$S_{10} = \{S_3[1], S_3[1] + 2S_3[2], ..., S_3[1] + ... + nS_3[n]\}$

VII. EXPERIMENTS

In this section, we show the accuracy and efficiency of our method on different data sets. We use the relative error defined as $RE = \frac{\epsilon_B}{\epsilon_0} = \frac{\sum_i \Pi[i](A[i] - \widehat{A}[i])^2}{\sum_i \Pi[i](A[i])^2}$ for point-wise approximation, and $RE = \frac{\epsilon_B}{\epsilon_0} = \frac{\sum_{i,j} \Pi_{i,j}(A^{[i,j]} - \widehat{A}^{[i,j]})^2}{\sum_{i,j} \Pi_{i,j}(A^{[i,j]})^2}$ for range-sum approximation, where ϵ_B is the absolute approximation error with B buckets. The experiment was done by using a Linux machine with 2.80GHz processor and 2047 MB memory.

A. Point-wise queries

In this section, we compare the 2-step, M-step and W-wav algorithms using both synthetic and real data sets. In these experiments, we set $I = 1$ for the M-step method. For other I values, the approximation error is between error of 2-step, and error of M-step with $I = 1$.

a) Synthetic Data: The synthetic data set contains 4096 data points, and they are extracted from a normal, an exponential and a uniform distribution. The weights are extracted from a zipf distribution with $\alpha = 0.2$, 0.5 and 0.8. In the following we compare the 2-step and M-step algorithms for Point-wise approximation against the W-Way algorithm in terms of *accuracy*, *efficiency*, *time* and *skewness*.

Accuracy Accuracy results are shown in Figure 8(a) when the data set is extracted from an exponential distribution with parameter $\lambda = 0.01$. Major consideration to make is related to the shape of these curves: the error for W-wav jumps to $2.5\epsilon_0$ and remains there even after $B = 200$ while both the 2-step and M-step always keep the error under $0.5\epsilon_0$. The reason of such behavior is related to the fact that when a data set contains very large values, W-wav takes more wavelets from this region than necessary, while the 2-step and M-step algorithms can set 0 as coefficients for the "unwanted" wavelets.

Efficiency Efficiency results are shown in Figure 8(b) when the data set is extracted from a normal distribution with mean=10 and variance=100. Notice here the 2-step and M-step algorithms are capable to reduce the error to $0.5\epsilon_0$ around $B = 200$, while the error of W-wav is still

TABLE II.
WEIGHT REDUCTION TABLE

$I \cap J = \phi$	$LS_j = ((L_2+1)\Pi_{\lvert L_1\rvert} + \frac{L_2(L_2+1)h}{2} S_3[j1+L_2] - [(\Pi_{\lvert 1\rvert} + h(L_1-1))(S_9[j1+L_2] - S_9[j1-2]) + h[(1-j1)(S_9[j1+L_2] - S_9[j1-1]) + (S_{10}[j1+L_2] - S_{10}[j1-1])$
$I \cap J = \phi$	$LS_i = \Pi_{\lvert L_1\rvert}(S_9[i1+L_2] - S_9[i1-1]) + h(1-i1)(S_9[i1+L_2-1] - S_9[i1-1]) + (S_{10}[i1+L_2-1] - S_{10}[i1-1]) + \Pi_{\lvert L_1\rvert} \frac{L_2(L_2+1)h}{2} S_3[i1-1]$
$I = J$	$LS_j = ((L_2+1)\Pi_{\lvert 1\rvert} + \frac{L_2(L_2+1)h}{2} S_3[i2] - [\Pi_{\lvert 1\rvert}(S_9[i2] - S_9[i1-2]) + h[(1-i1)(S_9[i2] - S_9[i1-1]) + (S_{10}[i2] - S_{10}[i1-1])$
$I = J$	$LS_i = \Pi_{\lvert 1\rvert}(S_9[i2] - S_9[i1-1]) + h(1-i1)(S_9[i2-1] - S_9[i1-1]) + (S_{10}[i2-1] - S_{10}[i1-1]) + \Pi_{\lvert L_1\rvert} \frac{L_2(T_2+1)h}{2} S_3[i1-1]$
$I \cap J = \phi$	$HS_j = [(1-i1)(S_1[i2] - S_1[i1-1]) + (S_2[i2] - S_2[i1-1]))](S_3[j2] - S_3[j1-1]) + (i2-i1+1)[S_4[j2] - S_4[j1-1] + (2S_1[j1-1] - S_1[i2])(S_3[j2] - S_3[j1-1])]$
$I \cap J = \phi$	$HS_i = (j2-j1+1)[S_1[j1-1](S_3[i2] - S_3[i1-1]) - (S_5[i2] - S_5[i1-1])] + [(j2+1)(S_1[j2] - S_1[j1-1]) - (S_2[j2] - S_2[j1-1])]S^{[i1,i2]}$
$I = J$	$HS_j = (S_6[i2] - S_6[i1-1]) - S_2[i1-1](S_3[i2] - S_3[i1-1])$
$I = J$	$HS_i = (i2+1)(S_8[i2] - S_8[i1-1]) - (S_7[i2] - S_7[i1-1]) + (i2-i1+1) * (S_1[i2] - S_1[i1-1]) + i1(S_1[i2] - S_1[i1-1]) - (S_2[i2] - S_2[i1-1])$

