Hierarchical Binary Histograms for Summarizing Multi-Dimensional Data

F. Furfaro, G. M. Mazzeo,
DEIS - University of Calabria
Via P. Bucci - Rende (ITALY)
lastname@si.deis.unical.it

D. Saccà,
ICAR - CNR
Via P. Bucci - Rende (ITALY)
sacca@icar.cnr.it

C. Sirangelo
DEIS - University of Calabria
Via P. Bucci - Rende (ITALY)
sirangelo@si.deis.unical.it

ABSTRACT

The need to compress data into synopses of summarized information often arises in many application scenarios, where the aim is to retrieve aggregate data efficiently, possibly trading off the computational efficiency with the accuracy of the estimation. A widely used approach for summarizing multi-dimensional data is the histogram-based representation scheme, which consists in partitioning the data domain into a number of blocks (called buckets), and then storing summary information for each block. In this paper, a new histogram-based summarization technique which is very effective for multi-dimensional data is proposed. This technique exploits a multi-resolution organization of summary data, on which an efficient physical representation model is defined. The adoption of this representation model (based on a hierarchical organization of the buckets) enables some storage space to be saved w.r.t. traditional histograms, which can be invested to obtain finer grain blocks, thus approximating data with more detail. Experimental results show that our technique yields higher accuracy in retrieving aggregate information from the histogram w.r.t. traditional approaches (classical multi-dimensional histograms as well as other types of summarization technique).

Categories and Subject Descriptors

E.4 [**CODING AND INFORMATION THEORY**]: Data compaction and compression; H.2.4 [**DATABASE MANAGEMENT**]: Miscellaneous

General Terms

Algorithms, Management

Keywords

Histograms, Multi-dimensional data, Range queries

1. INTRODUCTION

The need to compress data into synopses of summarized information often arises in many application scenarios, where the aim is to retrieve aggregate data efficiently, possibly trading off the computational efficiency with the accuracy of the estimation. Examples of these application contexts are range query answering in OLAP services [10], selectivity estimation for query optimization in RDBMSs [2, 9], statistical and scientific data analysis, window query answering in spatial databases [1], and so on. All of these scenarios are mainly interested in aggregating data within a specified range of the domain – these kinds of aggregate query are called *range queries*. To support efficient query answering, information is often represented adopting the multi-dimensional data model: data are stored as a set of measure values associated to points in a multi-dimensional space.

A widely used approach for summarizing multi-dimensional data is the histogram-based representation scheme, which consists in partitioning the data domain into a number of blocks (called *buckets*), and then storing summary information for each block [4, 5]. The answer of a range query evaluated on the histogram (without accessing the original data) is computed by aggregating the contributions of each bucket. For instance, a sum range query (i.e. a query returning the sum of the elements contained inside a specified range) is evaluated as follows. The contribution of a bucket which is completely contained inside the query range is given by its sum, whereas the contribution of a bucket whose range is external w.r.t. the query is null. Finally, the contribution of the blocks which partially overlap the range of the query is obtained estimating which portion of the total sum associated to the bucket occurs in the query range. This estimate is evaluated performing linear interpolation, i.e. assuming that the data distribution inside each bucket is uniform (*Continuous Values Assumption - CVA*), and thus the contribution of these buckets is generally approximate (unless the original distribution of frequencies inside these buckets is actually uniform).

It follows that querying aggregate data rather than the original ones reduces the cost of evaluating answers (as histogram size is much less than original data size), but introduces estimation errors, as data distributions inside buckets are not, in general, actually uniform. Therefore, a central problem when dealing with histograms is finding the partition which provides the "best" accuracy in reconstructing query answers. This can be achieved by producing partitions whose blocks contain as uniform as possible data distributions (so that CVA is well-founded).

Many effective summarization techniques have been proposed for data having a small number of dimensions. Unfortunately, these methods do not scale up to any number

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'05 March 13-17, 2005, Santa Fe, New Mexico, USA
Copyright 2005 ACM 1-58113-964-0/05/0003 ...$5.00.

of dimensions, so that finding a technique effective for high-dimensionality data is still an open problem.

1.1 Main Contribution

In this paper we propose a new class of multi-dimensional histogram which is based on binary hierarchical partitions, i.e. which can be constructed by recursively splitting blocks of the data domain into pairs of sub-blocks. One of the innovations introduced in our work is the definition of a different representation model: buckets are represented implicitly by storing the hierarchical partition overlying the histogram, whose representation can be accomplished more efficiently than the explicit representation of bucket boundaries. That is, the hierarchy adopted for determining the structure of a histogram is also used as a basis for representing it, thus introducing surprising efficiency in terms of both space consumption and accuracy of estimations.

In particular, our histogram (namely *GHBH* - *Grid Hierarchical Binary Histogram*) is based on a constrained partition scheme, where blocks of data cannot be split anywhere along one of their dimensions, but the split must be laid onto a grid partitioning the block into a number of equally sized sub-blocks. The adoption of this constrained partitioning can be exploited to define a more efficient physical representation of the histogram w.r.t. classical approaches, thus enabling a larger number of buckets to be stored within the same storage space bound. Thus, the saved space is invested by *GHBH* to obtain finer grain blocks, which approximate data in more detail.

In order to exploit the increase of the number of buckets effectively, we also introduce a new greedy criterion to choose, at each step of the construction process, the block to be split and where to split it. By means of experiments, we show that our technique produces lower error rates than well-known histograms MHIST [9] and MinSkew [1], as well as other state-of-the-art wavelet based summarization techniques [10, 11]. Experiments also show that *GHBH* can be effectively applied on high-dimensionality data, as its accuracy turns out to be almost unaffected by the increase of dimensionality.

2. BASIC NOTATIONS

Throughout the paper, a d-dimensional data distribution D is assumed. D will be treated as a multi-dimensional array of integers of size $n_1 \times \cdots \times n_d$. A *range* ρ_i on the i-th dimension of D is an interval $[l..u]$, such that $1 \leq l \leq u \leq n_i$. Boundaries l and u of ρ_i are denoted by $lb(\rho_i)$ (*lower bound*) and $ub(\rho_i)$ (*upper bound*), respectively. The size of ρ_i will be denoted as $size(\rho_i) = ub(\rho_i) - lb(\rho_i) + 1$. A *block* b (of D) is a d-tuple $\langle \rho_1, \ldots, \rho_d \rangle$ where ρ_i is a range on the dimension i, for each $1 \leq i \leq d$. Informally, a block represents a "hyper-rectangular" region of D. A block b of D with all zero elements is called a *null block*. Given a point in the multidimensional space $\mathbf{x} = \langle x_1, \ldots, x_d \rangle$, we say that $\mathbf{x}$ belongs to the block b (written $\mathbf{x} \in b$) if $lb(\rho_i) \leq x_i \leq ub(\rho_i)$ for each $i \in [1..d]$. A point $\mathbf{x}$ in b is said to be a *vertex* of b if for each $i \in [1..d]$ x_i is either $lb(\rho_i)$ or $ub(\rho_i)$. The sum of the values of all points inside b will be denoted by $sum(b)$.

Any block b inside D can be split into two sub-blocks by means of a $(d-1)$-dimensional hyper-plane which is orthogonal to one of the axis and parallel to the other ones. More precisely, if such a hyper-plane is orthogonal to the $i-$th dimension and intersects the orthogonal axis by dividing the range ρ_i of b into two parts $\rho_i^{low} = [lb(\rho_i)..x_i]$ and $\rho_i^{high} = [(x_i+1)..ub(\rho_i)]$, then the block b is divided into two sub-blocks $b^{low} = \langle \rho_1, \ldots, \rho_i^{low}, \ldots, \rho_d \rangle$ and $b^{high} = \langle \rho_1, \ldots, \rho_i^{high}, \ldots, \rho_d \rangle$. The pair $< b^{low}, b^{high} >$ is said the *binary split* of b along the dimension i at the position x_i. The i-th dimension is called *splitting dimension*, and the coordinate x_i is called *splitting position*.

Informally, a binary hierarchical partition can be obtained by performing a binary split on D (thus generating the two sub-blocks D^{low} and D^{high}), and then recursively partitioning these two sub-blocks with the same binary hierarchical scheme.

Definition 1. *Given a multi-dimensional data distribution D, a* binary partition $BP(D)$ *of D is a binary tree such that:*

1. *every node of $BP(D)$ is a block of D;*
2. *the root of $BP(D)$ is the block $\langle [1..n_1], \ldots, [1..n_d] \rangle$;*
3. *for each internal node p of $BP(D)$, the pair of children of p is a binary-split on p.*

In the following, the root, the set of nodes, and the set of leaves of the tree underlying a binary partition BP will be denoted, respectively, as $Root(BP)$, $Nodes(BP)$, and $Leaves(BP)$. An example of a binary partition defined on a two dimensional data distribution D of size $n \times n$ is shown in Fig. 1(a).

