UNIVERSITÀ DELLA CALABRIA

Dipartimento di ELETTRONICA, INFORMATICA E SISTEMISTICA

UNIVERSITÀ DELLA CALABRIA

Dipartimento di Elettronica,
Informatica e Sistemistica

Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica
XIX ciclo

Tesi di Dottorato

Effective Histogram-based Techniques for Summarizing Multi-dimensional Data

Giuseppe Massimiliano Mazzeo

UNIVERSITÀ DELLA CALABRIA
Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica

XIX ciclo

Tesi di Dottorato

Effective Histogram-based Techniques for Summarizing Multi-dimensional Data

Giuseppe Massimiliano Mazzeo

Coordinatore
Prof. Domenico Talia

Supervisore
Prof. Domenico Saccà

DIPARTIMENTO DI ELETTRONICA, INFORMATICA E SISTEMISTICA
Settore Scientifico Disciplinare: ING-INF/05

To my granny,
utmost model of
reliability and
devotion to work

Preface

Since databases (in any form) were born, the most important issues that have been capturing the interest of researchers are those related to the efficiency in information management. Hundreds of new research results and proposals are presented every year in the contexts of efficient data exploration and query execution. Optimizing these tasks can be effectively accomplished without accessing the data to be elaborated as long as it is somehow possible to estimate how they are distributed across their domain.

Several techniques for estimating data distributions have been proposed throughout the years. The very first ones based estimates on few parameters and several simplifying assumptions, such as uniform distribution of tuples among disk blocks (*uniform placement assumption*), uniform distribution of the tuples w.r.t. attribute values (*uniform frequency distribution assumption*), and independence between attribute value distributions (*attribute value independence assumption*). It was early proven that these assumptions are likely to lead to wrong estimates, which dramatically invalidate the optimization task. Therefore, in order to improve the estimation accuracy, several techniques for compactly representing the data distributions were proposed. All these techniques, even though based on many very different approaches, aim at summarizing the actual data distribution by means of lossy synopses.

The *data summarization* is often referred to also as *data compression* or *data reduction*. The basic idea is to represent the actual data distribution by a small number of synopses, which can be efficiently elaborated in order to obtain fast approximate answers to queries whose computation would result too slow if performed on the actual data. Of course, the use of synopses is effective only if the errors affecting the estimates based on them do not mislead the optimization task.

Some of the main application contexts to which the estimation techniques were targeted are the *block access estimation* for database design and query optimization, the *selectivity estimation* for query optimization, the *exploratory data analysis* in OLAP applications, the *statistical and scientific data analysis*, and the *window query answering* in spatial databases. In these application

contexts, efficiently aggregating data within specified ranges of their domain is such a crucial issue, that high accuracy in query answers becomes a secondary requirement.

For instance, query optimizers in relational DBMSs can build an effective query evaluation plan by estimating the selectivity of intermediate query results, which can be accomplished by retrieving aggregate information on the frequencies of attribute values. Obviously, the execution plan optimization for a given query is effective only if the overall time needed to compute the optimal query execution plan and to execute it is less than the time needed to execute the query without optimization: thus fast computation of aggregations is mandatory, as it determines the efficiency in computing the optimal query execution plan. Moreover a dramatic precision in evaluating aggregates is not needed, as knowing the order of magnitude of the size of intermediate query results suffices to build an effective execution plan.

Likewise, in decision support systems fast answers to range queries in *On-Line Analytical Processing* (OLAP) applications are mandatory to provide useful and timely reports. In this context pieces of information are represented as points in a multi-dimensional space, where dimensions define different perspectives for viewing data. This representation model is suitable to support data exploration, as users can navigate information and retrieve aggregate data by simply specifying the ranges of the data domain they are interested in. In fact *Decision Support Systems* (DSS) users are often concerned with performing preliminary explorations of the data domain, to find the regions where a more detailed analysis is needed. In this scenario, high accuracy in less relevant digits of aggregate-query answers is not needed, as providing their order of magnitude suffices to locate the regions of the database containing relevant information. At the same time, fast answers to these preliminary queries allow users to focus their explorations quickly and effectively, thus saving large amounts of system resources.

Moreover, there are other application contexts where the need to pose queries on summarized data arises from other issues than making query answering more efficient. For instance, summarizing data into appropriate synopses may be the only available option when database is remote, and a stationary connection with it is not guaranteed. In this case, if a dramatic precision of query answers is not required, users connected to the database can be equipped with synopses representing a summarized view of the remotely stored data, and issue queries on the synopses when disconnected from the DBMS. Obviously, the size of this summarized view must be much smaller than original data, especially when client-side storage space resources are very limited.

The data summarization techniques can be roughly classified into parametric and nonparametric techniques. The first ones assume that the data distribution to be summarized follows a known mathematical model, which can be either a statistical distribution or a polynomial function. Nonparametric techniques make no assumption about the actual data distribution,

and build synopses "letting the data speak for themselves". Histograms and sampling are examples of nonparametric summarization techniques.

Histogram-based techniques are those which received the major attention, and have been largely adopted in commercial relational DBMS systems, such as *DB2*, *Informix*, *Ingres*, *Microsoft SQL Server*, *Oracle*, *Sybase*, and *Teradata*. However, several different techniques, which are both theoretically and practically significant, have been proposed in literature.

A histogram over a multi-dimensional data distribution is generally built by partitioning the data domain into a number of hyper-rectangular blocks called *buckets*, and then storing summary information for each block. The answer of a range query is estimated on the histogram by aggregating the contributions of all buckets, without accessing the original data. That is, every bucket overlapping the query range is located and its contribution to the query answer is evaluated by performing suitable interpolation on the basis of its summary information.

As expected, on the one hand, querying the histogram rather than the underlying original data reduces the cost of evaluating answers (as the histogram size is much smaller than the original data size). On the other hand, the loss of information due to summarization introduces some approximation when queries are estimated on the histogram. Therefore, a crucial issue when dealing with histograms is finding the partition which provides the "best" accuracy in reconstructing query answers.

This problem has been widely studied by research community, and many techniques have been proposed for constructing histograms, which are very effective especially for one-dimensional data. Unfortunately, these methods do not scale up to high-dimensionality scenarios. That is, the strategies underlying most of the techniques working in the one-dimensional setting can be easily extended to the multi-dimensional case, but their performances (in terms of accuracy) worsen dramatically as dimensionality increases, till estimation errors become intolerable when dealing with high-dimensionality data. This is the effect of the well-known *curse of dimensionality*: as the number of dimensions increases, the number of buckets needed to achieve a satisfiable degree of accuracy requires a larger and larger amount of storage space for being stored. This means that no technique succeeds in constructing histograms yielding "reasonable" error rates within a "reasonable" space bound.

At the same time, no technique based on other approaches than histograms is known to provide satisfiable accuracy in the multi-dimensional scenario, so that finding an effective summarization technique for high-dimensionality data is still an open problem.

Main contributions of this thesis

This thesis is an effort towards the definition of effective summarization techniques for multi-dimensional data. The state-of-the-art techniques are exam-

ined and the issues related to their inability in effectively summarizing multi-dimensional data are pointed out. The attention is particularly focused on histogram-based summarization techniques, which are the most flexible, the most studied and the most adopted in commercial systems. In particular, hierarchical binary partitions are studied as a basis for effective multi-dimensional histograms, focusing the attention on two aspects which turn out to be crucial for histogram accuracy: the representation model and the strategy adopted for partitioning data into buckets. As regards the former, a very specific space-efficient representation model is proposed where bucket boundaries are represented implicitly by storing the partition tree. Histograms adopting this representation model (which will be said to be *Hierarchical Binary Histograms - HBH*) can store a larger number of buckets within a given amount of memory w.r.t. histograms using a "flat" explicit storage of bucket boundaries. On top of that, the introduction of a constraint on the hierarchical partition scheme is studied, allowing each bucket to be partitioned only by splits lying onto a regular grid defined on it: histograms adopting such a constrained partitioning paradigm will be said to be *Grid Hierarchical Binary Histograms* (*GHBH*). The introduction of the grid-constrained partitioning of *GHBH*s can be exploited to further enhance the physical representation efficiency of *HBH*s. As regards the construction of effective partitions, some new heuristics are introduced, guiding the data summarization by locating inhomogeneous regions of the data where a finer-grain partition is needed.

The two physical representation schemes adopted by *HBH* and *GHBH* can be viewed as a form of lossless compression to be used on top of the summarization accomplished by histograms (which is a form of lossy compression). The combination of these forms of compression are shown to result in a relevant improvement of histograms effectiveness. On the one hand, the proposed compression-based representation models provide a mechanism for efficiently locating the buckets involved in query estimation, thus reducing the amount of time needed to estimate queries w.r.t. traditional flat representation models. On the other hand, applying lossless compression on top of summarization reduces the loss of information due to summarization, as it enables a larger amount of summary data to be stored within a given storage space bound: this turns out to yield lower error rates of query estimates.

By means of experiments, a thorough analysis of different classes of histograms based on hierarchical partitions is provided: the accuracy provided by combining different heuristics (both the new proposals and the "classical" heuristics of two well-known techniques, namely *MHIST* and *Min-Skew*) with either the traditional *MBR*-based representation model or the novel specific tree-based ones (both the unconstrained and the grid-constrained one) is studied. These results provide an insight into the value of compression in the context of histograms based on hierarchical partitions. Interestingly, it is shown that the impact of both *HBH* and *GHBH* representation models on the accuracy of query estimates is not simply orthogonal to the adopted heuristic. Thus, the best combination of these different features is identified,

which turns out from adopting the grid-constrained hierarchical partitioning of *GHBH*s guided by one of the new heuristics.

GHBH is compared with state-of-the-art techniques (*MHIST*, *Min-Skew*, *GENHIST*, as well as other wavelet-based summarization approaches), showing that new technique results in much lower error rates and satisfiable degree of accuracy also at high-dimensionality scenarios.

Another important contribution of this thesis consists in the proposal of a new approach for constructing effective histograms. The superiority of *GHBH* w.r.t. the other histogram-based techniques has been found to depend primarily on the most accurate adopted criterion for guiding the data domain partitioning. In fact, traditional techniques for constructing histograms often yield partitions where dense and sparse regions are put together in the same bucket, thus yielding poor accuracy in estimating queries on summarized data.

Despite *GHBH* adopts a criterion which in theory avoid this situation, there is an intrinsic limit in all the top-down partitioning techniques. That is, histograms which are obtained by iteratively splitting blocks by starting from the block coinciding with the whole data domain, could not have the actual possibility to reach all the dense regions in order to isolate them. In fact, each split yields an increase in the number of bucket and, as the number of buckets is bounded, the number of split that can be performed is bounded as well. Therefore, in large domains, where data are particularly skewed, the number of available splits could not be large enough to reach in a top-down split-sequence all the dense regions. Thus, it could happen that *GHBH* starts the partitioning of data domain following the correct direction which leads to isolating dense regions, but at a certain point the number of available buckets, and thus of available splits, is saturated.

This problem could be avoided by adopting a bottom-up strategy, which first locates the dense region of the data, and then aggregates them into buckets according to some suitable strategy. The problem of searching dense regions is very close to the *data clustering* problem, that is the problem of grouping database objects into a set of meaningful classes.

The enhancement of the histogram construction has been tried by exploiting the capability of clustering techniques to locate dense regions. A new technique, namely *CHIST* (*Clustering-based Histograms*), for constructing multidimensional histograms on the basis of a well known density-based clustering algorithm, namely *DBSCAN*, is proposed. *CHIST* algorithm first invokes *DBSCAN* algorithm for partitioning the data into dense and sparse regions, and then further refines this partitioning by adopting a grid-based paradigm. *CHIST* is compared to *GHBH* and it is shown to provide lower error rates, especially in "critical" settings, that is when query selectivity is particularly low or the compression ratio is very high. It is worth remarking that in these settings, experiments comparing *GHBH* to the other techniques showed that *GHBH* still provides acceptable error rates, while those provided by other techniques are completely unacceptable. *CHIST* is also extended to the case that data to be summarized are dynamic. In this case, the re-execution of

the clustering algorithm at each data update could result prohibitive, due to the high computational cost of this task. Thus, on the basis of *Incremental DBSCAN* algorithm, a strategy for efficiently propagating data updates to the histogram is proposed. By means of experiments it is shown that the incremental approach, for small updates (i.e., bulk of updates 100 times smaller than the overall data size) can be computed from 100 to 200 times faster than the from-scratch re-computation of the histogram, and the accuracy remains almost unaffected.

Organization of this thesis

This thesis is organized as follows.

In Chapter 1 some application contexts which can benefit from the possibility of accurately approximating multi-dimensional data distributions are introduced, paying particular attention to the block access estimation problem, which is a basic problem both in query optimization and database design, to the selectivity estimation of range predicates, which is at the basis of query optimization, and to the data exploratory analysis in OLAP applications. Together with some other application contexts, these problems are showed to be a particular case of the general problem of efficiently and accurately estimating range queries over multi-dimensional data. The data summarization is addressed as the main approach for supporting the task of efficiently estimating range queries.

In Chapter 2 an overview of the data main summarization techniques is presented. In particular some parametric techniques based either on statistical models or analytical ones, such as the wavelet-transform-based techniques are described. As regards nonparametric techniques, several histogram-based summarization techniques are described. A brief overview of sampling-based techniques and of some hybrid techniques, combining some characteristic of both parametric and nonparametric techniques is presented as well.

In Chapter 3 a complete study of the histograms based on hierarchical binary space partitioning is presented. This class of histograms is very important as the majority of histogram-based techniques present in literature belongs to it. The main characteristics are investigated and then exploited to propose a very space-efficient tree-based representation model, which can be also exploited to efficiently compute range queries over the histogram. Then, it is shown how a constraint over the splitting position during the data domain partitioning can be exploited for further enhancing the space-efficiency of the histogram representation. As regards the histogram construction, first it is presented a optimal algorithm having polynomial complexity, which minimizes the weighted variance of the the histogram buckets, that is a commonly

adopted measure of the histogram quality. Then, a set heuristics which can be used to accomplish the greedy construction of the histogram is studied. Some of them are new, whereas others have been already adopted in other histogram-based techniques. Those heuristics are compared, showing that although they can be all applied with the same complexity, some of them, in particular a novel one, enable much more effective partitions of the data domain. The representation models are studied as well, and finally, the new technique, namely *GHBH*, is compared to some state-of-the-art techniques, such as *MHIST*, *Min-Skew*, *GENHIST*, as well as other wavelet-based summarization approaches. Finally, experiments studying the histogram construction efficiency, depending on the original data representation model and on the possibility of using some pre-computed structures, and other experiments studying the query execution times are presented.

In Chapter 4 a new approach for effective histogram construction, exploiting the cluster analysis, is proposed. The main idea on which this approach is based is that locating dense and sparse regions can be exploited to partition the data into homogeneous buckets, preventing dense and sparse regions from being summarized into the same aggregate data. The use of clustering techniques to support the histogram construction is investigated in the context of either static and dynamic data, where the use of incremental clustering strategies is mandatory due to the inefficiency of performing the clustering task from scratch at each data update. Experiments shows that these clustering-based histograms outperform the classical histogram-based summarization techniques, and the incremental approach make their adoption feasible even in those contexts where data are subject to updates.

Finally, the conclusions are presented and some promising issues to be developed in the future are addressed.

Acknoledgements

I consider myself very lucky for having had the opportunity to spend the years of my PhD course at the DEIS department of University of Calabria, where the organization is perfect and people are extraordinary both in professionalism and humanity. I wish to thank everybody for all the pleasant moments I spent at the department and for every single discussion which was helpful in my scientific ripening path and, not least, for their friendship. In particular, I wish to thank Prof. Domenico Saccà, my supervisor, for his continuous flow of suggestions (not always easy to be understood at once, but definitely always fruitful), and Prof. Domenico Talia for his careful coordination of the PhD course.

I wish to thank Prof. Sergio Greco for his excellent direction of the department and for his kind attention to the needs of us PhD students.

My biggest acknowledgement goes to my friend Filippo Furfaro, who introduced me to the world of research and has continuously been my heedful guide, trying to keep me away from several *distrauctions*[1].

I wish to thank very much Cristina Sirangelo, for having always been a dear friend and for having shared with me the most interesting topics of my studies. Thank you to Prof. Sergio Flesca, for always reminding me that fine thinking and colorful manners are not necessarily incompatible, Elio Masciari, for being a model of dynamism (which unfortunately I am not able to follow), and Andrea Pugliese, for the funny comments we exchanged about several linguistic oddness.

Thank you very much to the department staff, in particular to Giovanni Costabile and Francesco De Marte, for their invaluable technical support.

And last but not least, I wish to thank my dear chum Francesco Parisi, for reminding me every morning the day of the week together with some other "useful information" (especially on Wednesdays).

Rende, November 2006 *Giuseppe Massimiliano Mazzeo*

[1] neologism denoting interest in an auction, which takes the attention off something more important. The word is particularly appropriate for online auctions.

Contents

Preface v

1 Data summarization: application contexts 1
1.1 Block access estimation in database performance analysis 2
1.2 Selectivity estimation in query optimization 5
1.3 Exploratory data analysis in OLAP applications 12
1.4 Estimating data distributions in other contexts 15
1.5 Generalization of the data distribution estimation problem 17
1.5.1 Basic notations 18

2 Data summarization: state-of-the-art techniques 21
2.1 Mathematical models 22
2.2 Discrete Wavelet Transform 28
2.2.1 Basic mathematical background 28
2.2.2 Fast Wavelet Decomposition 30
2.2.3 Querying wavelet coefficients 32
2.2.4 Wavelet coefficients thresholding 34
2.2.5 Multi-dimensional DWT 34
2.2.6 Wavelet-based summarization techniques 36
2.3 Histograms 40
2.3.1 Querying histograms 42
2.3.2 A Histogram Taxonomy 45
2.3.3 Multi-dimensional Histograms 46
2.3.4 Histogram-based summarization techniques 48
2.3.5 Non-classical Histograms 52
2.4 Sampling 58
2.5 Histograms vs. other techniques 62

3 Hierarchical Binary Histograms 65
3.1 Hierarchical Binary Partitions 66
3.2 Flat Binary Histograms 66

3.3 Hierarchical Binary Histograms 68
3.4 Grid Hierarchical Binary Histograms 72
3.5 Spatial efficiency of representation models 74
3.6 Constructing V-Optimal histogram 77
3.7 Greedy algorithms for histogram construction 80
3.7.1 Greedy criteria evaluation for *FBH* and *HBH* construction 84
3.7.2 Greedy criteria evaluation for *GHBH* construction 86
3.7.3 Using pre-computation for evaluating greedy criteria 87
3.7.4 Complexity of Greedy Algorithm 88
3.7.5 Workspace size for Greedy Algorithm 90
3.8 Estimating range queries 91
3.9 Experimental analysis 94
3.9.1 Synthetic data 94
3.9.2 Real life data 95
3.9.3 Experimental plan 95
3.9.4 Comparing *FBH* and *HBH* under different greedy criteria 96
3.9.5 Comparing *HBH* with *GHBH* 101
3.9.6 *GHBH* versus other techniques 105
3.9.7 Query estimation times 110
3.9.8 Histogram construction times 112

4 Clustering-based Histograms 115
4.1 *CHIST*: Clustering-based Histogram 117
4.1.1 Step I: clustering data 117
4.1.2 Step II: summarizing data into buckets 118
4.1.3 Step III: representation of the histogram 121
4.2 Incremental maintenance of *CHIST* 126
4.2.1 Step I: incremental clustering 127
4.2.2 Step II: storage space distribution among layers and partitioning 130
4.2.3 Step III: rearrangement of buckets 133
4.3 Costs of the non-incremental and incremental approaches 133
4.4 Experimental analysis 136
4.4.1 Comparing *CHIST* with *GHBH* 136
4.4.2 Efficiency of the incremental approach 137

Conclusions 141

References 147

Index 157

1

Data summarization: application contexts

The problem of data access optimization has always been one of the most central topics in computer science. Indeed, this problem addresses a large class of issues, such as the data modelling, both at logical and physical level, and the information retrieval techniques. Studies related to the definition of new logical data models, the organization of disk resident data and the query optimization have always been and continue to be topical issues. Researches related to these topics have always been strongly encouraged by the wider and wider spreading of the automatic management of information in all the every-day-life activities, which in turn has always been encouraged by the advances in technologies owing to the results of the researches themselves.

Larger and larger amounts of information require hardware systems with larger and larger computational power, which, indeed, are still improving their performances according to Moore's law [102], conjecturing that the transistor density of integrated circuits, and thus the chip computational power, with respect to minimum component cost, roughly doubles every two years.

However, the advances in hardware performances could be not enough to guarantee efficient management of larger and larger amounts of available information (the capacity of hard drives, and thus the amount of stored data, is subject to the same exponential growth), if they were not supported by efficient information-representation paradigms and efficient techniques for elaborating it. To this aim, since computer science was born, several database models have been proposed. The *relational* database model, introduced in 1970 by Codd [32], has been the most successful among all the ones proposed throughout the years (e.g., hierarchical, network, and object-oriented).

Whatever is is the model adopted by a database designer, the physical database design is a crucial activity, as it mainly determines the performances of information management. Furthermore, as the most commonly adopted language in relational database management systems (RDBMS) is *SQL* [18], which is a declarative language, instead of a procedural one, queries posed to the database could require to be optimized, in order to be efficiently executed.

Both the problems of physical database design and query optimization are based on cost models which describe the database performances [16, 94, 95, 100, 119, 133]. Even if those models may include several parameters, the number of accessed pages in evaluating a queries has widely been considered the most important one. Being this parameter an unknown variable, estimating the number of accessed blocks for a given query was one of the most studied problem in 1970s and in the early 1980s.

In the following section a brief overview of the studies related to the *block access estimation* problem will be presented. Then, the *selectivity estimation* problem, which is strictly related to the first one will be introduced. Subsequently, the problem of estimating data distribution in OLAP and some other application contexts will be briefly discussed. Finally, all these problems will be generalized into the problem of estimating efficiently and accurately range queries on multi-dimensional data distributions, and basic notations for the rest of the thesis will be provided.

1.1 Block access estimation in database performance analysis

Let R be a relation containing n tuples stored in m blocks (pages). If a query entails retrieving k tuples from R, what is the expected number of blocks that will be accessed?
The problem above is known as the *block access estimation* problem and, intuitively enough, it has a wide importance in the database performance related studies.

At the beginning, the problem was studied under the assumption that any of the m pages can be accessed with the same probability at any of the k retrievals. This assumption implies that data are unstructured (in a structured file, any access is somehow related to the previous ones).

In [113] Rothnie and Lonzano showed that, in an unstructured file, the number of block accesses grows almost linearly with the number k of tuples retrieved. Therefore, the use of secondary indexes becomes less and less effective as the number k of tuples retrieved approaches the number m of pages.

A first computation of the expected number $X(k, m)$ of accessed blocks was proposed by Cardenas in 1975 [17]. He stated that

$$X(m,k) = m \cdot \left[1 - \left(1 - \frac{1}{m}\right)^k\right] \tag{1.1}$$

This formula can be easily obtained considering that when retrieving one tuple the probability to access the j-th block is $\frac{1}{m}$, then the probability of not accessing the j-th block is $1 - \frac{1}{m}$. Cardenas, further assumed that the k retrievals are independent one another, therefore the probability of accessing the j-th block in none of the k retrievals is $(1 - \frac{1}{m})^k$. Thus, the probability

of accessing the j-th block almost once is $I_j = 1 - (1 - \frac{1}{m})^k$. Therefore, the expected number of accessed blocks is $\sum_{j=1}^{m} I_j$, which gives Equation 1.1.

Two years later, in [133], Yao observed that the assumption that the k records are selected independently implies the selection with replacement, that is a record can be selected more than once. Therefore, he pointed out that Cardenas' formula refers to the case of *selection with replacement*. Yao, instead, studied the problem imposing that the k tuples are distinct. He showed that the probability of selecting the j-th block, in the case of *selection without replacement*, is $I_j = 1 - C_k^{n \cdot d}/C_k^n$, where $d = 1 - \frac{1}{m}$ (thus, $n \cdot d$ is the number of tuples outside the j-th block). Therefore, the expected number of accessed blocks is $Y(n, m, k) = m \cdot [1 - C_k^{n \cdot d}/C_k^n]$. The formula was given by Yao in the following form:

$$Y(n, m, k) = m \cdot \left[1 - \prod_{i=1}^{k} \frac{n \cdot d - i + 1}{n - i + 1}\right]. \tag{1.2}$$

Yao remarked that the formula proposed by Cardenas is a lower bound of his formula and gives good approximation when $k \ll n$ and $m \ll n$.

In [26], Cheung observed that another assumption, besides the selection with replacement as observed by Yao, is necessary so that Equation 1.1 is correct: results containing the same records have to be considered different if the records are chosen in different order. Cheung showed that when the k tuples are considered as a set, the probability of accessing j-th block in no one of the k retrievals is $I_j = 1 - C_k^{n \cdot d + k}/C_k^{n+k-1}$, from which, the correct formula for $X(m, k)$ is the following

$$X(m, k) = m \cdot \left[1 - \prod_{i=1}^{k} \frac{n - p + i - 1}{n + i - 1}\right] \tag{1.3}$$

In Cheung's work the expected number of distinct records was also shown to be $\frac{nk}{n+k-1}$, where n is the maximum number of possible distinct tuples (the domain size of the projection attribute) and k is the number of retrieved tuples.

A closed formula which approximate Equation 1.2 was introduced in 1983 by Whang and Wiederhold [131]. The error was studied analytically and it was shown that it can not be larger than 3%. In [92] Luk studied the problem in the case that the number of retrieved records and block size are not fixed, but they are represented by stochastic variables. An interesting result presented in [92] is that, given a set $K = \{k_1, k_2, \ldots, k_s\}$ of possible values for k, $Y(n, m, \bar{k}) = Y(n, m, \frac{1}{s} \cdot \sum_{i=1}^{s} k_i)$ is very often greater than $\bar{Y}(n, m, k) = \frac{1}{s} \cdot \sum_{i=1}^{s} Y(n, m, k_i)$. Luk showed that if $Y(n, m, \bar{k}) < \bar{Y}(n, m, k)$ the difference is negligible, and when the average number of tuples per pages n/m is not small (e.g., greater than 10), it always holds that $Y(n, m, \bar{k}) > \bar{Y}(n, m, k)$. If the values in K have small variance the approximation $Y(n, m, \bar{k}) \approx \bar{Y}(n, m, k)$ is good. For a normal distribution of values in

K, by means of experiments, Luk showed that $Y(n, m, \bar{k})$ is from 10% to 20% greater than $\bar{Y}(n, m, k)$ when the maximum value in K is between 2 times and 10 times m. In this cases, he suggested to subtract the 10% from $Y(n, m, \bar{k})$, in order to have and approximation of $\bar{Y}(n, m, k)$ within the 10%. Even if the study of Luk was restricted to some particular cases, it clearly showed that when a database is designed on the basis of uniform distribution assumption of the result size of the queries, the expected number of block accesses is likely to be overestimated, thus guiding the database designer to wrong decision in choices about index designing.

In [29] Christodoulakis showed that a generalization of Cardenas' formula (Equation 1.1), where the number of records per block is not constant, is a *Shur concave function*, and by exploiting the theory of majorization [98], he proved that the assumption of uniform distribution of records among blocks often leads to an overestimate of the actual number of block accesses. In [31] he proved that also the assumptions of uniformity and independence of attribute values in database relations may lead to large errors in database performance estimation, when the assumptions are not satisfied. Specifically, uniform and independence assumption were analytically proved to represent an upper bound of the expected number of accessed blocks during a query evaluation. Furthermore, the author interestingly argued that the uniform query distribution is a pessimistic assumption as well. In fact, according to the author, the user is more often interested in cases specified by restricting conditions, which involves only few records, rather than usual cases.

Indeed, the idea that not all records are not accessed with the same frequency dates back to 1963. In [61] Heising conjectured that Pareto's principle, proposed by American industrial engineer Juran in 1941 and stating the about the 80% of consequences stems from about the 20% of causes[1], applies also to the most commercial computer science applications, so that the the 80% of the transactions runs over the 20% most accessed records in a file.

After proving that the assumption of uniform distribution of number of tuples inside blocks leads to wrong estimations, in [30] Christodoulakis proposed to model the distribution of tuples inside disk blocks w.r.t. attribute values by means of multivariate distributions based on Pearson and normal families. Thus, the expected number of accessed disk blocks can be evaluated on the basis of the statistical model.

In 1986 Vander Zanden et al. proposed an approximate fast computation of the expected number of accessed blocks in retrieving k tuples when records are not uniformly distributed among blocks [124]. Being p_i the number of tuples in the i-th block ($1 \leq i \leq m$), they proposed to order the blocks according to decreasing values of p_i and to approximate the distribution of p_i values by partitioning the ordered sequence of p_i into m' ranges and approximating p_i

[1] Italian economist Vilfredo Pareto in 1906 observed that the 80% of wealth in Italy was owned by the 20% of population, and Juran later observed that the same situation holds is several fields

values inside the same range by means of their average. The, the number of accessed pages can be estimated in time $\mathcal{O}(m')$. By exploiting the theory of majorization [98], they showed that the estimation made on the histogram is an upper bound of the expected number of accessed blocks. Accuracy could be increased by increasing the number of intervals, which means lowering the upper bound. In the case that one only interval is defined, the expected value coincides with that of Cardenas' formula (Equation 1.1). The criterion proposed to group block densities together was based on a recursive splitting of the vector representing the ordered block densities (i.e., number of tuples per block) into two sub-vectors: one containing values larger than the average and the other less.

Studies related to the block access problem didn't continue in the direction based on a more accurate approximation of the data distribution inside blocks. Indeed, the researchers' interest, since the early 1980s, had been moving towards another topic, strictly related to the block access estimation problem, namely the selectivity estimation. While the block access estimation is related to data distribution estimation at the physical level (number of tuples per disk block), the selectivity estimation is based on data distribution at the logical level, that is, the interest is on how data are distributes within the data domain. This problem will be discussed in the following section.

1.2 Selectivity estimation in query optimization

The interest in the problem of query optimization has significantly increased since declarative languages, such as *SQL*, became the most prominent ones in retrieving information form databases. A declarative-language-based query evaluator module of a DMBS consists of two main components: the *query optimizer* and the *query execution engine*. The query optimizer transforms the queries posed by the users into a sequence of physical operators that are implemented by the query execution engine. The presence of the query optimizer is mandatory as the user submits a query in an high-level language, describing by means of conditions the information in which s/he interested, and there are usually several possibilities to translate the high-level specifications into sequences of low-level operators. Each possible sequence of low-level operator for evaluating a query is called *evaluation plan* or *access plan*. The choice of the evaluation plan can affect very significantly the efficiency of the query evaluation task. Even though in order to choose the most efficient evaluation plan the query optimizer should take into account several factors, the number of disk accesses has been widely adopted as a good approximation of the actual execution cost.

One of the most simple kind of queries to be optimized is the selection of tuples from a single relation, by specifying some equality or inequality conditions on a subset of the relation attributes w.r.t. values of the attributes domain.

This set of queries can be represented by the following SQL query:

```
SELECT *
FROM R
WHERE cond_1(R.X_1) AND cond_2(R.X_2) AND ... AND cond_d(R.X_d);
```

where R is a relation with at least d attributes, namely $X_1, X_2, \ldots, X_d$, each defined on an ordered discrete domain $D_i = [inf_i..sup_i]$, and $cond_i$ is a simple range selection condition defined on $R.X_i$ of the form $l_i \leq X_i \leq h_i$. Observe that this condition incudes the cases of equality ($inf_i = sup_i$) and one-side inequality conditions ($l_i < inf_i$ or $h_i > sup_i$).

In this query the crucial choice which can affect query execution efficiency is related to the indexes: accessing data by means of an index, could improve the efficiency but it requires additional accesses to the index data structure itself. Thus, in evaluating this query it must be chosen which index have to be exploited[2].

Assuming that secondary indexes are available on d' of the d attributes, there are $2^{d'}$ possible query evaluation plans, depending on whether each one of the d' available indexes is adopted or not. When d" indexes are chosen to be accessed, that is the indexes defined on $X_{i_1}, X_{i_2}, \ldots, X_{i_{d"}}$, the query execution engine will access each one of these d" indexes, and for the j-th selected index it will find the set P_j of pages containing the tuples satisfying $cond_{i_j}$. Then, the intersection $P = P_1 \cap P_2 \cap \ldots \cap P_{d"}$ will be computed and the pages in P will be accessed in order to select the qualifying tuples (i.e., the tuples satisfying all the d conditions). Of course, the case in which $d" = 0$, i.e. the query is executed without accessing any index, the query engine will scan the entire relation in order to select the qualifying tuples.

In order to select the optimal execution plan, the number of both the accessed index and data pages has to be estimated, for each possible query plan. As the number of needed estimations could become very large as d' increases, each estimation must be performed very quickly, otherwise the aim of the query optimization would be invalidated.

The cost of a query plan clearly depends on the cost of accessing all the indexes exploited by the query plan the number of data pages to be accessed. The estimation of the number of accessed index pages is straightforward, as the tuples are ordered inside the index pages according to the index key. The number of pages accessed, in the case of a tree-based index, such as a B$^+$-Tree [33], is $\lceil \log_{\lfloor n/m_i \rfloor} n \rceil + \lceil m_i \cdot k_i / n \rceil$, where n is the number of indexed tuples, m_i is the number of pages spanned by the index on X_i and k_i is the

[2] The problem of index choice is crucial also in the database design [114]: the adoption of an index, can make query execution more efficient, but require additional costs to propagate changes in data to it. Thus, in designing a database it must be chosen which/how index have to be implemented. In [106] Piatetsky-Shapiro proved that the optimal choice of secondary keys an $\mathcal{NP}$-hard problem.

number of tuples satisfying $cond_i$. The estimation of the number of accessed data pages can be accomplished by adopting Yao's formula (Equation 1.2), or the formula proposed by Whang and Wiederhold for a faster approximate evaluation [131], on the basis of the number n of tuples contained in the relation R, the number m of data pages spanned by R and the number k of tuples satisfying the conditions $cond_{i_1}, cond_{i_2}, \ldots, cond_{i_{d'}}$ jointly[3].

Observe that, for each query plan the estimations of both the number of accessed index and data pages are based on some unknown parameters, that is the number k_i of tuples satisfying the condition $cond_i$ and the number k of tuples satisfying d" conditions jointly. Therefore, the estimation of these parameters, which represent the selectivity of the conditions, is fundamental for the query plan cost evaluation.

As remarked before, the estimations must be performed quickly, therefore, an exact evaluation of the selectivity of the conditions (e.g., by means of detailed tables storing the number of tuples assuming each value of the attributes domains) is infeasible.

On the other hand, an accurate estimation of the selectivity of each condition is mandatory. In fact, wrong selectivity estimations would bias the estimated cost of each execution plan, thus possibly leading the query optimization module to a wrong choice.

Thus, the problem of selecting the optimal query plan, can be effectively solved by efficiently estimating how many of tuples satisfy each condition specified by the query separately how many tuples satisfy all the conditions jointly. This task can be achieved by answering range queries on the data distribution representing the attribute value frequency distribution and the joint frequency distribution of tuples, which are defined in the following.

Definition 1.1 (Attribute values frequency distribution) *Let R be a relation, X an attribute of R defined on the domain $D = [inf..sup]$. The attribute values frequency distribution of X is a function $f_X : [inf..sup] \rightarrow [0, 1]$ such that $f_X(v)$ is the ratio between the number of tuples $t \in R$ satisfying the condition $t.X = v$ and the overall number of tuples in R.*

The number of tuples satisfying several conditions specified on different attributed jointly is represented by the multi-dimensional function defined in the following.

Definition 1.2 (Joint frequency distribution) *Let R be a relation defined over attributes $X_1 \ldots X_d$, with X_i defined over the domain $D_i = [inf_i..sup_i]$. The joint frequency distribution of the attributes $X_1 \ldots X_d$ is a function $f : D \rightarrow [0, 1]$, where $D = D_1 \times D_2 \times \cdots \times D_d$, such that, for each $\langle v_1, v_2, \ldots v_d \rangle$ with $v_i \in D_i$, $f(v_1, v_2, \ldots v_d)$ is the ratio between the number*

[3] If a primary index were defined over an attribute $R.X_i$, tuples would be ordered in the pages according to $R.X_i$, and accessing the primary index, with cost $\mathcal{O}(\log_{\lfloor n/m_i \rfloor} n)$, would enable to exactly determining the number of pages containing the tuples satisfying $cond_i$.

of tuples $t \in R$ satisfying the conditions $t.X_i = v_i \ \forall i \in [1..d]$ and the overall number of tuples in R.

The most simple assumption on which it is possible to base the estimation of the frequency distribution for each attribute X_i, is the *uniform frequency distribution assumption* (UFD), which states that there is an equal number of tuples assuming each possible value of the domain on which X_i is defined.

More formally, the value frequency distribution of the attribute X_i, f_{X_i}, is estimated to be constant and equal to

$$\tilde{f}_{X_i}(v) = \frac{1}{sup_i - inf_i + 1}, \forall v \in [inf_i..sup_i]. \tag{1.4}$$

Therefore, the portion of tuples satisfying a condition $l_i \leq t.X_i \leq h_i$ is estimated as

$$\sum_{v=l_i}^{h_i} \tilde{f}_{X_i}(v) = \frac{h_i - l_i + 1}{sup_i - inf_i + 1}.$$

In order to estimate the joint frequency distribution, the correlation among attributes has to be evaluated. The most simple assumption consists in the absence of correlation, namely in the *attribute value independence assumption* (AVI). According to this assumption, the joint frequency distribution of d attributes is estimated as the product of the frequency distributions of the single attribute values. Therefore, $f(v_1, v_2, \ldots, v_d)$ is estimated as

$$\tilde{f}(v_1, v_2, \ldots, v_d) = \prod_{i=1}^{d} f_{X_i}(v_i). \tag{1.5}$$

These two assumptions, mainly owing to their simplicity, were the first to be adopted in the majority of models proposed in 1970s for studying the problem of secondary indexes selection both in database design [114, 119] and query optimization [116].

The following example illustrate how the query evaluation plan is selected by means of these two assumptions.

Example 1.1 Suppose to submit the following query to a relational DBMS:

```
SELECT *
FROM Employee
WHERE age<=40 AND income>25 000;
```

being *Employee* a relation containing 50 000 tuples stored in 1 250 pages. Suppose that non-clustered indexes (i.e., tuples in data pages are not ordered according to index key) are defined on the attributes *age* and *income* and each index is a B^+-Tree index stored in 200 pages. Suppose that the values of the attribute *age* range between 18 and 65 and the values of the attribute *income* range between 10 001 and 50 000.

Let c_i denote the cost of choosing the i-th of the four possible query execution plans, which are the following:

1. disregarding both indexes and scanning the entire relation, evaluating both conditions on each tuple of *Employee*;
2. accessing index on *age*, selecting pages *id*s with tuples satisfying $age \leq 40$ and then accessing only these pages for retrieving the qualifying tuples;
3. accessing index on *income*, and the proceeding like in the previous case;
4. accessing both indexes and then accessing only the pages that are referenced by both indexes (i.e., the data pages in the intersection between the two index results).

Each c_i is given by the estimated number of accessed index pages, I_i, and the estimated number of accessed data pages, P_i.

Obviously, according to the first possibility, $I_1 = 0$ and $P = 1\,250$, as the entire relation spans 1 250 pages.

In order to estimate the other costs, an estimation of the selectivity of each condition separately and of both conditions jointly is needed.

On the hypothesis of uniform distribution assumption, the query optimizer would estimate that

$$k_1 = 50\,000 \cdot \frac{40 - 17}{66 - 17} \cong 23\,958$$

tuples satisfy condition $age \leq 40$ and

$$k_2 = 50\,000 \cdot \frac{50\,000 - 25\,000}{50\,000 - 10\,000} = 31\,250$$

tuples satisfy the condition $income > 25\,000$. Adopting the attribute value independence assumption, the estimated number of tuples satisfying both conditions would be

$$k = 50\,000 \cdot \frac{40 - 17}{65 - 17} \cdot \frac{50\,000 - 25\,000}{50\,000 - 10\,000} \cong 14\,974.$$

Thus, the cost of accessing index on *age* can be estimated as

$$I_{age} = \lceil \log_{\lfloor 50\,000/200 \rfloor} 50\,000 \rceil + \left\lceil 200 \cdot \frac{23\,958}{50\,000} \right\rceil = 98,$$

and similarly the cost of accessing index on *income* can be estimated as

$$I_{income} = \lceil \log_{\lfloor 50\,000/200 \rfloor} 50\,000 \rceil + \left\lceil 200 \cdot \frac{31\,250}{50\,000} \right\rceil = 127.$$

The 2nd and the 3rd query plans entail accessing indexes on *age* and *income*, respectively, therefore $I_2 = I_{age} = 98$ and $I_3 = I_{income} = 127$. The 4nd query plan entails accessing both indexes, thus $I_4 = I_{age} + I_{income} = 225$.

In order to estimate the number of accessed data pages, it is possible to adopt Equation 1.2, which gives $P_2 = P_3 = P_4 = 1\,250$ for all the three estimated numbers of tuples, k_1, k_2 and k, that have to be retrieved, according to the 2nd, 3rd and 4th possible query plans.

Therefore, the estimated cost of 2nd, 3rd and 4th query plan is, respectively, $c_2 = I_2 + P_2 = 1\,348$, $c_3 = I_3 + P_3 = 1\,377$ and $I_4 = I_4 + P_4 = 1\,475$. On the basis of these estimations, the query optimizer will choose the first query evaluation plan, which disregards the indexes and access the entire relations. □

Unfortunately, the simplifying assumptions of *UFD* and *AVI* are often too simplistic as they rarely holds in real-life data distributions, and it is likely that they mislead the query optimizer. Uniform distributions are quite unlikely in real-life data, while correlation among attributes is quite likely to hold[4]. In order to better handle real data distributions, several techniques for accurately and efficiently approximating them have been proposed. One of the very first works addressing the problem of approximating effectively data distributions was proposed by Merret and Otoo in 1979 [101]. They proposed to use tables, for summarizing attribute values distributions. These tables can be obtained by partitioning the domain of each attribute, thus obtaining some multi-dimensional range, and storing the count of tuples falling into each of these ranges. Instead of applying the uniform frequency distribution assumption to the overall data domain, the uniformity assumption is applied locally inside each sub-domain corresponding to a cell of the table.

	10 001–20 000	20 001–30 000	30 001–40 000	40 001–50 000	Total
18–29	13 500	300	0	0	13 800
30–41	17 750	1 990	10	0	19 750
42–53	7 750	3 065	30	5	10 850
54–65	3 500	2 045	40	15	5 600
Total	42 500	7 400	80	20	50 000

Table 1.1. Two-dimensional table representing the number of tuples grouped by some ranges of *age* and *income* values

Table 1.1 depicts the distribution of tuples for the relation considered in the previous example. By means of this table, the query optimizer, for the same previous example, would have made the following estimations.

Example 1.1 (continued) The number of tuples satisfying the condition $age \leq 30$ is estimated as

[4] For instance, functional dependencies, which are a very common form of constraints adopted in relational databases, represent strong correlations among attributes.

$$k_1 = 13\,800 + 19\,750 \cdot \frac{40 - 29}{41 - 29} \approx 31\,904.$$

Notice that this is likely to be a better estimation than the uniform frequency distribution assumption over the whole domain, as the partial contribution to the estimation of 13 800 tuples with $age \in [18..29]$, is an exact contribution. The estimation is only performed on the number of tuples with $age \in [30..40]$.

The number of tuples satisfying the condition on *income* is estimated as

$$k_2 = 5\,400 \cdot \frac{30\,000 - 25\,000}{30\,000 - 20\,000} + 80 + 20 = 3\,800.$$

In fact, from Table 1.1 it emerges that the exact number of tuples with $income \in [30\,001..50\,000]$ is exactly $80 + 20$. Instead, the number of tuples with $income \in [25\,001..30\,000]$ must be estimated.

Finally, the estimated number of tuples satisfying both the conditions is $k = 1\,071$.

Thus, the cost of accessing index on *age* can be estimated as

$$I_{age} = \lceil \log_{\lfloor 50\,000/200 \rfloor} 50\,000 \rceil + \left\lceil 200 \cdot \frac{31\,904}{50\,000} \right\rceil = 130,$$

and similarly the cost of accessing index on *income* can be estimated as

$$I_{income} = \lceil \log_{\lfloor 50\,000/200 \rfloor} 50\,000 \rceil + \left\lceil 200 \cdot \frac{3\,800}{50\,000} \right\rceil = 18.$$

The 2nd and the 3rd query plans entail accessing indexes on *age* and *income*, respectively, therefore $I_2 = I_{age} = 130$ and $I_3 = I_{income} = 18$. The 4nd query plan entails accessing both indexes, thus $I_4 = I_{age} + I_{income} = 148$.

The estimated number of accessed data pages for the 2nd, the 3rd and the 4th evaluation plan, is $P_2 = 1\,250$, $P_3 = 1\,198$ and $P_4 = 725$, respectively.

Therefore, the 4th evaluation plan results to be the more efficient.

Observe that the estimation of the number of tuples satisfying both the conditions was not based on the attribute value independence assumption, but on an approximation of the joint frequency distribution by means of a table.

By adopting the attribute value independence assumption, on the basis of the estimated number of tuples satisfying the condition on *age* and that on *income* separately, the estimated number of tuples satisfying both conditions would be $k = 2\,425$, which would make the estimated number of accessed data pages equal to $P_4 = 1\,079$. In this case, the cost of the 3rd evaluation plan, $c_3 = I_3 + P_3 = 1216$, would be estimated less than the cost of the 4th evaluation plan, $c_4 = I_4 + P_4 = 1\,363$. □

It is quite intuitive that the estimation based on the adoption of a table is likely to be more accurate, than the estimation based on the hypothesis of uniform frequency distribution and, moreover, the approximation of the joint

frequency distribution by means of a table is likely to be more accurate than the attribute value independence assumption, even though the single attribute distributions are estimated by means of tables on each single attribute.

Form the example above it emerged how different estimates of condition selectivity can significantly yield different efficiency in executing a simple query. Therefore, it turns out that efficiently and effectively estimating data density distribution is a crucial task in query optimization, even in the most simple cases. Indeed, selectivity estimation is also at the basis of more complex cases of query optimization, where projection, selection and join operators are defined within the same query. In this case, the query optimizer should choose the best order in operators execution, i.e. that which minimize a cost function according to some model [22, 77]. Intuitively, the best execution plan is that which yields the smaller in size intermediate results. The majority of the studies related to this topic is mainly focused on the optimization-related aspect of the problem. For instance, in [123] Swami and Gupta studied how to apply algorithms for general combinatorial optimization problems based on local search to the *large join query optimization problem* (LJQOP). They found that among *perturbation walk*, *quasi-random sampling*, *iterative improvement*, and *simulated annealing*, the iterative improvements usually yields the best execution plans. Instead, Ioannidis, Wong, and Kang in [67] and [68] found that simulated annealing can identify lower cost access plans w.r.t. iterative improvement. Independently on the adopted optimization algorithm, which is necessarily an approximate one in order to avoid the exponential cost of the unacceptable exhaustive search technique, the estimation of each single operation cost is based on the selectivity estimation of the operation itself. Any optimization strategy requires that the selectivity estimation of the singles operations is performed efficiently, due to the large number of estimation that could be required, and effectively, otherwise the chosen query plan could be very far from being optimal.

1.3 Exploratory data analysis in OLAP applications

In *Decision Support Systems* (DSS) several tools for managing huge amount of data can help *knowledge workers* in choosing the best marketing strategies for maximizing their company profits. Traditional data representation models, such as the relational one, are not suitable for those kind of tasks. In fact, relational DBMSs have been essentially targeted to the *on-line transaction processing* (OLTP), where many transactions can be run concurrently and commonly entail inserting, reading or updating few records. Thus, these DBMSs are optimized for maximizing the transaction throughput, guaranteeing consistency and recoverability of data, which are the most critical aspects in operational databases. Instead, they are not targeted for data analysis tasks which require efficient access to a large number of records per transaction.

Therefore, in the last few years, the definition of both new data representation models and management systems became mandatory, in order to efficiently support DSS. Specifically, new technologies for the *on-line analytical processing* (OLAP) were required, i.e. technologies tailored for the integration and analysis of large amounts of data, where transaction concurrency is not the primary requirements, whereas transactions on large amounts of data are common and must be performed as efficiently as possible. The *data warehouse* has emerged as the system capable of guaranteing those requirements [21]. According to Immon, a data warehouse is a "subject oriented, integrated, time-varying, non-volatile collection of data that is used primarily in organizational decision making" [66].

Data warehouses must support the integration of large amount of consolidated historical data, possibly gathered from several sources, and thus may be required to store terabytes of data.

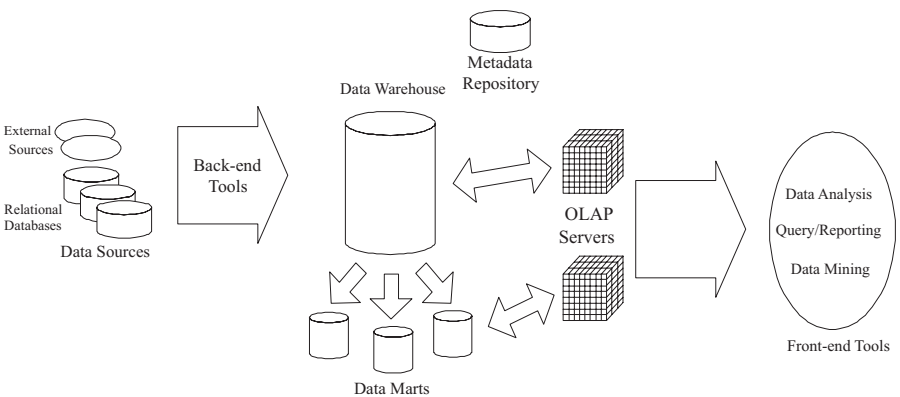


Fig. 1.1. Data Warehouse architecture

The general architecture of a data warehousing system is depicted in Fig. 1.1. Its main components are:

- *Back-end tools*: are adopted to extract and integrate data from different sources, which generally require to be pruned by possible errors and to be transformed according to a standard schema stored in the *metadata repository*;
- *Data warehouse*: is the database which can be based either on the relational model or on the multi-dimensional model;
- *Data marts*: are more specific databases, each one specialized on a particular subject of interest. They are often constructed as an alternative to the data warehouse, as they are more simple, and thus more cheap, to be built;

- *OLAP servers*: provide a logical multi-dimensional model for data and the basic functionalities for efficiently operating on them;
- *Front-end tools*: are those tools which are adopted by the final user (the knowledge worker) for analyzing data.

The multi-dimensional model adopted by the OLAP servers is needed in order to efficiently support front-end tools. In the multi-dimensional data are logically organized in structures called *data cubes* [52, 53]. For example, consider a relation *Sales* with attributes *Product*, *Year*, and *Amount*. This relation contains three attribute which in a DSS have different functions: in fact, one of them represent a *measure* that has to be studied with respect to some contexts which are represented by the other two attributes, year and product, which are called *dimensional* attributes. The relation *sales* can be also represented as a two-dimensional array, where each dimension correspond to a different dimensional attribute, and the value inside the cell represent the value associated with the corresponding dimensional values.

Product	Year	Amount
P1	2001	3 000
P1	2002	4 000
P2	2002	2 000
P1	2003	3 000
P2	2003	4 000
P3	2003	2 000
P1	2004	2 000
P2	2004	4 000
P3	2004	5 000
P4	2004	3 000
P3	2005	6 000
P4	2005	9 000

(a)

	P1	P2	P3	P4
2001	3 000	*null*	*null*	*null*
2002	4 000	2 000	*null*	*null*
2003	3 000	4 000	2 000	*null*
2004	2 000	4 000	5 000	3 000
2005	*null*	*null*	6 000	9 000

(b)

Fig. 1.2. A table (a) and its corresponding multi-dimensional representation (b).

In Fig. 1.2 a possible instance of the relation *Sales* (a) and its multi-dimensional representation (b) are depicted. It can be observed that several values in the two-dimensional array are null. This is due to the fact that the array contains the cells corresponding to all the possible combinations of the values which the dimensional attributes can assume (formally, there is a cell for each element of the cartesian product of the dimensional attribute domains). Instead, in a relation, only some combinations of attribute values appear as records. The difference between the number of cells and the number of tuples of the original relation becomes more and more relevant as the number of dimensions increases. In fact, while usually a larger number of attributes does

not make increase the number of tuples, the d-dimensional array size increase exponentially w.r.t. d. This phenomenon affects the density of data cubes (i.e., the ratio between the number of non-null values and the number of cells) at high-dimensionality, which in the real cases are often very very sparse.

A hierarchy is often defined over dimensional attributes. For instance, if an attribute *date* belongs to a relation, a hierarchy *year-month-day* can be defined over *date*. Thus, several data cubes can be obtained by adopting aggregations at different hierarchical levels on the attribute *date*.

The basic functionalities which OLAP servers provide to front-end tools are aggregate operators, which enable to adapt the multi-dimensional view of data:

- *roll-up*: increases the level of aggregation on a dimensional attribute which is characterized by a hierarchy;
- *drill-down*: is the opposite to the roll-up operator, decreasing the level of aggregation on a dimensional attribute;
- *slice-and-dice*: selects a hyper-rectangular subregion of the data cube by assigning precise values to some dimensional attributes;
- *pivoting*: exchange the order of the dimensional attributes, thus re-orienting the multi-dimensional view of data.

A typical data analysis process consists in exploring data in order to find "interesting" regions of the data cube (*data exploratory analysis*). For example, both isolated non-null points of the data cube (outliers) and particularly dense regions can be of interest in a data mining process. In order to efficiently locate these regions, the study of terabytes of data can result infeasible. Therefore, data are first studied at a high level of aggregation, which represent much more compact versions of the data cube, thus efficiently manageable, but still enabling dense or sparse regions to be located. Then, by means of slice-and-dice and drill-down operations, the analysis is specialized towards the regions of interest, without needing to work on the overall data cube which, as remarked before, can be infeasible.

In order to have an exact representation of the data cube at several levels of aggregations, several data cubes should be materialized (of course, an aggregation on-the-fly would not be an efficient solution). This could be infeasible, as a too large number of different aggregations could be possible [60]. Therefore, another solution to the problem of efficiently estimating the density of data cube regions, by means of range queries, is needed. Of course, in order to drive the data analysis towards the regions of actual interest, estimation must be as accurate as possible.

1.4 Estimating data distributions in other contexts

Besides the three application contexts previously described, for which several proposed estimation techniques have been specifically targeted, there are sev-

eral other contexts which require an efficient and effective estimation of a data distribution.

For example, in scientific and statistical databases [118], users could be interested in the number of records satisfying certain criteria rather than the records themselves. Executing such a query by computing the result set and then applying a count operator, would yield a waste of computational power. The availability of data statistical summary could enhance the system performance [88].

Another scenario where statistical summary could be useful is that in which the user may have not idea of the result size of the query s/he poses. If the result size were to large, probably many of the results could be uninteresting for the user. Thus, if the system gave a warning message before executing a time-consuming process, the user could rewrite the query specifying more restricting conditions, thus avoiding a waste of computational resources.

A more specific problem arises in those contexts where a user may be not interested in all the tuples satisfying a query s/he poses, but rather in a subset of them, characterized by the N tuples which "better" satisfy a property of interest. For example, consider the case in which a user is interested in buying some recently published book from an online shop. From a search form s/he could specify a lower bound for the publishing date, but s/he could obtain too many or too few results. One possibility could be reformulating the query with another date. Another possibility could be ordering the results, if too many, according to the publication date. The better solution would be having the possibility to ask for the N most recent books. Of course, in this case, the DBMS could give an exact response, but a trivial algorithm based on the ordering of all the result tuples on the attribute of interest could be too expensive w.r.t. the limited number of the actually returned tuples. This problem is known as *Top N query* [24, 36] on the attribute X and it is equivalent to a simple selection query of the form $\sigma_{X>k}(q)$, where q is the result of the query and k is a cutoff parameter strictly depending on N and on the data distribution w.r.t. the attribute X. Knowing the value k before executing the query would enable to filter the results during the query execution, and not as an additional step. Of course, if the tuple distribution w.r.t. the values of X were exactly known, k could be exactly computed. However, the required cost of this computation would make void the benefits of knowing k in advance. Therefore, a possible solution consists in quickly evaluating k on an approximation of the data distribution, and to filter the query result on the basis of the approximate value of k. Of course, the query result would be acceptable only if the error in approximating k were not to large.

Another context in which estimating efficiently data distribution can be useful is that of *geographical information systems* (GISs) [3, 96]. In this context, spatial data can represent several information associated to points of a two-dimensional or three-dimensional data domain, such as altitude, pollution, temperature, rainfall, humidity, presence of roads, etc. A typical query

on a geographical database could be "compute the average pollution in a given area". It is obvious that the efficient evaluation of this query could be achieved by efficiently evaluating a distribution corresponding to the pollution levels w.r.t. the geographical space.

The GIS context could be generalized adding the time as another dimension of the data distribution. The monitoring of several environmental variables can be very important in order to predict natural catastrophes. In fact, if past situations are known to have leaded to environmental problems, predictions of future ones can be achieved by analyzing the data distribution representing the current situation. These kind of data, which are continuously collected into a database and represent information associated to instants of time, are called *data streams* [135, 136]. As those data can be huge in volume and quick response to queries could be mandatory, even in this context an efficient evaluation of a data distribution is needed.

The problem of query optimization could become more important in the case of distributed database systems [134]. In fact, in this case, transferring intermediate results to one node of the distributed system to another could be needed. Thus, as transmission costs usually overwhelm the local ones (cpu, main and secondary memory costs), maintaining as small as possible intermediate query results could be even more important than in the local context. Of course, the strategy of the query optimizer is not very different: only in the cost model, the weight of data transmissions has to be taken into account. Another important issue in distributed systems but also in local multi-processor systems is the load-balancing for query execution [110]. When executing a join operation, load-balancing algorithms can fairly distribute the overall load among processors on the basis of the underlying data distributions of the relations involved in the join. Thus, even in distributed systems, an effective and efficient technique for estimating data distributions, possibly remotely resident, continue to be very important.

1.5 Generalization of the data distribution estimation problem

From the previous sections, it emerges that the problem of estimating a multidimensional data distribution, both efficiently and accurately, is a fundamental requirement in several important contexts. As the estimations must be performed quickly, original data distribution can not be directly accessed, as it would require a computational cost similar to those of the task that the estimates are required to support. The data summarization is a widely accepted solution, which entails accepting a trade-off between estimation accuracy and efficiency. In fact, the previously described problems can be supported by the possibility of efficiently computing *range queries*, with approximate answers, on the summarized data distribution. The storage space taken up by the summarized representation of the actual data distribution is generally considered

indicative of the query estimation costs. In fact, the larger the amount of information to be processed, the larger the processing time. Therefore, the storage space bound which can be invested in representing summarized data can be evaluated on the basis of the estimation efficiency required by the systems. Data summarization techniques aim at approximately describe the original data within an imposed storage space bound and with the minimum possible loss of information. Therefore, the problem of efficiently and accurately estimating data distribution, is translated into the following problem:

Problem 1.1 (Data summarization problem). Given a d-dimensional data distribution D and a storage space bound B, summarize D in a lossy representation D' within storage space B minimizing a measure of the error of the estimates performed on D'.

This problem, for the several important contexts in which it finds applications, has been largely studied, and several techniques for data summarization have been proposed in the last three decades. A general overview of the these technique will be provided in Chapter 2. In the following, some basic notations which will be adopted throughout the rest of this thesis are introduced.

1.5.1 Basic notations

The d-dimensional data distribution D will be treated as d-dimensional array of integers with volume n^d (without loss of generality, all dimensions of D will be assumed to have the same size).

The values contained in the array cells can represent different information, depending on the application context. For instance, in the selectivity estimation context, D represents the joint frequency distribution of a relation, that is each value associated to the point with coordinates $\langle v_1, \ldots, v_d \rangle$ represent the number of d-tuples of the relation whose i-th attribute is equal to v_i ($\forall i \in [1..d]$). In the OLAP analysis context, D represent the data cube, therefore the values in its cells represent the values of the measure attribute which has to be studied w.r.t. the d dimensional attributes by DSS tools. The cells of D which do not correspond to any entry of the data distribution are assumed to contain the value 0.

The number of non-null cell of D will be denoted as N_z. A *one-dimensional range* ρ_i on the i-th dimension of D is an interval $[l..u]$, with $1 \leq l \leq u \leq n$. 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 one-dimensional range on the dimension i, for each $1 \leq i \leq d$. Informally, a block represents a hyper-rectangular region of D, that is a sub-array of D. A block b of D with all zero

elements is said to be a *null block*. The volume of a block $b = \langle \rho_1, \ldots, \rho_d \rangle$ is denoted by $vol(b)$ and is given by

$$vol(b) = size(\rho_1) \times \cdots \times size(\rho_d).$$

Given a point in the multidimensional space $\mathbf{p} = \langle p_1, \ldots, p_d \rangle$, $\mathbf{p}$ will be said to belong to the block b (written $\mathbf{p} \in b$) if $lb(\rho_i) \leq p_i \leq ub(\rho_i)$ for each $i \in [1..d]$. A point $\mathbf{p}$ in b is said to be a *vertex* of b if for each $i \in [1..d]$ p_i is either $lb(\rho_i)$ or $ub(\rho_i)$. The value associated to a point $\mathbf{p}$ in D will be denoted as $D[\mathbf{p}]$. The sum of the values associated to all points inside b will be denoted as $sum(b)$. Thus,

$$sum(b) = \sum_{\mathbf{p} \in b} D[\mathbf{p}]. \tag{1.6}$$

A block will be also referred to as *multi-dimensional range*, or simply *range* (observe that a one-dimensional range is a d-dimensional range with $d = 1$).

The *Minimal Bounding Rectangle (MBR)* of a block b, is the block with the minimum size which contains all the non null values of b.

A *range query* Q on D is specified by a block b and its answer is $sum(b)$. The answer of a range query can represent different information according to the application context. For instance, in the case of selectivity estimation, $sum(b)$, with $b = \langle \rho_1, \ldots, \rho_d \rangle$, represents the number of tuples with the i-th attribute A_i satisfying the conditions $lb(\rho_i) \leq A_i \leq ub(\rho_i)$, with $i \in [1..d]$, jointly. In the case that the range b of the query corresponds to a single point $\mathbf{p} = \langle p_1, \ldots, p_d \rangle$, i.e. $lb(\rho_i) = ub(\rho_i) = p_i \ \forall i \in [1..d]$, the query will be said *equality query* and its answer is simply $D[\mathbf{p}]$.

The summarized data distribution will be denoted as $\tilde{D}$. Thus, the estimation of the value associated to a point $\mathbf{p}$ will be denoted as $\tilde{D}[\mathbf{p}]$.

The *cumulative distribution* or *partial sum distribution* of D will represent the d-dimensional array C defined on the same domain of D and such that, for each point $\mathbf{p} = \langle p_1, \ldots, p_d \rangle \in D$, being $b_{\mathbf{p}} = \langle [0..p_1], \ldots, [0..p_d] \rangle$, it holds $C[\mathbf{p}] = sum(b_{\mathbf{p}})$. Informally, each point $\mathbf{p}$ of the cumulative distribution is associated the sum of all the values associated to the points contained in the sub-array of D having the points $\mathbf{0} = \langle 0, \ldots, 0 \rangle$ and $\mathbf{p}$ as two opposite vertices.

In order to measure the homogeneity of the data inside a block, the *Sum of Squared Errors* (*SSE*) will be adopted, defined as follows:

$$SSE(b) = \sum_{\mathbf{p} \in b} \left(b[\mathbf{p}] - avg(b) \right)^2, \tag{1.7}$$

where $avg(b) = \frac{sum(b)}{vol(b)}$ is the average of values inside b (including the null values).

The *marginal distribution* of a block b along the i-th dimension, denoted as $marg_i(b)$, is given by the "projection" of the internal data distribution of b on the i-th dimension, and represents a one-dimensional array. Specifically,

$$marg_i(b)[j] = \sum_{\mathbf{p} \in b \wedge \mathbf{p}[i]=j} D[\mathbf{p}]. \tag{1.8}$$

Fig. 1.3 shows marginal distributions for a two-dimensional block.

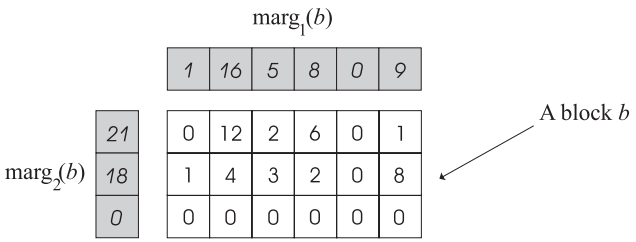


Fig. 1.3. Marginal distributions

Of course, marginal distributions can be viewed as one-dimensional data distribution, defined on the projection dimension.

2

Data summarization: state-of-the-art techniques

As pointed out in the previous chapter, the problem of efficiently estimating range queries on multi-dimensional data with tolerable approximations finds application in several important contexts. The data summarization has been widely accepted as the main solution which can enable a effective trade-off between accuracy and efficiency in estimations. A general overview of the multi-dimensional data summarization techniques is presented in this chapter. Simple techniques had been proposed since the late 1970s to the late 1990s. After 1997, the interest in the data summarization techniques has considerably increases and several more complex techniques have been proposed. The interest in the problem was probably stimulated by the paper [7], published in 1997 as a special issue on data summarization (addressed as data reduction in that paper), of the *IEEE Data Engineering Bulletin*. That paper was born from the initiative of Hellerstein, who gathered several of the most prominent figures in several research fields for a wide discussion on the data summarization problem, as he was convinced of the necessity to unify apparently distant approaches typical of different research fields in order to find an ultimate solution to the still open problem of effectively summarizing multi-dimensional data.

The data summarization techniques are usually roughly divided into two broad classes: *parametric* and *nonparametric* techniques.

Techniques belonging to the former class adopt a mathematical model to fit the data distribution, and aim at finding the the parameters of the model which yield the best approximation of the actual data distribution. As a model is usually characterized by few parameter, these techniques can yield a large compression factor. On the other hand, data can be effectively approximated only if they actually follow some model. Techniques based on mathematical transforms, such as the Fourier or the wavelet transform, are often classified among the parametric techniques [7]. After transforming data, the original data distribution is represented by a model corresponding to the anti-transform. The idea on which these techniques are based is that the

adopted transform usually provide a sort of compaction of the information, which enables to significantly reduce the amount of data to be accessed with a relatively little loss of information.

Nonparametric techniques make no assumption about a possible model followed by the overall data distribution, therefore they do not try to represent data by means of some concise formula. They rather try to summarize the overall data set by means of pieces of "local" information. Among these techniques, the most popular are the histogram- and the sampling-based ones. Histogram-based techniques partition the data set and represent each subset of data (called bucket) by means of some aggregate information (e.g., the count or the sum of the values inside the bucket). Sampling-based techniques extract a sample from the overall data set and use it to make inferences about the original data set. It is worth noting that these techniques are strictly related to the parametric ones. In fact, in order to infer estimates from summarized data, it is necessary to make some assumption about the original local data distribution (which is often assumed to be a uniform distribution) in the case of histogram-based techniques and some assumption about the sample (which is often assumed to be unbiased) in the case of sampling-based techniques. However, other assumptions are possible as well. Another nonparametric summarization technique is that based on *kernel functions*. A kernel function is a continuous function $K(x)$, usually symmetric w.r.t. x and unimodal (i.e., it has one only peak, usually in $x = 0$), which integrates to 1 over its domain. A *kernel estimator* can be viewed as a generalization of sampling, where each sampling point spreads its information across its neighborhood according to a kernel function. Therefore, if X_i is a sampling point, this point will give a contribution equal to $K(x) = \frac{1}{h}K\left(\frac{x-X_i}{h}\right)$ to the approximate data distribution. The parameter h is called *bandwidth* of the function, and controls how wide is the neighborhood of a sample point X_i affected by X_i. The data distribution, given a set of n samples, is represented by the weighted superimposition of the respective n kernel functions. Kernel estimators have been widely studied in statistics [129], resulting a very effective estimator, but they have not been much investigates in the database related contexts.

In the following sections the main techniques belonging to both classes will be discussed. In particular some of the parametric-, wavelet-, histogram-, and sampling-based approaches which are most known in literature will be described. Some interesting hybrid techniques, which combine the characteristic of both classes, will be presented as well.

2.1 Mathematical models

Parametric summarization techniques assume that data distribution follows a precise mathematical model. The model can be either a statistical distribution (e.g., uniform, normal, etc.) or a polynomial function. The choice of

parameters, representing either the statistical distribution or the polynomial function, is generally achieved by minimizing a measure of the approximations. The least square method is one of the most commonly adopted methods for choosing the parameters of a model, which entails minimizing the sum of the squared differences between original and approximate values. That is, if D is the original data distribution and $\tilde{D}_P$ is the approximation of D according to a mathematical model with an assignment P to its parameters,

$$SSE_P = \sum_{\mathbf{i} \in D} (D[\mathbf{i}] - \tilde{D}_P[\mathbf{i}])^2 \tag{2.1}$$

has to be minimized. For example, if a data set is supposed to follow a normal distribution, it will be represented simply by means of the parameters representing the average μ and the standard deviation σ of the normal distribution function adopted as mode, and an assignment to μ and σ which minimize Equation 2.1 must be found.

Even though a single value can be estimate quickly, a naive approach for estimating range queries, based on the estimation of all the values inside the range, could be infeasible. There are two possibilities to perform efficiently the estimation of range queries: computing the integral of the mathematical function over the range of the query or approximating the cumulative distribution instead of the original data distribution, and the estimating only the values coinciding with the vertices of the range.

Parametric techniques based on statistical models perform very well when data actually follow, even approximately, some model. For instance, when the creation of tuples in a database depends on real world phenomena which follow some known distribution, parametric techniques based can successfully accomplish the task of approximating with good accuracy actual data. In this cases, the achievable compression ratio is very high, as is it possible to describe even large data sets by means of a few parameters. On the counterpart, finding an effective model for a data set could be a hard task, and it is quite unlikely to effectively approximate a data distribution by means of wrong models. In addition, real data rarely follow a precise known statistical distribution.

Regression models are the most adopted mathematical models and they are more flexible than the statical ones. *Linear regression* is the most simple kind of regression model, and it assumes that data associated to an attribute X are represented by $D[x] = q + mx$, where x belongs to the domain of the attribute X. Multiple regression techniques can model multi-dimensional data distributions. For instance, $D[\langle x_1 \dots x_d \rangle] = q + m_1 x_1 + \dots m_d x_d$ is a multiple linear regression model for a d dimensional data distribution, where x_i is a value that the i-th attribute can assume. Nonlinear regression models enable to describe the data distribution D by adopting different powers of the attribute values, an possibly by multiplying the values of different attributes (e.g., $D[\langle x_1, x_2 \rangle] = q + m_1 x_1^2 + m_2 x_1 x_2$.

A possibility which often reduces the hardness to fit data distributions by means of analytical models consists in applying a suitable mathematical trans-

form to the data, changing the domain in which they are represented. Even though the transform itself does not reduce the amount of information to be stored, it often provides an effective "information compaction" which make easier discriminating of the most important "pieces" of information[1]. The original data distribution will be represented by means of the anti-transform whose parameters are only the most "important" values in the transform domain. For example, a time series can be represented by means of a Fourier series and approximated by cutting off to zero the coefficients of the Fourier series which least contribute to the signal power. As the transform provides energy compaction, discarding coefficients of the fourier series yields a lower loss of energy than disregarding the same number of data in time domain. These techniques are called transform-based techniques, and they can represent a subclass of the parametric techniques by distinguishing from the direct parametric techniques which directly exploit a model to approximate the data distribution.

In the context of multimedia data compression, mathematical transforms have been successfully exploited. For instance, *MPEG*, *JPEG* and *MP3*, are currently the standard formats for exchanging video, image and audio files, respectively, on the web, due to their excellent tradeoff between quality and storage space occupancy. The basic principle on which these lossy compression techniques are based, is the low sensitivity of human senses to some frequency components of the perceived signal spectrum. Removing the information associated to these frequencies does not affect significantly the perceived quality of the signal, but enable a substantial reduction of the information to be stored and thus the file size. Therefore, the majority of the multimedia compression techniques first compute the spectrum of the signals, usually by means of the *Discrete Cosine Transform* (DCT), and then remove those components which are less significant, by avoiding storing (i.e., implicitly setting to zero) those coefficients that in the anti-transform would add the least important components to the signal. The idea of employing the same techniques for summarizing general data, in particular in the contexts of selectivity estimation and OLAP analysis, was not applied until the late 1990s. In [7], *wavelets* were reported for the first time as a possible technique for data summarization in the context of approximate data analysis. Wavelets enable even better performances in compressing some kind of data than the DCT: in fact, the image compression format JPEG2000, based on wavelet transform, has been proved to perform better than the JPEG format, although it has not become the standard on the web as the majority of web browsers do not support it. Wavelets present some advantages w.r.t. DCT. Wavelet coefficients are localized (i.e., are related to local portions of the data distribution) whereas each DCT depends on the overall data domain. Thus, wavelets enables to repre-

[1] The same transforms, when applied to physical signals, provide an effective energy compaction which enable to discriminate the components of the signal associated with the greatest amount of energy.

sent with few coefficients even large local spikes, which instead affects a large number of DCT coefficients. Furthermore, a data distribution of size n can be decomposed according to wavelet transform in time $\mathcal{O}(n)$, whereas no algorithms for computing the DCT in time less than $\mathcal{O}(n \log n)$ are known, even though $\Omega(n \log n)$ has not been proved to be a lower bound for the complexity of the DCT computation problem.
Indeed, there exists a variant of the DCT, namely the *Short-time Discrete Cosine Transform* (STDCT), with properties similar to those of wavelets, such as coefficient localization and efficient computational time $\mathcal{O}(n)$ [82], even though its use has never been investigated in the database-related contexts.

Wavelet-based summarization techniques will be described in the following section. Now some of the most important parametric summarization techniques proposed in literature will described.

A seminal work in the context of query optimization was proposed by Selinger et al. in 1979 [116]. In that work, the authors described how their experimental DBMS, *System R*, which had been under development at IBM San Jose Research Laboratory since 1975 and it is the ancestor of *DB2*, worked in selecting execution plans for queries. The system used simple statistical information, such as the minimum and maximum value of each attribute domain, and assumed that each intermediate value was present with the same frequency. Therefore, they modelled tuple frequency distribution w.r.t. each attribute data domain by means of a uniform distribution (uniform frequency distribution assumption, see Equation 1.4). Furthermore, they modelled the joint distribution of tuples w.r.t. different attributes assuming no correlation among attributes (attribute value independence assumption, see Equation 1.5).

Thus, the only parameters to be stored, in order to estimate the selectivity of predicates on a single attribute X_i of a relation, were the highest and lowest values of the attribute domain, sup_i and inf_i, respectively, and the number of tuples T in the relation. On the basis of these information, and on the basis of the uniform frequency distribution assumption, the number of tuples satisfying the condition $l_i \leq X_i \leq u_i$ is estimated as $T \cdot \frac{u_i - l_i + 1}{sup_i - inf_i + 1}$.

Unfortunately, the uniform distribution model has always been considered unlikely to effectively model real data distributions. Zipf's law and Pareto's principle are rules of thumb which hold in several real life contexts, thus influencing the characteristics of information as well, and describe several natural phenomena as being far from following uniform distributions.

Specifically, the American linguist and philologist Zipf in 1940s observed that in a text written in a natural language containing N_z distinct words, ranking them on the basis of the number of their occurrences in the text, the number of occurrences $f(i)$ of the i-th word is approximately proportional to $\frac{1}{i}$ [138]. On the basis of his studies, the specific inverse exponential distribution represented by

$$f(i) = \frac{K}{i^z}, i \in [1..N_z], \text{ with } K = \sum_{i=1}^{N_z} \frac{1}{i^z}, \tag{2.2}$$

is called *Zipf distribution*. In the case that the distribution is adopted to model ranked word occurrences, on the basis of Zipf's studies, $z \approx 1$.

Even though this law states the unsuitability of models based on uniform distribution in some contexts, it gives a precise model which can approximate with good accuracy data distributions in the same contexts. For instance, Zipf distribution was proposed to model indexed files by Fedorowicz [41, 42], Schuegraf [115], and Siler [119].

The need to take into account different statistical distribution was proposed in several works between the late 1970s and the early 1980s. For instance, Hill studied the database performances for Poisson, Uniform and Zipf distribution of the index access key [63]. Christodoulakis in his PhD thesis [27] suggested to model attribute value frequencies by means of univariate Pearson distributions, which provide a range of distributions from the uniform to the normal one. In particular, in [28] he showed that, many attributes follow unimodal distributions, i.e. distributions with one only peak, such as the normal one, that can be approximated by a family of distributions including the Pearson types 2 and 7 and the normal distribution. The parameters (i.e., the various moments) of the models can be evaluated in a single scan and dynamically maintained.

In [88] Lefons et. al. proposed to store statistics of data by representing the distribution of attribute values by means of linear combinations of the first k Legendre's polynomials. They based the proposal on the observation that common polynomials of large degree may be subject to large oscillations, while Legendre polynomials are more "stable".

Fedorowicz in [43] proposed to adopt multiple linear regression models to approximate the frequency distribution of words in a bibliographic system. As the word occurrences are known to follow a Zipf distribution, he proposed to model the actual word distribution by means of a nonlinear regression model, and to apply the logarithm in order to transform the model into a multiple-linear-regression-based one.

In [122] Sun et al. proposed a parametric technique for approximating the attribute value frequencies by means of multiple regression models. For approximating the frequency distribution of a single attribute value X_i, which is represented by an array $D_i[x]$ defined on a discrete domain $v_1..v_m$, they adopted a regression model based on polynomial continuous functions of the following form

$$f_i(x) = \sum_{i=-k_2}^{k_1} c_i x^i.$$

The degree of the polynomial function is $k_1 + k_2$. They found that by adopting values of k_1 between 0 and 6 and values of k_2 between 1 and 4 a large variety of different data could be approximated with good accuracy. The choice of

the model coefficients c_i ($i \in [-k_2..k_1]$) was performed by means of the least square method, thus minimizing

$$\sum_{j=1}^{m} \left(D_i[v_j]) - f_i(v_j)\right)^2 .$$

The number tuples satisfying the condition $l \leq X_i \leq u$ ($l \geq inf_i$ and $u \leq sup_i$) can the then estimated as

$$\frac{\int_l^u f_i(x)}{\int_{v_1}^{v_m} f_i(x)}.$$

Observe that the denominator is a constant, therefore it must not recomputed at each estimation. As regards the joint distribution of d attributes $x_1 \dots x_d$, they proposed a multiple regression model, still based on polynomial functions, defined on d variables. The coefficients to be computed are those to be multiplied for all the possible terms $x_1^{g_1} \cdot x_2^{g_2} \cdot \dots \cdot x_d^{g_d}$ with $g_i \in [-k_2..k_1]$. The estimation of the number of tuples satisfying several condition on different attributes jointly can be performed by means of a d-multiple integral. Of course, the cost of performing estimation when the dimensionality increases, but also the best coefficients for the model, becomes infeasible.

Also in [25] Chen and Roussopoulos proposed to adopt a polynomial functions for approximating attribute value frequency distributions. They suggested to adopt $6th$ degree polynomial functions as model, thus they posed $k_2 = 0$ and $k_1 = 6$ according to the proposal in [122]. The novelty of the method relied in the use of query feedbacks for updating the approximating function coefficients, without scanning the actual data. As queries are performed, the exact answers are used as feedback, at no extra cost, for tuning the coefficients of the model. In addition, the possibility to weight feedbacks, giving higher relevance to more recent feedbacks, enable a very efficient update of the coefficient in order to adapt them to updates in data actual distribution.

In [86] Korn et al. proposed to approximate continuous univariate and multivariate data distributions by means of splines. In particular, they proposed to partition the data domain by means of an algorithm for the knot placement of a kernel estimator, and then to approximate data inside the so obtained ranges by means of cubic splines.

In [117] Shanmugasundaram et al. proposed a parametric-like technique which instead of adopting one only function for modelling the overall data distribution, it adopts several localized gaussian distributions. Specifically, by means of the *Expectation-Maximization* (EM) clustering technique [10], they proposed to find a certain number of data clusters, and to approximate each cluster by means of a d-dimensional gaussian distribution adopting $1 + 2d$ coefficient for representing the covariance matrix which is approximated by a diagonal matrix. In the case that a measure of the approximation of a cluster is below a threshold, the cluster is split and more gaussian models are adopted to approximate it.

2.2 Discrete Wavelet Transform

The word *wavelet* is a translation from the French word *ondelette*, that was coined in the early 1980s by the French geophysicist Jeán Morlet. The word derives from the particular kind of functions, having the form of fast decaying and localized waves, that are used as basis from a class of mathematical transforms. Wavelet-based transforms are suitable both for continuous functions and discrete series. In this context, the attention will be focused on the *Discrete Wavelet Transform* (DWT), as this is the suitable one for approximating discrete data distributions on discrete domains.

2.2.1 Basic mathematical background

By means of a continuous wavelet transform, it is possible to represent a function $f(t)$ as a linear combinations of a set of functions derived by dilating and translating (dilations and translations will be defined in the following) a function $\psi(t)$, called *mother wavelet*, and a scaling function $\phi(t)$ called *father wavelet*. In the discrete context, these functions are simply vectors and will be denoted as $\psi[\cdot]$ and $\phi[\cdot]$.

By applying the *DWT*, any vector v of size $n = 2^k$ can be represented as

$$v[t] = s_{00} \cdot \phi[t] + \sum_{i=0}^{\log n-1} \sum_{j=0}^{2^i-1} d_{ij} \cdot \psi_{ij}[t],\ 0 \leq t < n \tag{2.3}$$

where the vectors ϕ and ψ_{ij} have the same length n of v, ψ_{00} coincides with ψ and, for other values of i, j, ψ_{ij} represents the j-th translation of the i-th dilation of ψ. Decomposing v entails computing the coefficients s_0 and d_{ij}.

Equation 2.3, can be also written as

$$v = s_{00} \cdot \phi + \sum_{i=0}^{\log n-1} \sum_{j=0}^{2^i-1} d_{ij} \cdot \psi_{ij} \tag{2.4}$$

The set of vectors ϕ and ψ_{ij} is called *basis*, and it is univocally determined by the mother and father wavelet. Several wavelet transforms can be defined, depending on the set basis adopted in the transformation.

The first proposed DWT largely precedes the introduction of the word *wavelet*, and dates back to 1909, when the Hungarian mathematician Alfréd Haar introduced a mathematical transform based on the use of stepwise functions as mother and father wavelet: in the discrete context, the father wavelet ϕ is a vector with $\phi[t] = m$ for each $t \in [0..n-1]$ and the mother wavelet ψ is a vector such that $\psi[t] = m$ for $t \in [0..\frac{n}{2}-1]$ and $\psi[t] = -m$ for $t \in [\frac{n}{2}..n)$. In order to simplify the definition of the other basis vectors, $\psi[t] = 0$ for $t \notin [0..n-1]$ will be assumed.

In Fig. 2.1 the father (a) and mother (b) wavelets of the Haar basis are depicted. Of course, as remarked before, in the discrete context ϕ and ψ are

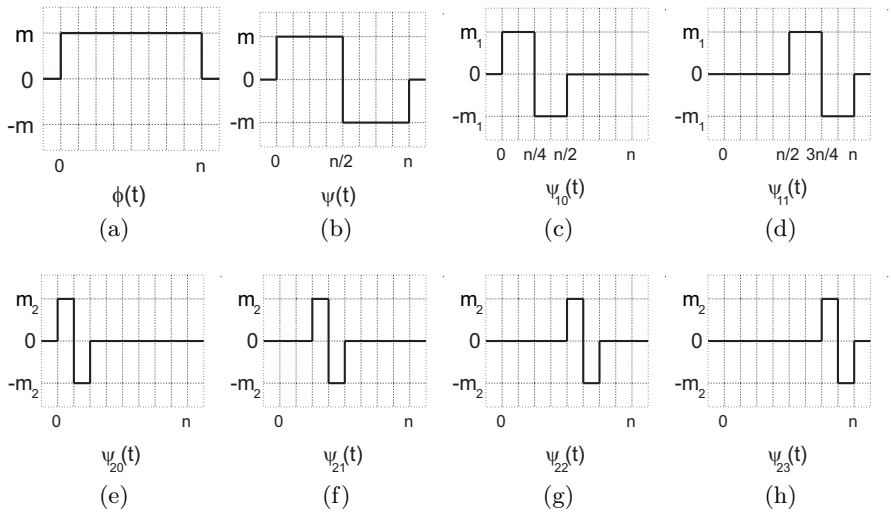


Fig. 2.1. Haar basis: father wavelet (a) and mother wavelet (b), 1st (c) and 2nd translation of the first order dilation of mother wavelet, 1st (e), 2nd (f), 3rd (g) and 4th (h) translation of the second order dilation of mother wavelet

not functions defined on the continuous as depicted in the figure, but vectors. The remaining vectors are obtained by dilating and translating of the mother wavelet. Specifically, $\psi_{ij}[t]$ is equal to $\frac{m_i}{m} \cdot \psi[2^i \cdot t - j \cdot \frac{n}{2^i}]$. the factor 2^i multiplied by t yields the dilation of the i-th order of the mother wavelet and the addend $-j \cdot \frac{n}{2^i}$ yields a translation of the dilations.

Figures 2.1 (c) and (d) depict the two possible translations of the first order dilating of the mother wavelet and Figures 2.1 (e), (f), (g) and (h) depict the four possible translation of the second order dilating. The highest order of dilating is the $(\log n - 1)$-th one, which yields vectors with all but two contiguous elements null. Observe that, the definition of ψ_{ij} requires that the vector has dyadic length (i.e., a number of elements equal to a power of 2). Several solutions have been adopted in order to generalize the DWT to vectors of non-dyadic length. The most simple one is the zero-padding, which entails extending the vector with the minimum number of zeros which makes the vector of dyadic length.

It is possible to show that the vectors which constitute the basis are mutually orthogonal. This property ensures that any vector can be represented as linear combinations of these vectors.

The multiplicative constant of each vector is chosen such that each vector has unitary modulus. It is easy to show that $m = m_0 = \frac{1}{\sqrt{n}}$, and $m_i = \sqrt{2} \cdot m_{i-1}$ for $0 < i < \log n$. These multiplicative factors makes the Haar basis orthonormal. The normalization of vectors in the basis ensures that the amount of information associated to each product between a coeffi-

cient of the transform and its corresponding vector of the basis depends only on the coefficient magnitude.

$$\mathbf{W} = \begin{bmatrix} +\frac{\sqrt{(2)}}{4} & +\frac{\sqrt{(2)}}{4} & +\frac{1}{2} & 0 & +\frac{\sqrt{(2)}}{2} & 0 & 0 & 0 \\ +\frac{\sqrt{(2)}}{4} & +\frac{\sqrt{(2)}}{4} & +\frac{1}{2} & 0 & -\frac{\sqrt{(2)}}{2} & 0 & 0 & 0 \\ +\frac{\sqrt{(2)}}{4} & +\frac{\sqrt{(2)}}{4} & -\frac{1}{2} & 0 & 0 & +\frac{\sqrt{(2)}}{2} & 0 & 0 \\ +\frac{\sqrt{(2)}}{4} & +\frac{\sqrt{(2)}}{4} & -\frac{1}{2} & 0 & 0 & -\frac{\sqrt{(2)}}{2} & 0 & 0 \\ +\frac{\sqrt{(2)}}{4} & -\frac{\sqrt{(2)}}{4} & 0 & +\frac{1}{2} & 0 & 0 & +\frac{\sqrt{(2)}}{2} & 0 \\ +\frac{\sqrt{(2)}}{4} & -\frac{\sqrt{(2)}}{4} & 0 & +\frac{1}{2} & 0 & 0 & -\frac{\sqrt{(2)}}{2} & 0 \\ +\frac{\sqrt{(2)}}{4} & -\frac{\sqrt{(2)}}{4} & 0 & -\frac{1}{2} & 0 & 0 & 0 & +\frac{\sqrt{(2)}}{2} \\ +\frac{\sqrt{(2)}}{4} & -\frac{\sqrt{(2)}}{4} & 0 & -\frac{1}{2} & 0 & 0 & 0 & -\frac{\sqrt{(2)}}{2} \end{bmatrix}$$

Fig. 2.2. Matrix representation of the DWT Haar basis for vectors of size 8

The n vectors ϕ and ψ_{ij} can be assembled in order to form a matrix $\mathbf{W} = [\phi^T, \psi^T, \psi_{10}^T, \psi_{11}^T, \ldots]$ of size $n \times n$. In Figure 2.2 the matrix representing the Haar basis for vectors of length 8 is depicted. If also the n coefficients s_{00} and d_{ij} are assembled into a vector $c = [s_{00}, d_{00}, d_{10}, d_{11}, \ldots]$, Equation 2.4 can be written as

$$\mathbf{W} \times c^T = v^T. \tag{2.5}$$

2.2.2 Fast Wavelet Decomposition

The wavelet decomposition could be computed by solving the system of n linear equations defined by Equation 2.5. The mutual orthogonality of vectors in $\mathbf{W}$ ensures that for each v there exists one only decomposition c, corresponding to the solution of the system of equations. