Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Hierarchical Binary Histograms for Summarizing Multi-dimensional Data

Filippo Furfaro, Giuseppe M. Mazzeo,
Domenico Saccà, Cristina Sirangelo

RT-ICAR-CS-04-01 **January 2004**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Hierarchical Binary Histograms for Summarizing Multi-dimensional Data

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

Rapporto Tecnico N.: **RT-ICAR-CS-04-01**

Data: **January 2004**

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Cosenza, Via P. Bucci 41C, 87036 Rende(CS), Italy
[2] Università degli Studi della Calabria, DEIS, Via P. Bucci 41C, Rende (CS), Italy

Hierarchical Binary Histograms for Summarizing Multi-dimensional Data

Filippo Furfaro, Giuseppe M. Mazzeo, Domenico Saccà and Cristina Sirangelo

D.E.I.S. - Università della Calabria
{furfaro, mazzeo, sacca, sirangelo}@si.deis.unical.it

Abstract

Two new classes of histogram-based summarization techniques which are very effective for multi-dimensional data are proposed. These classes are based on a particular binary hierarchical partition scheme, where blocks of data are recursively split into two sub-blocks, and use a tree structure for the representation of blocks and their summarized data. One of the two classes adopts a constrained partition scheme, where the position where a block is split must be laid onto a fixed, discrete grid defined on the block itself. The adoption of this constrained partitioning leads to a more efficient physical representation w.r.t. histograms using unrestricted partition schemes, so that the saved space can be invested to obtain finer grain blocks, which approximate data with more detail. The problem of constructing effective partitions is addressed as well, and several criteria for efficiently deciding where blocks have to be split are defined and compared. Experimental results show that our techniques yield smaller approximation errors w.r.t. traditional ones (classical multi-dimensional histograms as well as other types of summarization technique).

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 [11], selectivity estimation for query optimization in RDBMSs [2, 10], statistical and scientific data analysis, window query answering in spatial databases [1, 8], 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 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 new classes of multi-dimensional histograms which are based on binary hierarchical partitions. These summary structures are 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, *HBH* and *GHBH*) 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. In particular, *HBH*s (*Hierarchical Binary Histograms*) and *GHBH*s (*Grid Hierarchical Binary Histograms*) differ from one another as the latter are based on a con-

strained 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. *HBH* as well as 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.

We also address the problem of constructing optimal *HBH*s and *GHBH*s w.r.t. the well-known SSE metric [4], which measures the effectiveness of histograms on the basis of the uniformity of the distributions contained inside its buckets. We show that, unfortunately, computing the optimal solutions is too expensive, so we adopt heuristics based on greedy approaches to construct suboptimal partitions in a reasonable amount of time. We present a general greedy algorithm which is parametric w.r.t. the kinds of histograms (*HBH* or *GHBH*) and to various greedy criteria for deciding, at each partitioning step, which blocks have to be split and at which points.

Experimental results show that *GHBH* give much smaller approximation errors than *HBH* which, in turn, perform better than state-of-the-art summarization techniques, such as wavelets [11, 12] and other types of multidimensional histogram [1, 10]. Experiments also show that particularly *GHBH* and, to a lesser extent, *HBH* can be effectively applied on high-dimensionality data, as their accuracy is shown to be almost unaffected by the increase of dimensionality.

1.1 Related Work

Histograms were originally proposed in [6] 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. These contributions are evaluated by assuming that the frequency distribution inside each bucket is uniform, thus performing linear interpolation. This introduces some approximation error, but this error is often tolerated as an approximate evaluation of the result size of intermediate queries often suffices to compute an efficient query execution plan.

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 [10], 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 [7], spatial databases [1, 8] and OLAP [11]. 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 [9], 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* [4]) has been shown to be a NP-Hard problem, even in the two-dimensional

case [9]. 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* [10] 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) is described in more detail in Section 8.6. From the experiments in [10], 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, using a different criterion for choosing the bucket to be split at each step.
Other approaches to the problem of summarizing multidimensional data are the wavelet-based ones. Wavelets are mathematical transformations implementing a hierarchical decomposition of functions [3, 11, 12]. They were originally used in different research and application contexts (like image and signal processing), and have recently been applied to selectivity estimation [3] and to the approximation of OLAP range queries over data cubes [11, 12]. The compressed representation of a data distribution is obtained in two steps. First, a wavelet transformation is applied to the data distribution, and N wavelet coefficients are generated (the value of N depends both on the size of the data and on the particular type of wavelet transform used). Next, among these N coefficients, the $m < N$ most significant ones (i.e. the largest coefficients) are selected and stored. Issuing a query on the compressed representation of the data essentially corresponds to applying the inverse wavelet transform to the stored coefficients, and then aggregating the reconstructed (approximate) data values.

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 $\mathbf{x}$ 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 Figure 1.

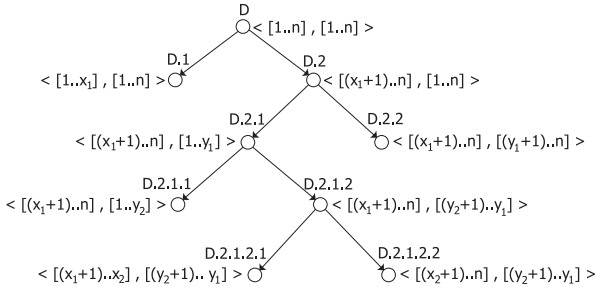


Figure 1: A binary partition

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 a flat binary histogram $FBH(D) = \{<b_1, sum(b_1)>, \ldots, <b_\beta, sum(b_\beta)>\}$, 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. 2 shows how the 2-dimensional flat binary histogram corresponding to the binary partition of Fig. 1 can be obtained by progressively performing binary splits on D. The histogram consists in the following set:
$\{ < \langle [1..x_1], [1..n] \rangle , 50 >, < \langle [x_1{+}1..n], [1..y_2] \rangle, 61 >,$
$< \langle [x_1{+}1..x_2], [y_2{+}1..y_1] \rangle, 0 >, < \langle [x_2{+}1..n], [y_2{+}1..y_1] \rangle, 63 >,$
$< \langle [x_1{+}1 \ldots n], [y_1{+}1 \ldots n] \rangle, 82 > \}$.

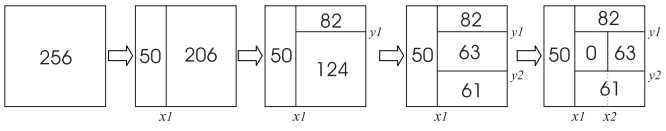


Figure 2: Constructing a 2D *FBH*

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 $\beta^{max}_{FBH} = \left\lfloor \frac{B}{32{\cdot}(2{\cdot}d{+}1)} \right\rfloor$

3.1 V-Optimal Flat Binary Histogram

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 *FBH* is $SSE(FBH) = \sum_{b_i \in Buckets(FBH)} SSE(b_i)$. Thus, *FBH* 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.

Theorem 1 *Given a d-dimensional data distribution D of size $O(n^d)$, the V-Optimal flat binary histogram $FBH^\star$ on D can be computed in $O(\frac{B^2}{d{\cdot}2^d} \cdot n^{2d+1})$*

Remark. Theorem 1 can be viewed as an extension of the results presented in [9], where the problem of finding the optimal binary hierarchical partition w.r.t several metrics (including the SSE) has been shown to be polynomial in the two-dimensional case [1]. We stress that this result cannot be extended to arbitrary partitions, where the problem of finding the V-Optimal histogram has been shown to be polynomial only in the one-dimensional case, and NP-hard even in the two-dimensional case [9].

4 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 Section 2.2 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 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.