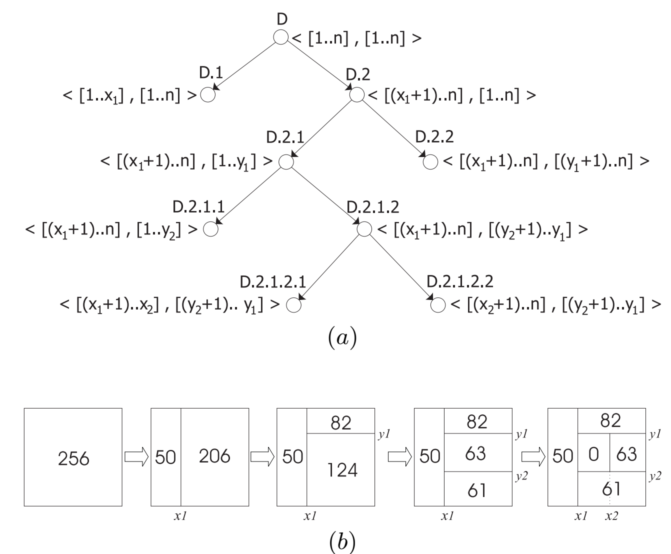

Figure 1: A binary partition

3. GHBH: GRID HIERARCHICAL BINARY HISTOGRAM

The idea underlying Grid Hierarchical Binary Histogram consists in storing the partition tree explicitly, in order to both avoid redundancy in the representation of the bucket boundaries and provide a structure indexing buckets. To enhance the efficiency of the histogram physical representation, we introduce further constraints on the partition scheme adopted to define the boundaries of the buckets. In particular every split of a block is constrained to lie onto a grid, which divides the block into a number of equally sized

sub-blocks. This number is a parameter of the partition, and it is the same for every block of the partition tree. As will be shown later, the adoption of this constraint enables some storage space to be saved, and to be invested to obtain finer grain blocks within the same storage space bound.
In the following, a binary split on a block $b = < \rho_1, \ldots, \rho_d >$ along the dimension i at the position x_i will be said a *binary split of degree k* if $x_i = lb(\rho_i) + \left\lceil j \cdot \frac{size(\rho_i)}{k} \right\rceil - 1$ for some $j \in [1..k-1]$.

DEFINITION 2. *Given a multi-dimensional data distribution D, a* grid binary partition *of degree k on D is a binary partition $GBP(D)$ such that for each non-leaf node p of $GBP(D)$ the pair of children of p is a binary-split of degree k on p.*

DEFINITION 3. *Given a multi-dimensional array D, a* Grid Hierarchical Binary Histogram of degree k *on D is a pair $GBH(D) = \langle P, S \rangle$ where P is a grid binary hierarchical partition of degree k on D, and S is the set of pairs $\langle p, sum(p) \rangle$ where $p \in Nodes(P)$.*

In the following, given $GHBH = \langle P, S \rangle$, $Nodes(GHBH)$ will denote the set $Nodes(P)$, whereas $Buckets(GHBH)$ will denote the set $Leaves(P)$. Fig. 2 shows an example of the construction of a two-dimensional 4th degree *GHBH*.