LS_j and HS_j stands for $\sum_{i\in I, j\in J} \Pi_{i,j} \sum_{k=1}^{j} A[k]$, in uniform length weights case and hierarchical sum weights case; LS_i and HS_i stands for $\sum_{i\in I, j\in J} \Pi_{i,j} \sum_{k=1}^{i} A[k]$ in those cases. L_1 is start interval length, $L_1 = j1 - i1 + 1$. L_2 is max shift length, $L_2 = min\{i2 - i1, j2 - j1\}$.

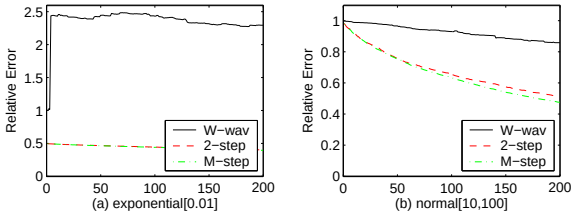


Figure 8. Accuracy

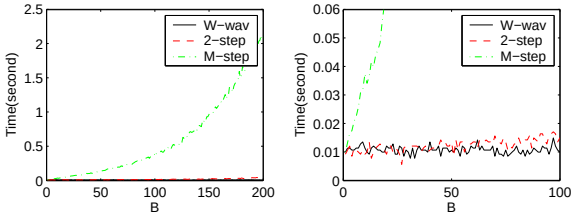
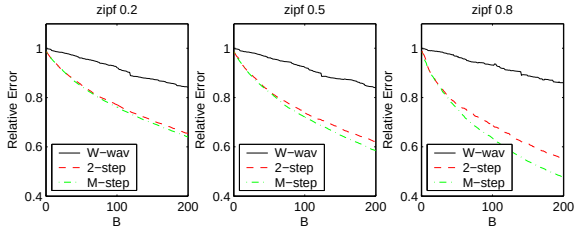
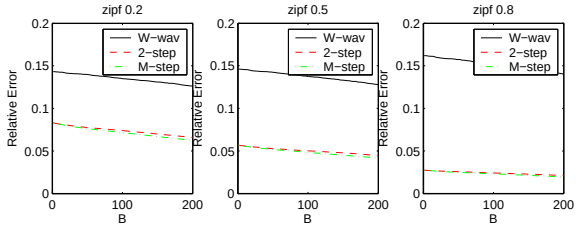


Figure 9. Efficiency



(a)Normal$[10, 100]$ data



(b)Uniform$[1, 10]$ data

Figure 10. Skewness

as high as $0.85\epsilon_0$. This is caused by cancellation among overlapped wavelets using the original coefficients. The 2-step and M-step algorithms can lower this side-effect by finding the best coefficient for each wavelet.

Time In Figures 9 we show the running time of the three algorithms M-step method requires a very large running time. It reaches 2 second when $B = 200$ (figure 8(a)). With the figure at a smaller scale (figure 8(b)),the running time for W-wav is $O(n)$, and is constant for all B. Running time for the 2-step is bounded by $O(n + B^3)$. For $B^3 < n$, that is $B < 16$, there is no difference of running time between the 2-step and W-wav algorithms. Even when B reaches 200, $200^3 \gg 4096$, the extra time for the 2-step is only 0.05 second.