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

In the following, given $HBH = \langle P, S \rangle$, the term $Nodes(HBH)$ will denote the set $Nodes(P)$, whereas $Buckets(HBH)$ will denote the set $Leaves(P)$.

4.1 Physical representation

A hierarchical binary histogram $HBH = \langle P, S \rangle$ 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. Therefore,

[1] Indeed [9] addresses the dual problem which is equivalent to finding the *FBH* which needs the smallest storage space and has a metric value below a given threshold

each non leaf node can be stored using a string of bits, having length $32 + \lceil log\ d \rceil + 1$, where 32 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 $String_P(HBH)$) 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 $m = |Nodes(HBH)|$) 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 $String_P(HBH)$. This array consists in a sequence of m 32-bit words and will be denoted as $String_S(HBH)$. 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 hierarchical binary histogram *HBH*, the set $Nodes(HBH)$ 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 $String_S(HBH)$ can be reduced to the representation of the sums of only non-derivable nodes.

On the right-hand side of Fig. 3 this representation paradigm is applied to the *HBH* shown on the left-hand side of the same figure.

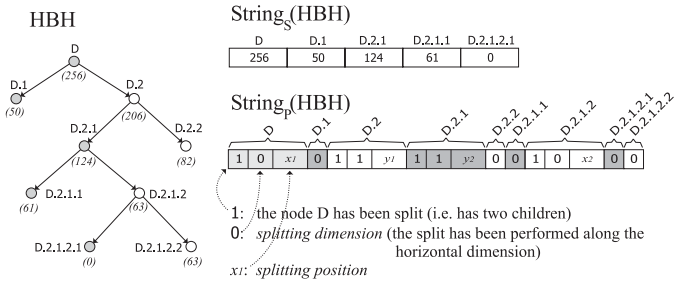


Figure 3: Representation of an *HBH*

In Fig. 3 non-derivable nodes are colored in grey, whereas derivable nodes are white. Leaf nodes of *HBH* are represented in $String_P(HBH)$ by means of a unique bit, with value 0. 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.

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 $String_S(HBH)$ and by employing one bit more for each node in $String_P(HBH)$ 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 *HBH*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 *HBH* 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 $String_P(HBH)$ non-leaf nodes do not need any additional bit either, since they cannot be null. According to this new representation model, each node in $String_P(HBH)$ is represented as follows:

- if the node is not a leaf it is represented using a string of length $32 + \lceil log\ d \rceil + 1$ bits, where 32 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 $String_S(HBH)$ represents the sum of all the non-derivable nodes which are not null.
A possible representation of the *HBH* shown on the left-hand side of Fig. 3 according to this new model is provided in Fig. 4. In particular, both non-leaf nodes and derivable leaf nodes are stored in the same way as in Fig. 3, whereas non-derivable leaf nodes are represented with a pair of bits. The first one of these has value 0 (which states that the node has not been split), and the second one is either 0 or 1 to indicate whether the node is null or not, respectively.

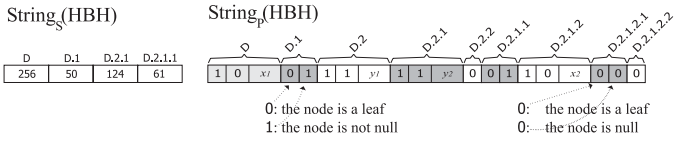


Figure 4: Efficient Representation of an *HBH*

According to the physical representation model presented above, it can be easily shown that maximum size of an *HBH* with β buckets is given by $\beta \cdot (67 + \lceil log\ d \rceil) - (34 + \lceil log\ d \rceil)$, which corresponds to the case that all but one leaf nodes are non-derivable, and all non-derivable nodes are not null.

We point out that the size of a hierarchical binary histogram *HBH* is less than the size of the "corresponding" flat binary histogram *FBH* having the same partition tree. In fact, the storage space needed to represent *FBH* with β buckets is $(2 \cdot d + 1) \cdot 32 \cdot \beta$, and $(2 \cdot d + 1) \cdot 32 > 67 + \lceil log\ d \rceil$ for any $d \geq 1$.

5 Grid Hierarchical Binary Histogram

In the previous section it has been shown how the exploitation of the hierarchical partition scheme underlying a histogram yields an effective benefit. That is, a hierarchical binary histogram can be represented more efficiently than the corresponding flat histogram, thus the available storage space can be used to represent a larger number of buckets.

We now introduce further constraints on the partition scheme adopted to define the boundaries of the buckets. The basic idea is that the use of a constrained partitioning enables a more efficient physical representation of the histogram w.r.t. histograms using more general partition schemes. The saved space can be invested to obtain finer grain blocks, which approximate data in more detail.

Basically, a Grid Hierarchical Binary Histogram *GHBH* is a hierarchical binary histogram whose internal nodes cannot be split at any position of any dimension: every split of a block is constrained to be laid 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. 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 4 *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 5 *Given a multi-dimensional array D, a* Grid Hierarchical Binary Histogram of degree k *on D is a hierarchical binary histogram $GHBH(D) = \langle P, S \rangle$ where P is a grid binary hierarchical partition of degree k on D.*

Fig. 5 shows an example of the construction of a two-dimensional 4th degree *GHBH*.

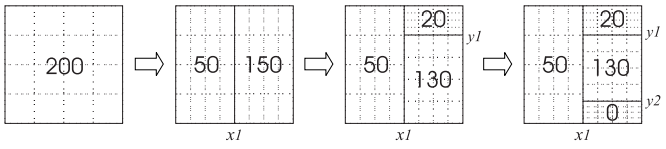


Figure 5: A 4th degree *GHBH*

Constraining each split of the partition to be laid onto a grid defined on the blocks of the histogram enables some storage space to be saved to represent the splitting position. In fact, for a grid binary partition of degree k, the splitting position can be stored using $\lceil log\ (k-1) \rceil$ bits, instead of 32 bits. 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 $log\ k$. Fig. 6 shows the representation of the grid hierarchical binary histogram of Fig. 5.

Proposition 1 *Given a multidimensional data distribution D and a space bound B, let HBH and GHBH be, respectively, a hierarchical and a grid hierarchical binary histogram on D within B. Then, the maximum number of buckets of HBH and GHBH is reported in Table 1.*

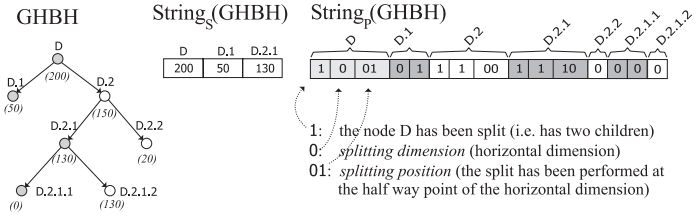


Figure 6: Representing the *GHBH* of Fig. 5

Histogram	***Maximum number of buckets***
HBH	$\beta^{max}_{HBH} = \left\lfloor \frac{B+\lceil log\ d \rceil + 2}{35+\lceil log\ d \rceil} \right\rfloor$
GHBH	$\beta^{max}_{GHBH} = \left\lfloor \frac{B+log\ k+\lceil log\ d \rceil - 30}{3+log\ k+\lceil log\ d \rceil} \right\rfloor$

Table 1

The bounds reported in the table above are computed by considering the case that the available storage space B is equal to the minimum storage space consumption of the *HBH* and the *GHBH* histogram (see Appendix for more details). Comparing the results summarized in Table 1 to the maximum number of buckets of an *FBH* β^{max}_{FBH} (see Section 3), the main conclusion that can be drawn is that the physical representation scheme adopted for an *HBH* permits us to store a larger number of buckets w.r.t. an *FBH* within the same storage space bound, as $35 + \lceil log\ d \rceil < 32 \cdot (2 \cdot d + 1)$. Analogously, the constraint on the splitting position of a *GHBH* further increases the number of buckets that can be represented within B, as we can assume that $32 > log\ k$.