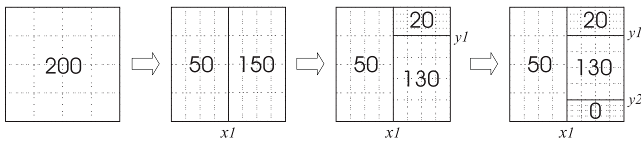

Figure 2: A 4th degree *GHBH*

3.1 Physical representation

A grid hierarchical binary histogram $GHBH = \langle P, S \rangle$ can be stored efficiently by representing P and S separately, and by exploiting some intrinsic redundancy in their definition. To store P, first of all we need one bit per node to specify whether the node is a leaf or not. As the nodes of P correspond to ranges of the multi-dimensional space, some information describing the boundaries of these ranges has to be stored. This can be accomplished efficiently by storing, for each non leaf node, both the splitting dimension and the splitting position which define the ranges corresponding to its children. As regards the splitting position, observe that, for a grid binary partition of degree k, it can be stored using $\lceil log\ (k-1) \rceil$ bits. If we did not adopt a grid constraining splits, we should use 32 bits to store the splitting position. This is the reason why the adoption of a grid enables some storage space to be saved. In the following, we will consider degree values which are a power of 2, so that the space consumption needed to store the splitting position will be simply denoted as $log\ k$. Therefore, each non leaf node can be stored using a string of bits, having length $log\ k + \lceil log\ d \rceil + 1$, where $log\ k$ bits are used to represent the splitting position, $\lceil log\ d \rceil$ to represent the splitting dimension, and 1 bit to indicate that the node is not a leaf. On the other hand, 1 bit suffices to represent leaf nodes, as no information on further splits needs to be stored. Therefore, the partition tree P can be stored as a string of bits (denoted as $String_P(GHBH)$) consisting in the concatenation of the strings of bits representing each node of P.

The pairs $\langle p_1, sum(p_1) \rangle, \ldots, \langle p_m, sum(p_m) \rangle$ of S (where $m = |Nodes(GHBH)|$) can be represented using an array containing the values $sum(p_1), \ldots, sum(p_m)$, where the sums are stored according to the ordering of the corresponding nodes in $String_P(GHBH)$. This array consists in a sequence of m 32-bit words and will be denoted as $String_S(GHBH)$. Indeed, it is worth noting that not all the sum values in S need to be stored, as some of them can be derived. For instance, the sum of every right-hand child node is implied by the sums of its parent and its sibling. Therefore, for a given grid hierarchical binary histogram *GHBH*, the set $Nodes(GHBH)$ can be partitioned into two sets: the set of nodes that are the right-hand child of some other node (which will be called *derivable nodes*), and the set of all the other nodes (which will be called *non-derivable nodes*). Derivable nodes are the nodes which do not need to be explicitly represented as their sum can be evaluated from the sums of non-derivable ones. This implies that $String_S(GHBH)$ can be reduced to the representation of the sums of only non-derivable nodes.

This representation scheme can be made more efficient by exploiting the possible sparsity of the data. In fact it often occurs that the size of the multi-dimensional space is large w.r.t. the number of non-null elements. Thus we expect that null blocks are very likely to occur when partitioning the multi-dimensional space. This leads us to adopt an ad-hoc compact representation of such blocks in order to save the storage space needed to represent their sums. A possible efficient representation of null blocks could be obtained by avoiding storing zero sums in $String_S(GHBH)$ and by employing one bit more for each node in $String_P(GHBH)$ to indicate whether its sum is zero or not. Indeed, it is not necessary to associate one further bit to the representation of derivable nodes, since deciding whether they are null or not can be done by deriving their sum. Moreover observe that we are not interested in *GHBH*s where null blocks are further split since, for a null block, the zero sum provides detailed information of all the values contained in the block, thus no further investigation of the block can provide a more detailed description of its data distribution. Therefore any *GHBH* can be reduced to one where each null node is a leaf, without altering the description of the overall data distribution that it provides. It follows that in $String_P(GHBH)$ non-leaf nodes do not need any additional bit either, since they cannot be null. According to this new representation model, each node in $String_P(GHBH)$ is represented as follows:

- if the node is not a leaf it is represented using a string of length $log\ k + \lceil log\ d \rceil + 1$ bits, where $log\ k$ bits are used to represent the splitting position, $\lceil log\ d \rceil$ to represent the splitting dimension, and 1 bit to indicate that the node is not a leaf.
- if the node is a leaf, it is represented using one bit to state that the node has not been split and, only if it is a non-derivable node, one additional bit to specify whether it is null or not.

On the other hand $String_S(GHBH)$ represents the sum of all the non-derivable nodes which are not null. Fig. 3 shows the representation of the grid hierarchical binary histogram of Fig. 2 obtained by adopting the physical representation model explained above.

In Fig. 3 non-derivable nodes are colored in grey, whereas derivable nodes are white. Derivable leaf nodes of *GHBH*

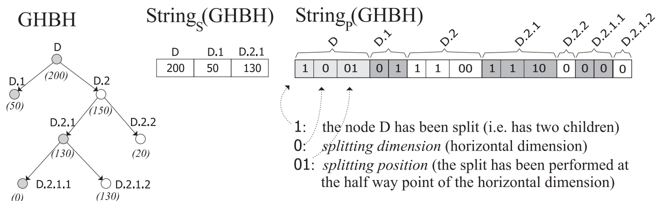

Figure 3: Representing the *GHBH* of Fig. 2

(such as the node D.2.1.2) are represented in $String_P(GHBH)$ by means of a unique bit, with value 0. Non-derivable leaf nodes (such as the nodes D.1 and D.2.1.1) are represented in $String_P(GHBH)$ by means of a pair of bits: the first of the two has value 0 (saying that the node has not been split), and the other indicates whether the sum associated to the node is zero or not.

As regards non-leaf nodes, the first bit of their representation has value 1 (meaning that these nodes have been split); the second bit is 0 if the node is split along the horizontal dimension, otherwise it is 1. The last 2 bits represent the splitting position (lying onto a 4th degree grid).

According to the physical representation model presented above, it can be easily shown that maximum size of a *GHBH* with β buckets is given by $\beta \cdot (35 + log\ k + \lceil log\ d \rceil) - (log\ k + \lceil log\ d \rceil + 2)$, which corresponds to the case that all but one leaf nodes are non-derivable, and all non-derivable nodes are not null.

It can be also shown that the maximum number of buckets of a *GHBH* built within the storage space bound B is given by: $\beta^{max}_{GHBH} = \left\lfloor \frac{B + log\ k + \lceil log\ d \rceil - 30}{3 + log\ k + \lceil log\ d \rceil} \right\rfloor$. This expression can be computed by considering the case that the available storage space B is equal to the minimum storage space consumption of the *GHBH* histogram.

By comparing the formula expressing β^{max}_{GHBH} with the maximum number of buckets $\beta^{max}_{flat} = \left\lfloor \frac{B}{32(2 \cdot d + 1)} \right\rfloor$ which could be obtained if a "flat" representation model were adopted (i.e. if each bucket were represented by storing its $2 \cdot d$ vertices), it is easy to show that *GHBH* enables a larger number of buckets to be stored within the same storage space bound.

4. GREEDY ALGORITHM

Our approach works as follows. It starts from the binary histogram whose partition tree has a unique node (corresponding to the whole D) and, at each step, selects the leaf of the binary-tree which is the most in need of partitioning and applies the most effective split to it. In particular, the splitting position must be selected among all the positions laid onto the grid overlying the block. Both the choices of the block to be split and of the position where it has to be split are made according to some greedy criterion. Every time a new split is produced, the free amount of storage space is updated, in order to take into account the space needed to store the new nodes, according to the different representation schemes. If any of these nodes corresponds to a block with sum zero, we save the 32 bits used to represent the sum of its elements. Anyway, only one of the two nodes must be represented, since the sum of the remaining node can be derived by difference, by using the parent node.

A number of possible greedy criteria can be adopted for choosing the block which is most in need of partitioning and how to split it. We tried several greedy criteria, and it turned out that the most effective one (called *Max-Var/Max-Red*) consists in choosing, at each step, the block b having maximum SSE (the SSE of a block b is defined as $SSE(b) = \sum_{\mathbf{j} \in b_i} (D[\mathbf{j}] - avg(b_i))^2$), and splitting it at $< dim, pos >$ producing the maximum reduction of $SSE(b)$ (i.e. $SSE(b) - (SSE(b^{low}) + SSE(b^{high}))$ is maximum w.r.t. every possible split on b).

The resulting algorithm scheme is shown below. It uses a priority queue q where nodes of the histogram are ordered according to their need to be partitioned. At each step, the node at the top of the queue is extracted and split, and its children are in turn enqueued. Before adding a new node b to the queue, the function *Evaluate* is invoked on b. This function returns both the value of $SSE(b)$ (which represents a measure of the need of being partitioned of b), and the position dim, pos of the most effective split (i.e. which yields the largest reduction of SSE). In particular, the splitting positions to be evaluated and compared are all the positions lying onto the grid defined on b.

Greedy Algorithm
B is the storage space available for the summary.
begin
  $q.initialize(\ );$
  $b_0 := \langle [1..n_1], \ldots, [1..n_d] \rangle;$
  $H := new\ Histogram(b_0);$
  $B := B - 32 - 2;$ // the space needed to store
                // H is subtracted from B
  $< need, dim, pos > =\ Evaluate(b_0);$
  $q.Insert(<\ b_0, < need, dim, pos > >);$
  while $(B > 0)$
    $< b,\ < need, dim, pos > >= q.GetFirst(\ );$
    $< b^{low}, b^{high} >= BinarySplit(b, dim, pos);$
    MemUpdate(B,b,dim,pos);
    if $(B \geq 0)$
      $H := Append(H, b, b^{low}, b^{high});$
      $q.Insert\left(< b^{low}, Evaluate(b^{low}) >\right);$
      $q.Insert\left(< b^{high}, Evaluate(b^{high}) >\right);$
    end_if
  end_while
  return H;
end

Therein:

- the instruction $H := new\ Histogram(b_0)$ builds a *GHBH* consisting in the unique bucket b_0;
- the procedure *MemUpdate* takes as argument the storage space B and the last performed split, and updates B to take into account this split;
- the function *Append* updates the histogram by inserting $< b^{low}, b^{high} >$ as child nodes of b.

