A New Histogram-based Technique for Compressing Multi-Dimensional Data

Filippo Furfaro[1], Giuseppe M. Mazzeo[1], Domenico Saccà[1,2] and Cristina Sirangelo[1]

[1] D.E.I.S. - Università della Calabria
[2] ICAR - CNR

{furfaro, mazzeo, sacca, 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.

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 [12], selectivity estimation for query optimization in RDBMSs [2, 11], statistical and scientific data analysis, window query answering in spatial databases [1, 9], 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 [5, 6]. The answer to 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 of dimensions, so that finding a technique effective for high-dimensionality data is still an open problem.
In this paper we propose a new class of multi-dimensional histograms which is based on binary hierarchical partitions. This summary structure is obtained by recursively splitting blocks of the data domain into pairs of sub-blocks. We stress that binary hierarchical partition approaches have already been adopted for constructing histograms – indeed we shall discuss meaningful examples of such a class, that we call *FBH*, *Flat Binary Histograms*. The novelty of our histograms (namely, *GHBH* - Grid Hierarchical Binary Histogram) is that 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. Furthermore, *GHBH*s are 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 enables a more efficient physical representation of the histogram w.r.t. other histograms using more traditional partition schemes. Thus, the saved space can be invested to obtain finer grain blocks, which approximate data in more detail. Indeed, the ability of creating a larger amount of buckets (in a given storage space) does not guarantee a better accuracy in estimating range queries, as it could be the case that the buckets created by adopting a constrained scheme contain very skewed (non uniform) distributions. However, by means of several experiments (which are not reported in this paper, due to space limitations, but can be found in [3]), we compared our technique with state-of-the-art ones, and obtained that the use of a grid based partition scheme provides a virtuous trade off (which holds also in the high dimensionality context) between the possibility of creating a larger number of buckets and the restriction on the choice of the splitting position.

1.1 Related Work

Histograms were originally proposed in [7] in the context of query optimization in relational databases. Query optimizers compute efficient execution plans on the basis of the estimation of the size of intermediate results. In this scenario, histograms were introduced to summarize the frequency distributions of single-attribute values in database relations to allow an efficient selectivity estimation of intermediate queries [2]. The frequency distribution of a single attribute A can be viewed as a one-dimensional array storing, for each value v of the attribute domain, the number of tuples whose A attribute has value v. A one-dimensional histogram (on the attribute A) is built by partitioning the frequency distribution of A into a set of non-overlapping blocks (called *buckets*), and storing, for each of these blocks, the sum of the frequencies contained in it (i.e. the total number of tuples where the value of the A attribute is contained in the range corresponding to the bucket).
The selectivity (i.e. the result size) of a query of the form $v' < R.A < v''$ is estimated on the histogram by evaluating a *range-sum* query, that is by summing the frequencies stored in the buckets whose bounds are completely contained inside $[v'..v'']$, and possibly by estimating the "contributions" of the buckets which partially overlap the query range. One-dimensional histograms are not suitable to estimate the selectivity of queries involving more than one attribute of a relation, i.e. queries of the form $v'_1 < R.A_1 < v''_1 \wedge \ldots \wedge v'_n < R.A_n < v''_n$. In this case, the *joint frequency distribution* has to be considered [11], i.e. a multi-dimensional array whose dimensions represent the attribute domains, and whose cell with coordinates $< v_1, \ldots, v_n >$ stores the number of tuples where $A_1 = v_1, \ldots, A_n = v_n$. The selectivity of a query Q of the form $v'_1 < R.A_1 < v''_1 \wedge \ldots \wedge v'_n < R.A_n < v''_n$ coincides with the sum of the frequencies contained in the multidimensional range $\langle [v'_1..v''_1], \ldots, [v'_n..v''_n] \rangle$ of the joint frequency distribution. In order to retrieve this aggregate information efficiently, a histogram can be built on the joint frequency distribution as in the one-dimensional case. A histogram on a multi-dimensional data distribution consists in a set of non overlapping buckets (hyper-rectangular blocks) corresponding to multi-dimensional ranges partitioning the overall domain.
The same need for summarizing multi-dimensional data into synopses of aggregate values often arises in many other application scenarios, such as statistical databases [8], spatial databases [1, 9] and OLAP [12]. In the latter case, the data to be summarized do not represent frequencies of attribute values, but measure values to be aggregated within specified ranges of the multidimensional space, in order to support efficient data analysis. This task is accomplished by issuing range queries providing the aggregate information which the users are interested in. The approximation introduced by issuing queries on summarized data (without accessing original ones) is tolerated as it makes query answering more efficient, and approximate answers often suffice to obtain useful aggregate information.
The effectiveness of a histogram (built in a given storage space bound) can be measured by measuring the uniformity of the data distribution underlying each of its buckets. As queries are estimated by performing linear interpolation on the aggregate values associated to the buckets, the more uniform the distribution inside the buckets involved in the query, the better the accuracy of the estimation. Therefore the effectiveness of