As will be shown later, the main consequence of this is that *HBH* provides a more effective summarization of D than *FBH*, and, in turn, *GHBH* provides a more detailed partition than *HBH*.

6 Optimal Hierarchical Histograms

We extend to *HBH* and *GHBH* both the notion of V-Optimal histogram (introduced for *FBH* in Section 3), and the results related to its computational complexity.

The SSE of a hierarchical histogram $\mathcal{H}$ based on a binary partition (where $\mathcal{H}$ can be either *HBH* or *GHBH*) is $SSE(\mathcal{H}) = \sum_{b_i \in Buckets(\mathcal{H})} SSE(b_i)$. Thus, $\mathcal{H}$ is V-Optimal (for a given space bound B) if it has minimum SSE w.r.t. all other histograms of the same type (resp. *HBH*, *GHBH*) with space bound B.

Theorem 2 *Given a d-dimensional data distribution D of size $O(n^d)$, the V-Optimal histograms HBH* and GHBH* on D can be computed in the complexity bounds reported in the table below:*

Type of histogram	***Complexity bound of computing the V-Optimal histogram***
HBH*	$O(d \cdot \frac{B^2}{2^d} \cdot n^{2d+1})$
GHBH*	$O(d \cdot \frac{B^2}{2^d} \cdot k^{d+1} \cdot n^d)$

Remark. Comparing results in Theorem 2 to that of Theorem 1, we can observe that the computational complexity of constructing a V-Optimal *FBH* is less than that of computing a V-Optimal *HBH* within the same storage space bound. Essentially, this is due to the more complex representation scheme adopted by *HBH*, whose buckets are represented differently depending on whether they are null or not, derivable or not (see Appendix for more details). However, the two complexity bounds have the same polynomial degree w.r.t. the size of the input data; moreover the aim of introducing *HBH* is not to make the construction process faster, but to yield a more effective histogram. The complexity of building *GHBH** is less than that of *HBH** as, in the former case, the number of splits that can be applied to a block are constrained by the grid. Note that if $k = n$ the complexities of the two cases coincide.

7 Greedy algorithms

Although Theorem 2 states that finding optimal histograms *HBH*$^\star(D)$ and *GHBH*$^\star(D)$ can be done in time polynomial w.r.t. the size of D, the polynomial bound has been obtained using a dynamic programming approach (see proof of the theorem in Appendix), 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, in the case of a *GHBH*, 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. The greedy strategies tested in our experiments are reported in the table shown in Fig. 8. Criteria denoted as *marginal* (marg) investigate *marginal distributions* of blocks. The marginal distribution of a block b along the i-th dimension is the "projection" of the internal data distribution on the i-th dimension, and can be viewed as an array $marg_i(b)$ of size n_i. Formally, the j-th element of $marg_i(b)$ is the sum of all elements inside b whose i-th coordinate has value j. In the following, the term *marginal SSE* will be used to denote $SSE(marg_i(b))$ for some $i \in 1..d$. Fig 7 shows marginal distributions for a two-dimensional block.

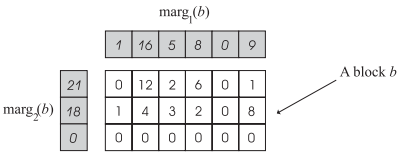


Figure 7: Marginal distributions

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 a measure of its need to be partitioned (denoted as *need*), and the position (dim, pos) of the most effective split, according to the adopted criterion. For instance, if *Max-Var/Max-Red* strategy is used, the function returns the SSE of b into *need*, and the splitting position which yields the largest reduction of SSE. Otherwise, if *Max-Red* criterion is adopted, the value of *need* returned by $Evaluate(b)$ is the maximum reduction of SSE which can be obtained by splitting b, and the pair $< dim, pos >$ defines the position corresponding to this split.

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);
    if $(B \geq 0)$
      $H := Append(H, b, b^{low}, b^{high})$;
      // H is modified according to the split of b only if
      // there is enough storage space to perform the split;
      $q.Insert\left(< b^{low}, Evaluate(b^{low}) >\right)$;
      $q.Insert\left(< b^{high}, Evaluate(b^{high}) >\right)$;
    end_if
  end_while
  return H;
end

Therein:

- the instruction $H := new\ Histogram(b_0)$ builds a hierarchical, or grid hierarchical, binary histogram 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.

The functions $MemUpdate$ and *Append* work differently depending on the type of histogram which is being built, as the space consumption of the splitting position is different in the two cases (32 bits for *HBH*, and $log\ k$

Criterion	The node b to be split, and the position $<dim, pos>$ where b is split
Max-Var/ Max-Red	the block b having maximum SSE is chosen, and split at the position $< dim, pos >$ producing the maximum reduction of $SSE(b)$ (i.e. $SSE(b) - \big(SSE(b^{low}) + SSE(b^{high})\big)$ is maximum w.r.t. every possible split on b)
Max-Var^{marg}/ Max-Red^{marg}	for each block, the marginal SSE along its dimensions are evaluated, and the block b having maximum marginal SSE is chosen (dim is the dimension s.t. $SSE(marg_{dim}(b))$ is maximum). Then, b is split at the position pos laying onto dim which yields the maximum reduction of $SSE(marg_{dim}(b))$ w.r.t. every possible split along dim
Max-Red	the strategy evaluates how much the SSE of every block is reduced by trying all possible splits. b and $< dim, pos >$ are the block and the position which correspond to the maximum reduction of SSE (i.e. $SSE(b) - \big(SSE(b^{low}) + SSE(b^{high})\big)$ is maximum w.r.t. every possible split on all non-split blocks of the histogram)
Max-Red^{marg}	the strategy tries all possible splits along every dimension of every block, and evaluates how much the marginal SSE (along the splitting dimension) is reduced by the split. b and $< dim, pos>$ 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

Figure 8: Splitting strategies

bits for *GHBH*). As regards the function *Evaluate*, in the case of *HBH*, the splitting positions to be evaluated and compared are all the positions between the boundaries of every dimension, whereas for *GHBH* the function computes only all possible splits laid onto the grid.

The computation of $Evaluate(b)$ can be accomplished more efficiently if the array F of *partial sums* and the array F^2 of *partial square sums* are available. Both F and F^2 have the same size as D and are defined as follows:
1) each element $F[i_1, \ldots, i_d]$ is the sum of all the values $D[j_1, \ldots, j_d]$ with $j_x \leq i_x$, for each $x \in [1..d]$ (i.e. $F[i_1, \ldots, i_d] = sum(\langle 1..i_1, \ldots, 1..i_d \rangle)$;
2) each element $F^2[i_1, \ldots, i_d]$ is the sum of all the values $(D[j_1, \ldots, j_d])^2$ with $j_x \leq i_x$, for each $x \in [1..d]$. It can be shown that using F and F^2 both the SSE of a block and the reduction of SSE due to a split can be computed in constant time. In Appendix it is also shown how the evaluation of the reduction of the marginal SSE along any dimension can be reduced to the computation of the reduction of the SSE. Obviously, the determination of the complexity of the proposed greedy algorithm when partial sums and partial square sums are used should take into account the cost of computing F and F^2, which is $O(2^d \cdot n^d)$.