5. EXPERIMENTAL RESULTS

In this section we present some experimental results about the accuracy of estimating sum range queries on hierarchical histograms. Performances (in terms of accuracy) of *GHBH*s are evaluated and compared with some state-of-the-art techniques in the context of multi-dimensional data summarization.

5.1 Measuring approximation error

The exact answer of a sum query q_i will be denoted as S_i, and the estimated answer as $\tilde{S}_i$. The *relative error* is defined as: $e_i^{rel} = \frac{|S_i - \tilde{S}_i|}{S_i}$. Observe that relative error is not defined when $S_i = 0$.
The accuracy of the various techniques has been evaluated by measuring the average relative error $\| e^{rel} \|$ of the answers of range queries belonging to the query set $QS^+(Vol)$, containing the sum range queries defined on all the ranges of volume Vol whose actual answer *is not* null.

5.2 Synthetic data

Our synthetic data are similar to those of [3, 11]. They are generated by creating an empty d-dimensional array D of size $n_1 \times \cdots \times n_d$, and then by populating r regions of D by distributing into each of them a portion of the total sum value T. The size of the dimensions of each region is randomly chosen between l_{min} and l_{max}, and the regions are uniformly distributed in the multi-dimensional array. The total sum T is partitioned across the r regions according to a Zipf distribution with parameter z. To populate each region, we first generate a Zipf distribution whose parameter is randomly chosen between z_{min} and z_{max}. Next, we associate these values to the cells in such a way that the closer a cell to the center of the region, the larger its value. Outside the dense regions, some isolated non-zero values are randomly assigned to the array cells. As explained in [3, 11], data-sets generated by using this strategy well represent many classes of real-life distributions.

5.3 Comparison with other techniques

We compared the effectiveness *GHBH* algorithm with the state-of-the-art techniques for compressing multi-dimensional data. In particular, we analyzed the histogram-based techniques MHIST [9] and MinSkew [1], and with the wavelet-based techniques proposed respectively in [10] and [11]. The experiments were conducted at the same storage space. First, we briefly describe these three techniques; then, we present the results of the comparison.

MHIST (*Multi-dimensional Histogram*). An MHIST histogram is built by a multi-step algorithm which, at each step, chooses the block which is the most in need of partitioning (as explained below), and partitions it along one of its dimensions. The block to be partitioned is chosen as follows. First, the *marginal distributions* [1] $marg_1(b), \ldots, marg_d(b)$ are computed for each block. The block b to be split is the one which is characterized by a marginal distribution (along any dimension i) which contains two adjacent values e_j, e_{j+1} with the largest difference w.r.t. every other pair of adjacent values in any other marginal distribution of any other block. b is split along the dimension i by putting a boundary between e_j and e_{j+1}. For each non-split block b, the sum of its elements, and the positions of the front corner and the far corner of the *minimal bounding rectangle (MBR)* containing all non-null elements of b are stored.

MinSkew. The MinSkew algorithm works as the MHIST one. The main difference between the two algorithms is that MinSkew uses a different criterion to select the block to be split and where to split it. That is, it tries all possible splits along every dimension of every block, and evaluates how much the SSE of the marginal distribution along the splitting dimension is reduced by the split. The block b and the splitting position $< dim, pos>$ are returned if the reduction of $SSE(marg_{dim}(b))$ obtained by splitting b along dim at position pos is maximum w.r.t. the reduction of any $SSE(marg_i(b))$ (where $i \in [1..d]$) which could be obtained by performing some split along i. The main difference between our *GHBH* and MinSkew is that the latter adopts a greedy criterion working on the marginal distributions, whereas the criterion used by *GHBH* investigates the internal distribution of candidate blocks. Moreover, *GHBH* uses a grid constraining the position of all possible splits.