a histogram depends on the underlying partition of the data domain. In [10], the authors present a taxonomy of different classes of partitions, and distinguish *arbitrary*, *hierarchical*, and *grid-based* partitions. Grid-based partitions are built by dividing each dimension of the underlying data into a number of ranges, thus defining a grid on the data domain: the buckets of the histogram correspond to the cells of this grid. Hierarchical partitions are obtained by recursively partitioning blocks of the data domain into non overlapping sub-blocks. Finally, arbitrary partitions have no restriction on their structure. Obviously, arbitrary partitions are more flexible than hierarchical and grid-based ones, as there are no restrictions on where buckets can be placed. But building the "most effective" multi-dimensional histogram based on an arbitrary partition (called *V-Optimal* [5]) has been shown to be a NP-Hard problem, even in the two-dimensional case [10]. Therefore several techniques for building effective histograms (which can be computed more efficiently than the V-Optimal one) have been proposed. Most of these approaches are not based on arbitrary partitions. In particular, *MHIST-p* [11] is a technique using hierarchical partitions. The *MHIST-p* algorithm works as follows. First, it partitions the data domain into p buckets, by choosing a dimension of the data domain and splitting it into p ranges. Then, it chooses a bucket to be split and recursively partitions it into p new sub-buckets. The criterion adopted by *MHIST-p* to select and split the bucket which is the most in need of partitioning (called *MaxDiff*) investigates the differences between contiguous values in the marginal distributions of data (see [11] for more details). From the experiments in [11], it turns out that MHIST-2 (based on binary partitions) provides the best results. In [1] the authors introduce *MinSkew*, a technique refining MHIST to deal with selectivity estimation in spatial databases (where data distributions are two-dimensional). Basically, *MinSkew* first partitions the data domain according to a grid, and then builds a histogram as though each cell of the grid represented a single point of the data source. The histogram is built using the same hierarchical scheme adopted by MHIST-2, by adopting a different criterion for choosing the bucket to be split at each step, based on the variance of marginal distributions of data (see [1, 3]).
Other approaches to the problem of summarizing multi-dimensional data are the wavelet-based ones. Wavelets are mathematical transformations implementing a hierarchical decomposition of functions [4, 12, 13]. They were originally used in different research and application contexts (like image and signal processing), and have recently been applied to selectivity estimation [4] and to the approximation of OLAP range queries over data cubes [12, 13].

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 $n_1 \times \ldots \times n_d$. 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 **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 **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 $sum(b)$.

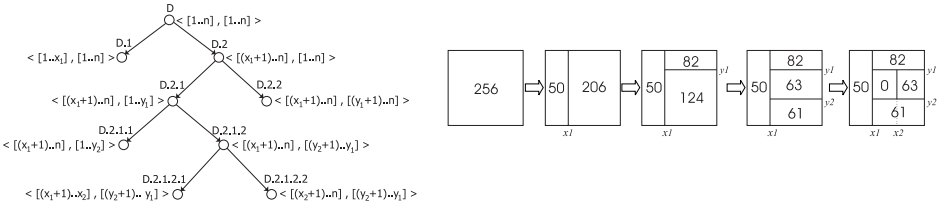
Any block b inside D can be split into two sub-blocks by means of a $(d-1)$-dimensional hyper-plane which is orthogonal to one of the axis and parallel to the other ones. More precisely, if such a hyper-plane is orthogonal to the $i-$th dimension and intersects the orthogonal axis by dividing the range 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).