Skewness Last, in Figures 10(a) and 10(b) we compare the three algorithms while changing the weight distributions (zipf 0.2, 0.5 and 0.8). The data set is extracted from normal and uniform distributions. Important to notice here how the performance of the W-wav method decreases as weights become more skewed, e.g. from to zipf 0.8. This characteristic can be explained because W-wav method combines weights with wavelets. The more skewed the weights are, the higher will be the probability that wavelets with heavy weights will be used. On the other hands, both the 2-step and M-step algorithms can ignore those exaggerated wavelets by setting their coefficients to 0.

b) Real Data: In order to validate the above performance metrics on more realistic data, we use a data set collected from the 66th day of the World Cup 1998 [20]. We chose this data set because it represents the set with the largest number of points. In these experiments, the data represents the query subject, which may be a web page or a picture, while the data value represents the max

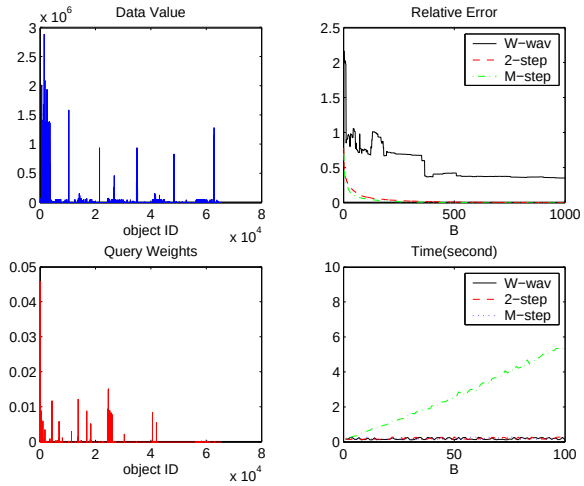


Figure 11. Relative error compare for world cup data

response size of the subject. Weights correspond to the number of queries for each data point after normalization. In this case as well, we observe the same dynamics as before for the relative error; the W-wav method provides a relative error above $2\epsilon_0$ at first, then it is slowly reduced to $0.35\epsilon_0$ at $B = 600$ and remains there through $B = 1000$ (Figure 11). The 2-step algorithm reduces its error to $0.35\epsilon_0$ with $B = 13$ while the M-step reduces its error to $0.35\epsilon_0$ with only $B = 9$. The running time difference between the 2-step method and the W-wav method is very small when $B < 100$.

B. Range-Sum Approximation

In this section we compare our proposed algorithm, the data mapping method, the weight mapping method with the following method, including the Naive method described in Section IV.

- **Data** Our data-mapping method, where $\Pi^{(DM)}[i] = \sum_{k=1}^{i} \sqrt{\Pi[k,i]} + \sum_{k=i+1}^{n} \sqrt{\Pi[i+1,k]}$
- **Data2** A simple straight forward data-mapping method, $\Pi^{(DM)}[k] = \sum_{1 \le i \le j \le k} \sqrt{\Pi[i,j]}$
- **Weight** Our weight-mapping method, $\Pi^{(WM)}[k] = \sum_{i \in [1,k], j \in [k,n]} \Pi[i,j]$
- **Weight2** A simple subtractive weight-mapping method, $\Pi^{(WM)} = \Pi^{(DM)}[i] - \Pi^{(DM)}[i-1]$, we use $\Pi^{(DM)}$ in **Data**
- **Naive** Naive method with new signal $X = \{A(0,0), A(0,1), ..., A(n-1, n-1)\}$.

The *Naive* method produces relative errors in the range $40 - 90$ while *Data* pushes it down below 1 (Figure 12). The reason of this dynamic is due to the fact that B is too small for data set with $O(n^2)$ length. In Figure 13 we compare the *Data* and *Data2* methods in terms of their accuracy. We highlight how *Data* performs better than *Data2* over different data sets, because the weights computed by *Data2* are not as accurate as the weights computed by *Data* that are derived directly from the error functions.