Wavelet-based Compression Techniques. We have considered the two wavelet-based techniques presented in [11] (that will be referred as WAVE1) and in [10] (WAVE2). The former applies the wavelet transform directly on the source data, whereas the latter performs a pre-computation step. First, it generates the partial sum data array of the source data, and replaces each of its cells with its natural logarithm. Then, the wavelet compression process is applied to the array obtained in such a way.

Diagrams (a) and (b) of Fig. 4 are obtained on four-dimensional synthetic data of size $8 \times 32 \times 256 \times 2048$, with density 0.1% and z=1. Several *GHBH*s of different degrees have been tested. The term *GHBH*(x) is used to denote a *GHBH* which uses x bits to store the splitting position. For instance, *GHBH*(0) is a *GHBH* where blocks can only be split only at the half way point of any dimension, so that no bit is spent to store the splitting position. Analogously, *GHBH*(3) is a *GHBH* where splits must be laid onto a grid partitioning block dimensions in 2^3 equal size portions, and so on.

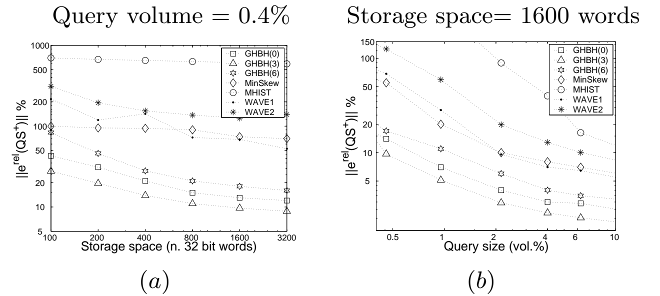

Figure 4: Comparing techniques

From the results shown in these diagrams, we can draw the conclusion that the use of grids provides an effective trade-off between the accuracy of splits and the number of splits which can be generated within a given storage space bound. The effectiveness of this trade-off depends on the degree of the allowed binary splits. In fact, when a high degree is adopted, a single split can be very “effective” in partitioning a block, in the sense that it can produce a pair of blocks which are more uniform w.r.t. the case that the splitting position is constrained to be laid onto a coarser grid. On the other hand, the higher the degree of splits, the larger the amount of storage space needed to represent each split. From our results, it emerges that *GHBH*(3) (using binary splits of degree $2^3 = 8$) gives the best performances in terms of accuracy, and as the number of bits used to define the grid increases, the accuracy decreases (the results of *GHBH*(x), for $x \neq 0, 3, 6$, are not reported for the sake of presentation of the diagrams).

[1] The marginal distribution of a block b along the i-th dimension, denoted as $marg_i(b)$, is given by the “projection” of the internal data distribution of b on the i-th dimension.

5.4 Sensitivity on dimensionality

We have tested the behavior of all the techniques when synthetic data with increasing dimensionality are considered. Diagrams 5(a) and (b) have been obtained by starting from a 7-dimensional data distribution (called D^7) containing about 18 million cells, where 15000 non null values (density=0.08%) are distributed among 300 dense regions. The data distributions with lower dimensionality (called D^i, with $i \in 3..6$) have been generated by projecting the values of D^7 on the first i of its dimensions. In this way, we have created a sequence of multi-dimensional data distributions, with increasing dimensionality (from 3 to 7) and with decreasing density (from 11% to 0.08%). Diagram 5(a) has been obtained by considering, for each D^i, all the range queries whose edges are a half of the size of the corresponding dimension of D^i. That is, we considered queries of size $\frac{1}{2^3} \cdot Vol(D)$ (that is 12.5%) in the 3D case, $\frac{1}{2^4} \cdot Vol(D)$ (that is 6.25%) in the 4D case, up to $\frac{1}{2^7} \cdot Vol(D)$ (that is 0.78%) in the 7D case. Likewise, diagram 5(b) has been obtained by considering range queries whose edges are 30% of the corresponding dimension of D^i. We point out that we could not consider queries with constant sizes (w.r.t. the volume of the data), as the size of "meaningful" queries in high dimensions is likely to be smaller than in low dimensions. For instance, in the 3D case a cubic query whose volume is 10% of the data volume can be considered "meaningful", as each of its edges is less than a half of the size of its corresponding dimension (as $0.1 \approx 0.46^3$; in the 10D case, a 10% query is not so meaningful, as it selects about the 80% of the size of every dimension ($0.1 \approx 0.8^{10}$). Both the diagrams 5(a) and 5(b) have been obtained by setting the compression ratio equal to 10% (the compression ratio for D^i is given by the ratio between the number of words used to represent the histogram, and the number of words used for the (sparse) representation of D^i).

Diagrams 5(a) and 5(b) show that the accuracy of every technique decreases as dimensionality increases, but *GHBH* gets worse very slightly. The worsening of MHIST and MinSkew at high dimensions could be due to the fact that, as dimensionality decreases, the projection has the effect of collapsing several distinct dense regions into the same one. This means that low dimensionality data consists in much less dense regions than high dimensionality ones. Therefore, every kind of histogram needs much more buckets to locate dispersed dense regions in the high dimensionality case w.r.t. the low dimensional one. Thus, we can conjecture that the number of buckets produced by MHIST and MinSkew, within the given space bound, does not suffice to distribute dense regions among different buckets: that is, these two techniques tend to include several dense regions into the same bucket, thus providing a poor description of their content. On the contrary, *GHBH* (due to the larger number of buckets built in the same storage space, and to the different criterion adopted to determine how to split buckets) manages to locate and partition dense regions by means of different buckets.

6. CONCLUSIONS

We have introduced a new class of histogram (namely, *GHBH*) which exploits his particular partition paradigm to make the representation of the histogram buckets more efficient w.r.t. the traditional "flat" representation scheme

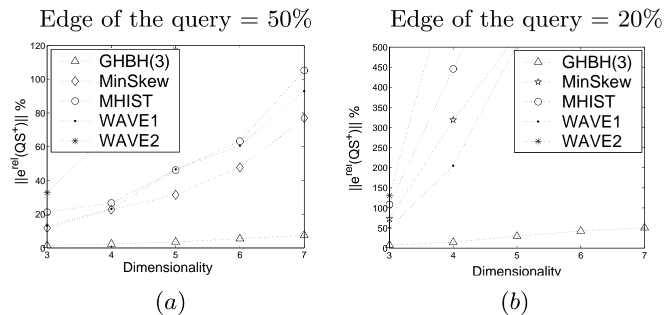

Figure 5: Sensitivity on dimensionality

adopted for classical histograms. In a *GHBH* every split must lie onto a grid dividing the block into a fixed number of equally sized sub-blocks. The adoption of this grid improves the efficiency of the physical representation, enabling some storage space to be saved and invested to obtain finer grain buckets, thus enhancing the accuracy of estimating range queries on the histogram.

We have provided experimental results comparing our histograms with other state-of-the-art multi-dimensional compression techniques, proving the effectiveness of our proposal, also for high-dimensionality data distributions.

7. REFERENCES

[1] Acharya, S., Poosala, V., Ramaswamy, S., Selectivity estimation in spatial databases, *Proc. ACM SIGMOD Conf. 1999*, Philadelphia (PA), USA.

[2] Chaudhuri, S., An overview of query optimization in relational systems, *Proc. PODS 1998*, Seattle (WA), USA.

[3] Garofalakis, M., Gibbons, P. B., Wavelet synopses with error guarantees, *Proc. ACM SIGMOD Conf. 2002*, Madison (WI), USA.

[4] Ioannidis, Y. E., Poosala, V., Balancing histogram optimality and practicality for query result size estimation, *Proc. ACM SIGMOD Conf. 1995*, San Josè (CA), USA.

[5] Jagadish, H. V., Jin, H., Ooi, B. C., Tan, K.-L., Global optimization of histograms, *Proc. SIGMOD Conf. 2001*, Santa Barbara (CA), USA.

[6] Kooi, R.P., *The optimization of queries in relational databases*, PhD thesis, CWR University, 1980.

[7] Korn, F., Johnson, T., Jagadish, H. V., Range selectivity estimation for continuous attributes, *Proc. SSDBM Conf. 1999*, Cleveland (OH), USA.

[8] Muthukrishnan, S., Poosala, V., Suel, T., On rectangular partitioning in two dimensions: algorithms, complexity and applications, *Proc. ICDT 1999*, Jerusalem, Israel.

[9] Poosala, V., Ioannidis, Y. E., Selectivity estimation without the attribute value independence assumption, *Proc. VLDB Conf. 1997*, Athens, Greece.

[10] Vitter, J. S., Wang, M., Iyer, B., Data cube approximation and histograms via wavelets, *Proc. CIKM 1998*, Washington, USA.

[11] Vitter, J. S., Wang, M., Approximate computation of multidimensional aggregates of sparse data using wavelets, *Proc. ACM SIGMOD Conf. 1999*, Philadelphia (PA), USA.