(a) (b)

Fig. 1. A binary partition

3 Flat Binary Histogram

As introduced in Section 1.1, several techniques proposed in literature, such as MHIST and MinSkew, use binary partitions as a basis for building histograms. In this section we provide a formal abstraction of classical histograms based on binary partitions. We

refer to this class as *Flat Binary Histograms*, to highlight the basic characteristic of their physical representation model. The term "flat" means that, classically, buckets are represented independently from one another, without exploiting the hierarchical structure of the underlying partition.

Definition 2. *Given a multi-dimensional data distribution D, the* Flat Binary Histogram *on D based on the binary partition $BP(D)$ is the set of pairs:*

$$FBH(D) = \{<b_1, sum(b_1)>, \ldots, <b_\beta, sum(b_\beta)>\},$$

where the set $\{b_1, \ldots, b_\beta\}$ coincides with $Leaves(BP)$.

In the following, given $FBH(D) = \{<b_1, sum(b_1)>, \ldots, <b_\beta, sum(b_\beta)>\}$, the blocks $b_1, \ldots, b_\beta$ will be called *buckets* of $FBH(D)$, and the set $\{b_1, \ldots, b_\beta\}$ will be denoted as $Buckets(FBH(D))$.

Fig. 1(b) shows how the 2-dimensional flat binary histogram corresponding to the binary partition of Fig. 1(a) can be obtained by progressively performing binary splits on D. The histogram consists in the following set:

$\{ < \langle [1..x_1], [1..n] \rangle , 50 >, < \langle [x_1+1..n], [1..y_2] \rangle, 61 >, \quad < \langle [x_1+1..x_2], [y_2+1..y_1] \rangle, 0 >,$
$< \langle [x_2+1..n], [y_2+1..y_1] \rangle, 63 >, \quad < \langle [x_1+1 \,..\, n], [y_1+1 \,..\, n] \rangle, 82 > \}.$

A flat binary histogram can be represented by storing, for each bucket of the partition, both its boundaries and the sum of its elements. Assuming that 32 bits are needed to encode an integer value, $2 \cdot d$ 32-bit words are needed to store the boundaries of a bucket, whereas one 32-bit word is needed to store a sum value. Therefore, the storage space consumption of a flat binary histogram $FBH(D)$ is given by: $size(FBH)$=$(2 \cdot d + 1) \cdot 32 \cdot |Buckets(FBH)|$ bits. Thus, given a space bound B, the maximum number of buckets of an FBH that can be represented within B is $\beta^{max}_{FBH} = \left\lfloor \frac{B}{32 \cdot (2 \cdot d+1)} \right\rfloor$

4 Grid Hierarchical Binary Histogram

The hierarchical partition scheme underlying a flat binary histogram can be exploited to define a new class of histogram, which improves the efficiency of the physical representation. It can be observed that most of the storage consumption ($2 \cdot d \cdot 32 \cdot |Buckets(FBH)|$) of a flat binary histogram is due to the representation of the bucket boundaries. Indeed, buckets of a flat binary histogram cannot describe an arbitrary partition of the multi-dimensional space, as they are constrained to obey a hierarchical partition scheme. The simple representation paradigm defined in the previous section introduces some redundancy. For instance, consider two buckets b_i, b_{i+1} which correspond to a pair of siblings in the hierarchical partition underlying the histogram; then, b_i, b_{i+1} can be viewed as the result of splitting a block of the multi-dimensional space along one of its dimensions. Therefore, they have 2^{d-1} coinciding vertices. For FBH histograms, these coinciding vertices are stored twice, as the buckets are represented independently of each other. We expect that exploiting this characteristic should improve the efficiency of the representation.

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 3. *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 4. *Given a multi-dimensional array D, a* Grid Hierarchical Binary Histogram of degree k *on D is a pair $GHBH(D) = \langle P, S \rangle$ 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 $GHBH = \langle P, S \rangle$, the term $Nodes(GHBH)$ 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*.