Theorem 3 *The complexity of greedy algorithms computing, respectively, a hierarchical and a grid hierarchical binary histogram, in the cases that pre-computation of F and F^2 are either performed or not, are listed in the following table, for all the greedy criteria reported in Fig. 8:*

	No pre-computation	*Using pre-computation*	
		Cost of pre-computation	*Cost of computation*
HBH	$O\left(n^d \cdot \log \beta_{HBH}^{max}\right)$	$O\left(2^d n^d\right)$	$O\left(2^d \cdot d \cdot n \cdot \beta_{HBH}^{max}\right)$
GHBH	$O\left(n^d \cdot \log \beta_{GHBH}^{max}\right)$	$O\left(2^d n^d\right)$	$O\left(\left(2^d \cdot d \cdot \alpha + \log \beta_{GHBH}^{max}\right) \cdot \beta_{GHBH}^{max}\right)$

where $\alpha = n$ if the Max-Var^{marg}/Max-Red^{marg} *criterion is adopted, and $\alpha = k$ for all other greedy criteria.*

Remark 1. From Theorem 3 it follows that, if we do not use pre-computation, an *HBH* can be constructed faster than a *GHBH*, as $\beta_{HBH}^{max} < \beta_{GHBH}^{max}$ (see Proposition 1). However we point out that the aim of *GHBH* is not to make the histogram construction more efficient w.r.t. *HBH*, but to build more effective partitions of the data domain.

Remark 2. When pre-computation is used, the cost of constructing F and F^2 dominates the complexity of producing the histogram. We point out that, although the complexity bounds in the cases where no pre-computation is performed are of the same order of magnitude (w.r.t. the size of D) as the cases where F and F^2 are used, greedy algorithms performing pre-computation work much better in practice. Indeed, pre-computation becomes very difficult to manage for large data distributions, as F and F^2 are dense and, when their volume is "large", they cannot be represented in main memory.

Remark 3. When pre-computation is used, the cost of producing a *GHBH* (except the cost of constructing F and F^2) does not depend on the data size, for all greedy criteria other than Max-Var^{marg}/ Max-Red^{marg}. In fact, all the other greedy criteria need the computation of either the variance of a block (Max-Var/Max-Red), or all the possible reductions of variance (Max-Var/Max-Red and Max-Red), or all the possible reductions of marginal variance (Max-Red^{marg}). All these quantities can be computed in constant time by using F and F^2 (see the appendix). On the other hand, the Max-Var^{marg}/ Max-Red^{marg} criterion has to compute the marginal variances of candidate blocks: to accomplish this, all the marginal distributions of the blocks must be computed, thus introducing a $O(n)$ computational overhead w.r.t. the other cases (see the appendix).

8 Experimental Results

In this section we present some experimental results about the accuracy of estimating sum range queries on hierarchical histograms. First, experiments analyzing the effectiveness of the proposed greedy algorithms (based on the greedy criteria reported in the table of Fig. 8) are presented. Then, performances (in terms of accuracy) of *HBH*s and *GHBH*s are evaluated, and compared with

some state-of-the-art techniques in the context of multi-dimensional data summarization. Finally, our techniques are tested on high-dimensionality synthetic and real-life data.

8.1 Measuring approximation error

The exact answer to a sum query q_i will be denoted as S_i, and the estimated answer as $\tilde{S}_i$. The *absolute error* of the estimated answer to q_i is defined as: $e_i^{abs} = |S_i - \tilde{S}_i|$. 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 absolute error $\| e^{abs} \|$ and the average relative error $\| e^{rel} \|$ of the answers to the range queries belonging to the following query sets:

1. $QS^+(Vol)$: it contains the sum range queries defined on all the ranges of volume Vol whose actual answer *is not* null;
2. $QS^0(Vol)$: it contains the sum range queries defined on all the ranges of volume Vol whose actual answer *is* null.

8.2 Synthetic data

Our synthetic data are similar to those of [3, 12]. They are generated by creating an empty d-dimensional array D of size $n_1 \times \ldots \times n_d$, 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 centre 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, 12], data-sets generated by using this strategy well represent many classes of real-life distributions.

8.3 Real life data

Real-life data were obtained from the *U.S. Census Bureau* using their *DataFerret* application for retrieving data. The data source is the *Current Population Survey* (CPS), from which the *March Questionnaire Supplement (1994)* file was extracted. 9 attributes have been chosen. Our measure attribute is *Total Wage and Salary Amount*,and the 8 functional attributes are *Age*, *Parent's line number*, *Major Occupation*, *Marital Status*, *Race*, *Family Type*, *Public Assistance Type*, *School Enrollment*. The corresponding 8 dimensional array has about 143 million cells, and contains 14328 non-null values.

8.4 Comparing Greedy Criteria

Performances (in terms of accuracy) of our greedy algorithms adopting the proposed greedy criteria have been compared. Results for greedy algorithms finding an *HBH* are shown in Fig. 9. The diagrams in this figure are obtained on a data distribution of size $500 \times 500 \times 500$ having a density of 0.2%. In this figure, the accuracy of the various criteria is evaluated w.r.t. the *storage space* available for the compressed representation (fixing the skew z inside each region as equal to 1), and w.r.t. the *skew* inside each region (provided that the number of 32 bit words available is 5000), for queries whose volume is 0.5% of the data domain.

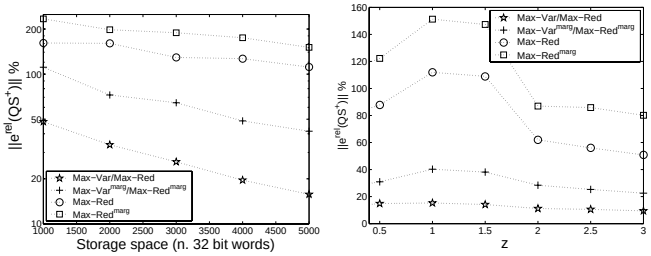
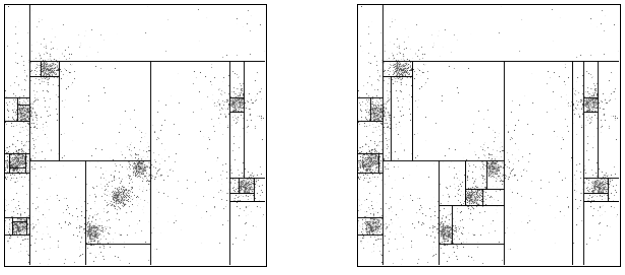


Figure 9: Comparing greedy criteria

The above diagrams show that the *Max-Var/Max-Red* criterion provides the best results, and, in particular, is less affected by the skewness of data. Interestingly, all the criteria are more effective in handling low and high levels of skew than intermediate ones ($z = 1.5$). When the skew is high, only a few values inside each region are very frequent, so that the dense regions contain mainly these values. Thus, the histogram groups these values into the same blocks causing small errors. Analogously, when the skew is small, the frequencies corresponding to different values are nearly the same and thus the data distribution is quite uniform, so that the CVA assumption generates small errors.
The fact that *Max-Red* gives worse performances than *Max-Var/Max-Red* can be motivated by observing that this criterion tends to progressively split "small" regions, as this often provides a greater reduction of the SSE metric. Fig. 10 shows the partitions obtained adopting, respectively, *Max-Red* and *Max-Var/Max-Red* on a two-dimensional data distribution with 9 dense regions, within the same storage space:



Max-Red Max-Var/Max-Red

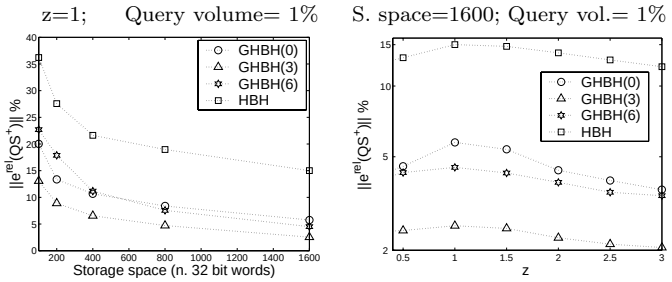
Figure 10: Two-dimensional partitioning

From Fig. 10 it emerges that the algorithm adopting *Max-Red* tends to split blocks belonging to the same dense region (yielding several small buckets), whereas *Max-Var/Max-Red* is "fairer" in selecting which region needs to be split.
Analogous results hold for *GHBH*: even in the case that splits are constrained to be laid onto a grid, the *Max-*

Var/Max-Red criterion provides the best accuracy. Diagrams are not shown for the sake of brevity.
Therefore, in the following only *HBH* and *GHBH* using the *Max-Var/Max-Red* criterion will be considered.

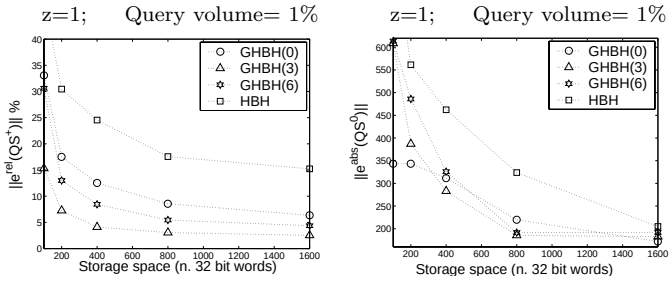
8.5 Comparing *HBH*s with *GHBH*s

These two classes of histogram have been compared on three-dimensional data distributions of size $200 \times 200 \times 200$ and $800 \times 800 \times 800$ having density 0.2% with $T = 5 \cdot 10^7$. In particular, 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.



(*a*) (*b*)

Figure 11: *HBH* vs *GHBH* on $200 \times 200 \times 200$ distr.



(*a*) (*b*)

Figure 12: *HBH* vs *GHBH* on $800 \times 800 \times 800$ distr.

From diagrams in Fig. 11 and Fig. 12, it emerges that *GHBH* algorithms perform better than *HBH* ones. This is due to the fact that, although the *HBH* algorithm is able to perform more effective splits at each step, the number of blocks generated by *GHBH* algorithms is much more. For instance, the *GHBH*(0) algorithm on average generates about two times the number of splits performed by the *HBH* algorithm within the same storage space bound. It is interesting to note that histograms generated by *GHBH*(0) are a little more sensitive w.r.t. data skewness than *GHBH* using finer grids (see Fig. 11(b)). This result is rather expected, as the constraint to split blocks into two equal size halves makes it necessary to define a lot of buckets to approximate a highly skewed region effectively. However, *GHBH*(0) counterbalances the rigidity of the partition scheme with a larger number of blocks which can be obtained w.r.t. the other techniques working with the same storage space bound.

From these results, 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*(2) and *GHBH*(3) (using binary splits of degree 4 and 8, respectively) give the best performances in terms of accuracy, and as the number of bits used to define the grid increases, the accuracy decreases. Moreover, the diagrams in Fig. 11 and Fig. 12 suggest that even if the size of data dimensions increase (from 200 to 800), the use of "few" bits (w.r.t. the size of the dimensions) for defining admissible splitting positions still give better performances (in terms of accuracy).
In the rest of the paper, all results on *GHBH* will be presented by using 3 bits for storing splitting positions.

8.6 Comparison with other techniques

We compared the effectiveness of *HBH* and *GHBH* algorithms with the state-of-the-art techniques for compressing multi-dimensional data. In particular, we analyzed the histogram-based techniques MHIST [10] and MinSkew [1], and with the wavelet-based techniques proposed respectively in [11] and [12]. 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 along every dimension 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 the $Max\text{-}Red^{marg}$ criterion to select the block to be split and where to split it. Indeed, MinSkew was introduced to deal with selectivity estimation in spatial databases (where 2D data need to be considered), so that it stores into each buckets a number of aggregate data which are useful in this context [1]. Our implementation is a straightforward ex-

tension of MinSkew to the multidimensional case, where each bucket stores only the sum of the contained data and the coordinates of the MBR.

Wavelet-based Compression Techniques. We have considered the two wavelet-based techniques presented in [12] (that will be referred as WAVE1) and in [11] (WAVE2). The former applies the wavelet transform directly on the source data, whereas the letter 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.

The diagrams of Fig. 13 are obtained on four-dimensional synthetic data of size $8 \times 32 \times 256 \times 2048$, with density 0.1% and z=1, whereas the diagrams of Fig. 14are obtained on real-life data. All the diagrams show that *GHBH* and *HBH* perform better than the other techniques.

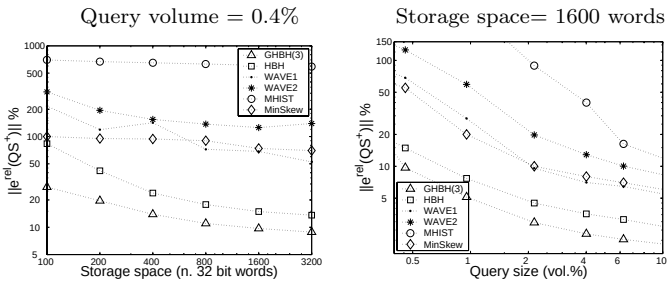


Figure 13: Comparing techniques (synthetic data)

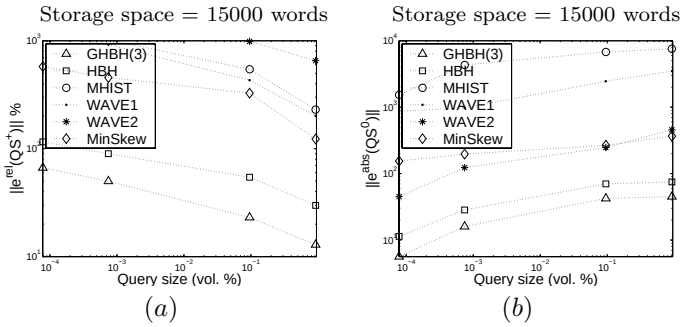


Figure 14: Comparing techniques (real-life data)

8.7 Sensitivity on dimensionality

We have tested the behavior of all the techniques when both synthetic and real-life data with increasing dimensionality are considered. Diagrams 15(a) and (b) refer to synthetic data. These diagrams were 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 15(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 15(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 two diagrams of Fig. 15 have been obtained by setting the compression ratio equal 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 show that the accuracy of every technique decreases as dimensionality increases, but *GHBH* and *HBH* get 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.

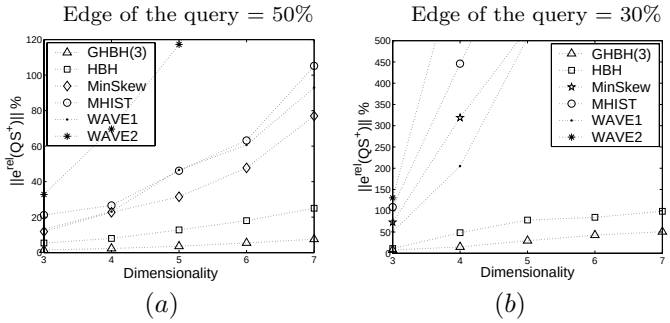


Figure 15: Sensitivity on dimensionality (synthetic data)

The same kind of experiments were performed on real-life data, yielding analogous results (Fig. 16).

Results shown in Fig. 16 can be motivated using the same conjectures as those used for synthetic data; in-

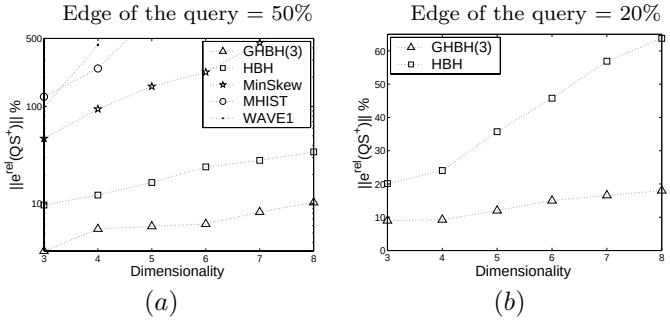


Figure 16: Sensitivity on dimensionality (real-life data)

deed, it often occurs that high-dimensionality real-life data come very close to the adopted synthetic model, as they are likely to consist in several clusters dispersed randomly in the data space.
Observe that *GHBH* and, to a lesser extent, *HBH* provide accurate answers even for "small" queries, where the other techniques are out of scale (Fig. 16(b)).

8.8 Remarks on experimental results

On the basis of the experimental results, we can draw the following conclusions:

1. the best performing greedy criterion among the considered ones is *Max-Var/Max-Red*: it produces more effective partitions and is not more costly (in terms of time complexity) w.r.t. the other criteria;
2. the physical representation scheme adopted for *HBH* introduces an effective improvement w.r.t. classical approaches based on binary partitions (MinSkew and MHIST): it enables us to produce more buckets within the same storage space, thus yielding more detailed partitions and better accuracy of the query estimates;
3. introducing a grid constraining the splitting position further enhances the effectiveness of the histogram: although splits in *GHBH* are less flexible than *HBH*, they can be represented more efficiently, and the space saved can be invested to perform further splits;
4. 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 two or three bits to encode the degree of the *GHBH* leads to the most accurate estimates for a wide class of data distributions;
5. *HBH* and *GHBH* are much less affected by the increase of dimensionality w.r.t. other techniques, and provide high accuracy even for queries with "small" size (w.r.t. the volume of data).

9 Conclusions

We have studied the use of binary hierarchical partitions as a basis for effective multi-dimensional histograms. We have introduced two new classes of histogram (namely, *HBH* and *GHBH*) which exploit their particular partition paradigm to make the representation of the histogram buckets more efficient w.r.t. the traditional "flat" representation scheme adopted for classical histograms. *HBH* and *GHBH* differ from one another in the type of split they allow on the data blocks. More precisely, in the partition underlying an *HBH* each block can be split at any position along any dimension, whereas 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 further improves the efficiency of the physical representation (w.r.t. *HBH*), enhancing the accuracy of estimating range queries on the histogram.
The problem of constructing optimal *HBH* and *GHBH* (w.r.t. the "classical" SSE metric) has been addressed too, as well as the issue of finding sub-optimal greedy solutions. Moreover we have provided several experimental results (on both synthetic and real-life data) 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.

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 2002*, Madison (WI), USA.

[4] Ioannidis, Y. E., Poosala, V., Balancing histogram optimality and practicality for query result size estimation, *Proc. SIGMOD 1995*, San Jose (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] Mamoulis, N., Papadias, D., Selectivity Estimation Of Complex Spatial Queries, *Proc. SSTD 2001*, Redondo Beach (CA), USA.

[9] Muthukrishnan, S., Poosala, V., Suel, T., On Rectangular Partitioning in Two Dimensions: Algorithms, Complexity and Applications, *Proc. ICDT 1999*, Jerusalem, Israel.

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

[11] Vitter, J. S., Wang, M., Iyer, B., Data Cube Approximation and Histograms via Wavelets, *Proc. CIKM 1998*, Washington, USA.

[12] Vitter, J. S., Wang, M., Approximate Computation of Multidimensional Aggregates of Sparse Data using Wavelets, *Proc. ACM SIGMOD Conf. 1999*, Philadelphia, PA, USA.

Appendix

Proposition 1 *Given a multidimensional data distribution D and a space bound B, let HBH and GHBH be, respectively, a hierarchical and a grid hierarchical binary histogram on D within B. Then, the maximum number of buckets of HBH and GHBH is reported in the following table:*

Histogram	Maximum number of buckets
HBH	$\beta_{HBH}^{max} = \left\lfloor \frac{B+\lceil log\ d \rceil+2}{35+\lceil log\ d \rceil} \right\rfloor$
GHBH	$\beta_{GHBH}^{max} = \left\lfloor \frac{B+log\ k+\lceil log\ d \rceil-30}{3+log\ k+\lceil log\ d \rceil} \right\rfloor$

Table 1

Proof. An *HBH* (as well as a *GHBH*) with β buckets has a space consumption which can vary between a minimum and a maximum value (depending on the partition tree and on the data distribution). We denote by $size_{HBH}^{min}(\beta)$ and $size_{GHBH}^{min}(\beta)$ the minimum space consumption of any *HBH* and, respectively, any *GHBH* having β buckets. There exists an *HBH* (resp. *GHBH*) with space bound B and β buckets if and only if $size_{HBH}^{min}(\beta) \leq B$ (resp. $size_{GHBH}^{min}(\beta) \leq B$). β_{HBH}^{max} and β_{GHBH}^{max} are obtained as the largest values of β which satisfy the latter inequalities. We will next compute $size_{HBH}^{min}$ and $size_{GHBH}^{min}$ as functions of β.
According to the physical representation of an *HBH* described in Section 4.1, the size of an *HBH* with β buckets can be expressed as the sum of four contributions: $size(HBH) = (2 \cdot \beta - 1) + (\beta - 1) \cdot (\lceil log\ d \rceil + 32) + ndl(HBH) + 32 \cdot ndn^{+}(HBH)$, where $ndl(HBH)$ and $ndn^{+}(HBH)$ stand for the number of non-derivable leaves of HBH and, respectively, the number of non-null non-derivable nodes of HBH. Analogously, we will denote by $ndl^{+}(HBH)$ and $ndl^{0}(HBH)$ the number of non-null non-derivable leaves and, respectively, the number of null derivable leaves of HBH. As $ndl(HBH) = ndl^{+}(HBH) + ndl^{0}(HBH)$ and $ndn^{+}(HBH) = \beta - ndl^{0}(HBH)$, then $size(HBH) = (2 \cdot \beta - 1) + (\beta - 1) \cdot (\lceil log\ d \rceil + 32) + 32 \cdot \beta + ndl^{+}(HBH) - 31 \cdot ndl^{0}(HBH)$. The latter expression has minimum value when $ndl^{+}(HBH) = 0$ and $ndl^{0}(HBH) = \beta - 1$, which occurs for an *HBH* with β buckets where all but one leaves are non-derivable and null. Analogously the size of a *GHBH* having β buckets is $size(GHBH) = (2 \cdot \beta - 1) + (\beta - 1) \cdot (\lceil log\ d \rceil + \log\ k) + 32 \cdot \beta + ndl^{+}(GHBH) - 31 \cdot ndl^{0}(GHBH)$. Thus the minimum storage consumption of an *HBH* and a *GHBH* having β buckets are, respectively: $size_{HBH}^{min}(\beta) = \beta \cdot (35 + \lceil log\ d \rceil) - \lceil log\ d \rceil - 2$ and $size_{GHBH}^{min}(\beta) = \beta \cdot (3 + \lceil log\ d \rceil + log\ k) - \lceil log\ d \rceil - log\ k + 30$. As said above, values of β_{HBH}^{max} and β_{GHBH}^{max} are straightforward.

Theorem 1 *Given a d-dimensional data distribution D of size $O(n^d)$, the V-Optimal flat binary histogram $FBH^{\star}$ on D can be computed in $O(\frac{B^2}{d \cdot 2^d} \cdot n^{2d+1})$.*

Proof. The problem of finding the V-Optimal *FBH* on D can be solved by the following dynamic programming approach. Given a block b of D, denoting the storage space needed to represent a single block as $\gamma = (2 \cdot d + 1) \cdot 32$, the minimum SSE of any *FBH* H on b with $size(H) \leq S$ can be defined recursively as follows:

1. $SSE^{*}(b, S) = \infty$, if $S < \gamma$;
2. $SSE^{*}(b, S) = SSE(b)$, if $S \geq \gamma \wedge (S < 2 \cdot \gamma \vee Volume(b) = 1)$;
3. $SSE^{*}(b, S) = min\{SSE^{*}(b^{low}, S1) + SSE^{*}(b^{high}, S2) |$ $\langle b^{low}, b^{high} \rangle$ *is a binary split on* b, $S1 > 0, S2 > 0, S1 + S2 = S\}$, otherwise

Our optimization problem consists in evaluating $SSE^{*}(D, B)$. As implied by the above recursive definition, $SSE^{*}(D, B)$ can be computed after evaluating $SSE^{*}(b, S)$ for each block b of D and each S in $[0..B]$ which is multiple of γ. At each step of the dynamic programming algorithm, $SSE^{*}(b, S)$ is evaluated by accessing $O(d \cdot n \cdot \frac{B}{d})$ values computed at the previous steps, as the possible binary splits of a block are $O(d \cdot n)$ and there are $O(\frac{B}{d})$ possible ways to divide S into two halves which are multiple of γ.
The number of different $SSE^{*}(b, S)$ to be computed are $O(\frac{B}{d} \cdot \frac{n^{2d}}{2^d})$, as the number of sub-blocks of D are $O(\frac{n^{2d}}{2^d})$, and the number of possible values of S are $O(\frac{B}{d})$.
On the other hand, the SSE of all the sub-blocks of D must be computed. It can be shown that the cost of accomplishing this task is dominated by $O(n^{2d})$. It follows that the overall cost of the dynamic programming algorithm is $O(\frac{B^2}{d \cdot 2^d} \cdot n^{2d+1})$.

Theorem 2 *Given a d-dimensional data distribution D of size $O(n^d)$, the V-Optimal histograms $HBH^{\star}$ and $GHBH^{\star}$ on D can be computed in the complexity bounds reported in the table below:.*

Type of histogram	Complexity bound of computing the V-Optimal histogram
$HBH^{\star}$	$O(d \cdot \frac{B^2}{2^d} \cdot n^{2d+1})$
$GHBH^{\star}$	$O(d \cdot \frac{B^2}{2^d} \cdot k^{d+1} \cdot n^{d})$

Proof.1. The problem of finding the V-optimal *HBH* can be formalized and solved following the same approach as the one just described for *FBH*s. The main difference is that when evaluating the optimal *HBH* on a block b, two distinct optimization problems must be addressed, corresponding to the cases that b appears in $HBH^{*}(D)$ as either a left-hand child or a right-hand child of some node. In fact, due to the physical representation paradigm (section 4.1), the storage consumption of $HBH(b)$ is different in these two cases. Intuitively enough, this leads to a recursive formulation of the V-optimal problem which is different from the one described for *FBH*s. We define the minimum SSE of any *HBH* H on b having $size(H) \leq S$ both in the case that b is considered as a left-hand child node (which we denote by $SSE_{left}^{*}(b, S)$) and a right-hand child node (which we denote by $SSE_{right}^{*}(b, S)$).

Both $SSE^*_{left}(b,S)$ and $SSE^*_{right}(b,S)$ can be defined recursively in a way that is similar to the recursive definition of $SSE^*(b,S)$ for *FBH*s. The main differences are that the non-recursive cases (i.e. the cases such that no *HBH* can be constructed or no split can be performed on b) express more complex conditions on the storage space (depending also on whether b is null or not). Moreover the recursive case is defined as the minimum value of $SSE^*_{left}(b^{low},S1) + SSE^*_{right}(b^{high},S2)$, for each possible binary split $< b^{low}, b^{high} >$ on b, and for each $S1$ and $S2$ which are consistent with the bound S on the overall space consumption allowed on b. The dynamic programming algorithm must compute both $SSE^*_{left}(b,S)$ and $SSE^*_{right}(b,S)$ for each sub-block of D and for each S in $[0..B]$. This algorithm computes $O(B \cdot \frac{n^{2d}}{2^d})$ values of $SSE^*_{left}(b,S)$ and $O(B \cdot \frac{n^{2d}}{2^d})$ values of $SSE^*_{right}(b,S)$, where each one is computed in time $O(d \cdot n \cdot B)$.

2. The problem of finding the V-Optimal *GHBH* of degree k can be formalized by means of some minor adaptation in the definition of $SSE^*_{left}(b,S)$ and $SSE^*_{right}(b,S)$ introduced for *HBH*s: 1) each constant which represents a storage space consumption is changed by replacing the 32 bits needed to represent the splitting position with $log\ k$ bits. 2) the minimum value of $SSE^*_{left}(b,S) + SSE^*_{right}(b,S)$ which define the recursive case is evaluated by considering only the binary splits of degree k.

The dynamic programming algorithm which computes all the values of $SSE^*_{left}(b,S)$ and $SSE^*_{right}(b,S)$ needed to compute $SSE^*_{left}(D,B)$ exhibits a different complexity bound as:

1. The cost of computing a single value of $SSE^*_{left}(b,S)$ or $SSE^*_{right}(b,S)$ is reduced to $O(d \cdot k \cdot B)$, since all the possible binary splits of degree k on a block are $d \cdot k$ (instead of $n \cdot k$).
2. Due to the restriction on the possible binary splits of a block, the recursive definition of $SSE^*(D,B)$ induces the computation of $SSE^*_{left}(b,S)$ or $SSE^*_{right}(b,S)$ for a proper subset of all the possible sub-blocks of D. It can be shown that the number of such blocks is $O(n^d \cdot \frac{k^d}{2^d})$ (instead of $O(\frac{n^{2d}}{2^d})$). Thus the number of values of $SSE^*_{left}(b,S)$ or $SSE^*_{right}(b,S)$ to be computed is $O(n^d \cdot \frac{k^d}{2^d})$ for each S in $[0..B]$.
3. The cost of computing the SSE of all the $O(n^d \cdot \frac{k^d}{2^d})$ blocks is $O(n^d \cdot k^d)$.

All considered, the cost of the dynamic programming algorithm which computes the V-Optimal *GHBH* of degree k on D is $O(d \cdot \frac{B^2}{2^d} \cdot k^{d+1} \cdot n^d)$. □

Theorem 3 *The complexity of greedy algorithms computing, respectively, a hierarchical and a grid hierarchical binary histogram, in the cases that pre-computation of F and F^2 are either performed or not, are listed in the following table, for all the greedy criteria reported in Fig. 8:*

	No pre-computation	*Using pre-computation*	
		Cost of pre-computation	*Cost of computation*
HBH	$O\left(n^d \cdot \log \beta^{max}_{HBH}\right)$	$O\left(2^d \cdot n^d\right)$	$O\left(2^d \cdot d \cdot n \cdot \beta^{max}_{HBH}\right)$
GHBH	$O\left(n^d \cdot \log \beta^{max}_{GHBH}\right)$	$O\left(2^d \cdot n^d\right)$	$O\left(\left(2^d \cdot d \cdot \alpha + \log \beta^{max}_{GHBH}\right) \cdot \beta^{max}_{GHBH}\right)$

where $\alpha = n$ if the Max-Var^{marg}/Max-Red^{marg} *criterion is adopted, and $\alpha = k$ for all other greedy criteria.*

Proof. The cost of the greedy algorithm is given by the sum of two contributions:

T^U: the cost of all the updates to the priority queue,

T^E: the cost of computing the function *Evaluate* for all the nodes to be inserted in the queue.

As to term T^U, at each iteration of the algorithm the first element of the priority queue is extracted and two new elements are inserted. The cost of either top-extraction and insertion is logarithmic w.r.t. the size of the queue, which is in turn bounded by the number of buckets of the output histogram. On the other hand, the number of iterations of the greedy algorithm is equal to the number of buckets it produces. Thus, if we denote as β the number of buckets of the histogram produced by the greedy algorithm, the overall cost T^U of the priority queue updates is $O(\beta \cdot log(\beta))$.

Moreover, if we denote as $T(Evaluate(b))$ the cost of computing the function *Evaluate* on the single block b, and we denote as H the binary histogram produced by the greedy algorithm, term T^E is given by $\sum_{b \in Nodes(H)} T(Evaluate(b))$.

We now discuss the complexity of computing the function $Evaluate(b)$ by distinguishing between the cases that pre-computation is either performed or not.

If pre-computation is not performed, the computational complexity of the function $Evaluate(b)$ is the same for all the proposed criteria. In fact, evaluating the SSE of a block b is trivially equivalent (in terms of complexity) to evaluating all the marginal SSE of b along its dimensions, as both these tasks can be performed by accessing once every element inside b. As regards the evaluation of the reduction of either $SSE(b)$ or $SSE(marg_{dim}(b))$ due to a split at $< dim, pos >$, these tasks can be accomplished in the same complexity bound. In fact, the following holds:

$$Red(b, dim, pos) = Red^{marg}(b, dim, pos)/P_{dim}$$

where: 1) $Red(b, dim, pos)$ and $Red^{marg}(b, dim, pos)$ are the reduction of $SSE(b)$ and, respectively, $SSE(marg_{dim}(b))$ due to the split of b along dim at position pos, 2) P_{dim} is the ratio between the volume of b and its size along the dimension dim. This result implies that the computation of the reductions of $SSE(b)$ corresponding to all the possible splits of b can be reduced to the computation of all the marginal distributions of the block (which can be performed by a linear

scanning of b), followed by the computation of the reduction of $SSE(marg_{dim}(b))$ along all the splitting points on dim, for each $dim = 1, .., d$ (which, on the whole, can be achieved by a linear scanning of all the marginal distributions).
Therefore, the function $Evaluate(b)$ implementing any of the criteria reported in Fig. 8 works in time linear in the size of b (in the case that no pre-computation is adopted).
We now consider the complexity of $Evaluate(b)$ in the case that pre-computation is performed before constructing the histogram. From the definition of SSE of a block, it holds that:
$SSE(b) = \sum_{\mathbf{i}\in b}(D[\mathbf{i}] - avg(b))^2 =$
$\sum_{\mathbf{i}\in b} D[\mathbf{i}]^2 - 2 \cdot \sum_{\mathbf{i}\in b} D[\mathbf{i}] \cdot avg(b) + \sum_{\mathbf{i}\in b}(avg(b))^2 =$
$\sum_{\mathbf{i}\in b} D[\mathbf{i}]^2 - 2 \cdot \frac{\left(\sum_{\mathbf{i}\in b} D[\mathbf{i}]\right)^2}{Volume(b)} + \left(\frac{\sum_{\mathbf{i}\in b} D[\mathbf{i}]}{Volume(b)}\right)^2 \cdot Volume(b) =$
$\sum_{\mathbf{i}\in b} D[\mathbf{i}]^2 - \frac{\left(\sum_{\mathbf{i}\in b} D[\mathbf{i}]\right)^2}{Volume(b)}$.

The terms $\sum_{\mathbf{i}\in b} D[\mathbf{i}]$ and $\sum_{\mathbf{i}\in b} D[\mathbf{i}]^2$ in the expression above can be evaluated as follows:

$$\sum_{\mathbf{i}\in b} D[\mathbf{i}] = \sum_{\mathbf{j}\in vrt(b)} (-1)^{\mathcal{C}(\mathbf{j},\mathbf{uv}(b))} \cdot F[\mathbf{j}]$$

$$\sum_{\mathbf{i}\in b} D[\mathbf{i}]^2 = \sum_{\mathbf{j}\in vrt(b)} (-1)^{\mathcal{C}(\mathbf{j},\mathbf{uv}(b))} \cdot F^2[\mathbf{j}]$$

In these expressions:
1) $vrt(b)$ is the set of vertices of b;
2) $\mathbf{uv}(b) = \langle ub(\rho_1), \ldots, ub(\rho_d)\rangle$ is the "upper" vertex of b;
3) $\mathcal{C}(\mathbf{i},\mathbf{j}) = \sum_{k=1}^{d} f(i_k, j_k)$, where: $f(a,b) = \begin{cases} 1, & a \neq b; \\ 0, & a = b. \end{cases}$

Therefore the SSE of a block can be evaluated accessing 2^d elements of F and 2^d elements of F^2, instead of accessing all the elements of the block. Clearly, also the reduction of $SSE(marg_{dim}(b))$ due to the split of b along any point on dim can be computed in $O(2^d)$, as it can be derived from the reduction of $SSE(b)$ due to the same split. On the contrary, evaluating $SSE(marg_{dim}(b))$ requires the computation and scanning of the marginal distribution of b along dim, which, using the array of partial sums, can be done in $O(2^d \cdot n)$. Therefore, for all the proposed greedy criteria but Max-Var^{marg}/ Max-Red^{marg}, in the case that pre-computation is used, $T(Evaluate(b)) = O(2^d \cdot \eta)$, where η is the number of reductions of SSE or marginal SSE which have to be computed. That is $\eta = d \cdot n$ for *HBH*, whereas $\eta = d \cdot k$ for *GHBH*.
In the case that Max-Var^{marg}/ Max-Red^{marg} is the adopted greedy criterion and pre-computation is used, the cost of computing the d marginal SSEs of the block is $O(2^d \cdot d \cdot n)$ for either *HBH* and *GHBH*, and dominates the cost of computing the reductions of marginal SSE.

To sum up, in the case that no pre-computation is used, $T(Evaluate(b)) = O(Volume(b))$, whereas, when pre-computation is adopted, $T(Evaluate(b)) = O(2^d \cdot d \cdot n)$ for *HBH*, and $T(Evaluate(b)) = O(2^d \cdot d \cdot \alpha)$ for *GHBH* (where $\alpha = k$ for all the greedy criteria but Max-Var^{marg}/ Max-Red^{marg}, for which $\alpha = n$).

Therefore the above defined term T^E (i.e. the cost of computing the function $Evaluate$ for all the blocks of the produced partition) gives different contributions to the cost of the greedy algorithm, according to the following three cases:

- in the case that no pre-computation is used T^E is $O(\sum_{b\in Nodes(H)} Volume(b))$, which is $O(n^d \cdot log(\beta))$ in the worst case (the produced binary partition is a complete binary tree).
- in the case that pre-computation is used, for *HBH*, T^E is $O(2^d \cdot d \cdot n \cdot \beta)$ (as $|Nodes(H)|$ is $O(\beta)$);
- in the case that pre-computation is used, for *GHBH*, T^E is $O(2^d \cdot d \cdot \alpha \cdot \beta)$, where α is defined above.

T^U (i.e. $O(\beta \cdot log(\beta))$ is always negligible w.r.t. T^E, for all the three cases above but the last one (*GHBH* with pre-computation) when $\alpha = k$.

Considering that, for each type of binary histogram, β is bounded by the expressions reported in Table 1, the overall cost of the greedy algorithm is straightforward for all the discussed cases.