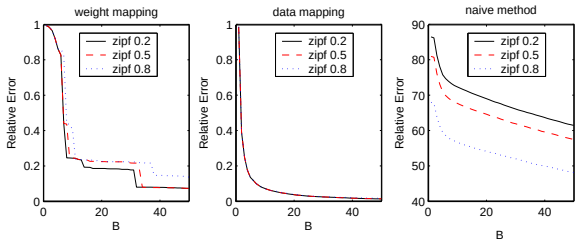


Figure 12. Relative Error for normal(10, 100) data

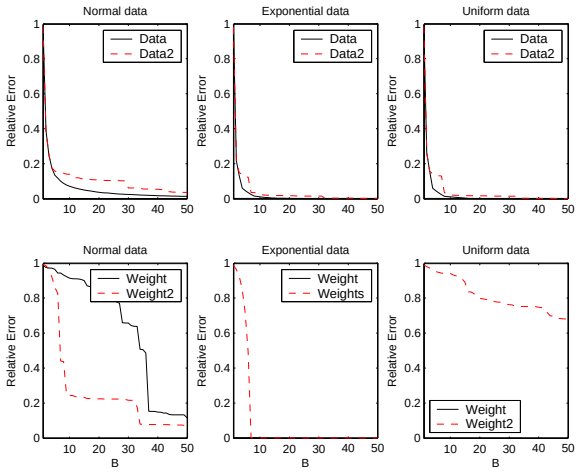


Figure 13. Data vs Data2($\alpha = 0.5$)

For exponential and uniform data, *Weight* is much better than *Weight2*. The error is almost 0. But in normal data, it is worse (Figure 13). The reason is that *Weight2* focuses only on intervals start with i, $i+1$, $i-1$ or ends with i, $i-1$, $i+1$, but ignores all other intervals $[i,j]$ that cover it. For the exponential and uniform data set, some intervals may contain very different values from others. Ignoring those intervals will incur large errors. *Weight* considers all intervals that cover the data point, but it exaggerates the middle part of the signal. For normal data, all intervals are similar, exaggeration of certain intervals will make them unfairly important than others. This causes error.

In Figure 14 we show relative error of *Data* and *Weight* method as a function of skewness of weights (zipf 0.2, 0.5 and 0.8). As a consequence, the skewness of the queries does not affect the algorithm accuracy because both *Data* and *Weight* method sums certain $\Pi[i,j]$s together to compute a new weight. This summation cancels out the skewness effect.

VIII. Conclusion

We studied nonuniform approximation for both point-wise and range-sum queries. Although the approximation is one dimension, it can be easily generalized to multi-dimensions, because the methods we used to choose wavelets and coefficients are not restricted by dimensionality. The wavelets are selected from the optimal solution for weighted data, however they are not optimal with respect to the original data. To overcome the above problem, we add a second step to optimize their coefficients for the

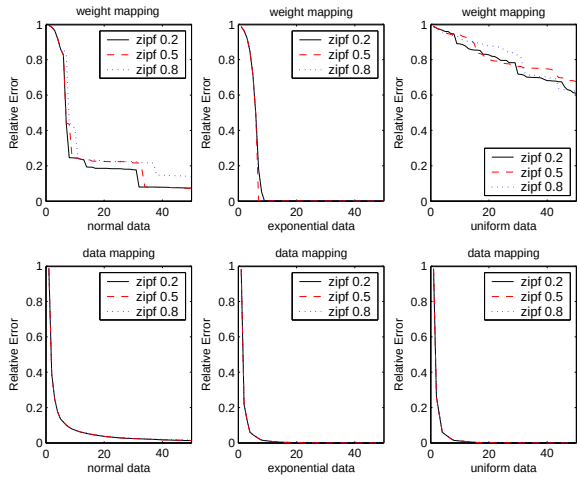


Figure 14. Query skewness

original data. This takes extra $O(B^3)$ time, but improves accuracy significantly. How to find the optimal wavelets for original data is still an open problem.

ACKNOWLEDGMENT

We thank Muthu Muthukrishnan for numerous inspiring discussions concerning this work, and reviewers for their detailed comments.

REFERENCES

[1] Y. Matias and D. Urieli, "Optimal workload-based weighted wavelet synopses," in *ICDT*, 2005.
[2] A. Aboulnaga and S. Chaudhuri, "Self-tuning Histograms: Building Histograms Without Looking at Data," *SIGMOD*, 1999.
[3] N. Bruno, S. Chaudhuri, and L. Gravano, "STHoles: A Multidimensional Workload-Aware Histogram," Tech. Rep., 2001.
[4] C. M. Chen and N. Roussopoulos, "Adaptive Selectivity Estimation Using Query Feedback," 1994, pp. 161–172.
[5] V. Ganti, M.-L. Lee, and R. Ramakrishnan, "ICICLES: Self-Tuning Samples for Approximate Query Answering," 2000, pp. 176–187.
[6] A. C. Konig and G. Weikum, "Combining histograms and parametric curve fitting for feedback-driven query result-size estimation," in *VLDB*, 1999.
[7] V. Markl, G. M. Lohman, and V. Raman, "Leo: An autonomic query optimizer for db2," in *IBM Systems Journal*, vol. 42, 2003.
[8] S. Muthukrishnan, M. Strauss, and X. Zheng, "Workload-optimal histograms on streams," in *ESA*, 2005.
[9] S. Muthukrishnan, "Subquadratic algorithms for workload-aware haar wavelet synopses," in *FSTTCS*, 2005.
[10] N. Koudas, S. Muthukrishnan, and D. Srivastava, "Optimal histograms for hierarchical range queries," in *PODS*, 2000.
[11] S. Muthukrishnan and M. Strauss, "Rangesum histograms," in *SODA*, 2003.
[12] N. Thaper, S. Guha, P. Indyk, and N. Koudas, "Dynamic multidimensional histograms," in *SIGMOD*, 2002.
[13] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim, "Approximate query processing using wavelets," vol. 10, no. 2-3.
[14] M. Garofalakis and A. Kumar, "Deterministic wavelet thresholding for maximum-error metrics," in *PODS*, 2004, pp. 166–176.
[15] Y. Matias and D. Urieli, "Optimal wavelet synopses for Range-Sum Queries," in *Proceedings of the 1st ACM Conference on Computer and Communications Security*, Nov. 1993.
[16] Y. Matias, J. S. Vitter, , and M. Wang, "Wavelet-Based Histograms for Selectivity Estimation," in *SIGMOD*, 1998.
[17] Y. Matias, J. S. Vitter, and M. Wang, "Dynamic Maintenance of Wavelet-Based Histograms," in *VLDB*, 2000.
[18] S. Guha and B. Harb, "Wavelet Synopsis for Data Streams: Minimizing Non-Euclidean Error," *KDD*, 2005.
[19] S. Guha, N. Koudas, and D. Srivastava, "Fast Algorithms for Hierarchical Range Histogram Construction," 2002.
[20] Http://ita.ee.lbl.gov/html/contrib/WorldCup.html.
[21] S. Chen and A. Nucci, "Nonuniform Compression in Databases with Haar Wavelet," 2007, pp. 223–232.
[22] M. Garofalakis and P. B. Gibbons, "Wavelet Synopses with Error Guarantees," 2002, pp. 476–487.
[23] S. Guha and B. Harb, "Approximation Algorithm for Wavelet Transform Coding of Data Stream," in *SODA*, 2006.
[24] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss, "Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries," 2001.
[25] L. Lim, M. Wang, and J. S. Vitter, "SASH: A Self-Adaptive Histogram Set for Dynamically Changing Workloads," 2003.

Su Chen is currently a Ph.D. candidate at Rutgers, The State University of New Jersey, USA. She received her MS degree in computer science from Fudan University, Shanghai, China, in 2001. Her research interests are massive data analysis, with applications in both network and database area, including modeling, approximation, mining and compression for both online and offline data processing.

Dr. Antonio Nucci is the Chief Technology Officer at Narus. He obtained his Ph.D. in Electrical Engineering from Politecnico di Torino, Turin, Italy in October 2003. His research interests include network measurement and management, traffic analysis, network security and surveillance, network design and traffic engineering. In his career he published more than 50 technical papers, articles in industry outlets such as Pipeline, Converge, Network Digest, IP Telephony and IEEE Spectrum, applied for 22 patents in optical and IP networking technology, and co-author a definitive textbook on managing large IP networks, titled "Design, Measurement and Management of Large-Scale IP Networks. Bridging the gap between Theory and Practice" to be published by Cambridge University Press in February 2008. He is very active both in the research arena and in the industry market space. He served numerous program/organizing committees of prestigious conferences such as IEEE INFOCOM and ACM SIGMETRICS.