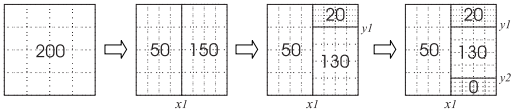


Fig. 2. A 4th degree *GHBH*

4.1 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 $sum(p_1), \ldots, sum(p_m)$: these sums are stored according to the order of the corresponding nodes in $String_P(GHBH)$. This array will be denoted as $String_S(GHBH)$ and consists in a sequence of m 32-bit words. 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.

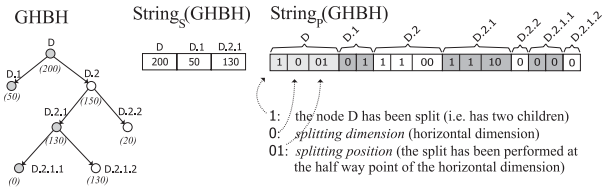


Fig. 3. Representing the *GHBH* of Fig. 2

In Fig. 3 non-derivable nodes are colored in grey, whereas derivable nodes are white. Derivable leaf nodes of *GHBH* (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 (see [3] for more details) that the maximum number of buckets of a *GHBH* built within the storage space bound B is given by:

$$\beta_{GHBH}^{max} = \left\lfloor \frac{B + log\ k + \lceil log\ d \rceil - 30}{3 + log\ k + \lceil log\ d \rceil} \right\rfloor$$

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 (see [3] for more details).
Comparing the formula expressing β_{GHBH}^{max} with that of β_{FBH}^{max} (see Section 3), we can draw the main conclusion that the physical representation scheme adopted for a *GHBH* enables a larger number of buckets to be stored w.r.t. an *FBH* within the same storage space bound. Moreover, it is worth noting that β_{GHBH}^{max} is much less affected by the increase of dimensionality w.r.t. β_{FBH}^{max}.

5 A Greedy Algorithm

As introduced in Section 1.1, one of the most important issues when dealing with multi-dimensional histograms is how to build the histogram which approximates “best” the

original data distribution, while being constrained to fit in a given bounded storage space. The SSE of a partition is a widely used metric to measure the "quality" of the approximation provided by histogram-based summary structures. The SSE of a histogram (based on an arbitrary partition) consisting in the buckets $\{b_1, \ldots, b_\beta\}$ is defined as $\sum_{i=1}^{\beta} SSE(b_i)$, where the SSE of a a single bucket is given by $SSE(b_i) = \sum_{\mathbf{j} \in b_i} (D[\mathbf{j}] - avg(b_i))^2$. Given a space bound B, the histogram which has minimum SSE among all histograms whose size is bounded by B is said to be *V-Optimal* (for the given space bound).
This notion of optimality can be trivially extended to histograms based on binary partitions. The SSE of a flat binary histogram *GHBH* is:

$$SSE(GHBH) = \sum_{b_i \in Buckets(GHBH)} SSE(b_i).$$

Thus, *GHBH* is V-Optimal (for a given space bound B) if it has minimum SSE w.r.t. all other flat binary histograms with space bound B.
In [3] we show that finding an optimal *GHBH* can be done in time polynomial w.r.t. the size of D, but the polynomial bound has been obtained using a dynamic programming approach, which is practically unfeasible, especially for large data distributions. In order to reach the goal of minimizing the SSE, in favor of simplicity and speed, we propose a greedy approach, accepting the possibility of not obtaining an optimal solution.
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. From the experiments shown in [3], where we compared several greedy criteria, 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, and splitting it at the position $< dim, pos >$ producing the maximum reduction of $SSE(b)$ (i.e. $SSE(b) - \left(SSE(b^{low}) + SSE(b^{high})\right)$ is maximum w.r.t. every possible split on b). All the other greedy strategies which have been compared with *Max-Var/Max-Red* are reported in [3].
The resulting algorithm scheme is shown below. It uses a priority queue 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

Let B be the storage space available for the summary.

begin

$q.initialize(\,)$; //the priority queue q is initialized;
$b_0 := \langle [1..n_1], \ldots, [1..n_d] \rangle$;
$H := new\ Histogram(b_0)$;
$B := B - 32 - 2$; // the space 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);
  // H is modified according to the split of b only
  //if there is enough storage space to perform the split;
  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.

It can be easily shown that the computation of $Evaluate(b)$ can be done in $O(Vol(b))$ (where $Vol(b)$ is the number of cells contained in b), as both the variance of b and all the reductions of $SSE(b)$ due to every possible split of b can be computed by accessing once the content of b.

Theorem 1. *Given a multi-dimensional data distribution D of size n^d and a natural k,* Greedy Algorithm *computes a GHBH of degree k on D in time $O\left(n^d \cdot log\ \beta_{GHBH}^{max}\right)$.*

Remark. The polynomial bound stated in Theorem 1 is of the same order of magnitude as those for MHIST [11] and MinSkew [1]. This means that the use of both a constrained partition scheme and a different criterion for splitting blocks does not affect the computational cost of building the histogram dramatically. Indeed, for a given storage space bound, the algorithm building a *GHBH* requires a larger number of steps to generate the histogram (w.r.t. MHIST and MinSkew algorithms), as it can produce a larger number of buckets, thus performing more splits. However, the aim of a *GHBH* is

not making the construction of a histogram more efficient, but to provide better accuracy in query answering. Due to space limitations, we do not provide our experimental results comparing *GHBH* with MHIST and MinSkew (the interested reader can find a complete discussion in [3]). From these experiments, it turns out that *GHBH* provide better performances (in terms of accuracy) w.r.t. the other techniques, due to the following reasons:

- the splitting criterion adopted by *GHBH* investigates the internal data distribution of buckets, rather than their marginal distributions (i.e. the projection of data on the dimension axes); in [3] it is shown how investigating only marginal distributions often makes locating dense regions of data (which should be associated to distinct buckets in order to achieve an accurate description of the original distribution) hard to perform, especially for high dimensionality data;
- the number of buckets generated by *GHBH* is much larger w.r.t. the other techniques, for a given storage space bound;
- the effectiveness of *GHBH* is related to the granularity of the grid: experiments show that partitions whose degree is "small" (w.r.t. the data domain size) suffice to obtain effective histograms. Although it is not possible to state the existence of a degree value which, regardless of the data distribution, yields the most effective histogram, our experiments show that the use of three bits to encode the degree of the *GHBH* leads to the most accurate estimates for a wide class of data distributions.

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. Furfaro, F., Mazzeo, M., Saccà, D., Sirangelo, C., Hierarchical Binary Histograms for Summarizing Multi-dimensional Data, ICAR-CNR Technical Report n.1, 2004, *available at* `http://wwwinfo.deis.unical.it/~sacca/recent_pub.html`
4. Garofalakis, M., Gibbons, P. B., Wavelet Synopses with Error Guarantees, *Proc. ACM SIGMOD 2002*, Madison (WI), USA.
5. Ioannidis, Y. E., Poosala, V., Balancing histogram optimality and practicality for query result size estimation, *Proc. SIGMOD 1995*, San Jose (CA), USA.
6. Jagadish, H. V., Jin, H., Ooi, B. C., Tan, K.-L., Global optimization of histograms, *Proc. SIGMOD Conf. 2001*, Santa Barbara (CA), USA.
7. Kooi, R.P., The optimization of queries in relational databases, PhD thesis, CWR University, 1980.
8. Korn, F., Johnson, T., Jagadish, H. V., Range Selectivity Estimation for Continuous Attributes, *Proc. SSDBM Conf. 1999*, Cleveland (OH), USA.
9. Mamoulis, N., Papadias, D., Selectivity Estimation Of Complex Spatial Queries, *Proc. SSTD 2001*, Redondo Beach (CA), USA.
10. Muthukrishnan, S., Poosala, V., Suel, T., On Rectangular Partitioning in Two Dimensions: Algorithms, Complexity and Applications, *Proc. ICDT 1999*, Jerusalem, Israel.
11. Poosala, V., Ioannidis, Y. E., Selectivity estimation without the attribute value independence assumption, *Proc. VLDB Conf. 1997*, Athens, Greece.
12. Vitter, J. S., Wang, M., Iyer, B., Data Cube Approximation and Histograms via Wavelets, *Proc. CIKM 1998*, Washington, USA.
13. Vitter, J. S., Wang, M., Approximate Computation of Multidimensional Aggregates of Sparse Data using Wavelets, *Proc. ACM SIGMOD Conf. 1999*, Philadelphia, PA, USA.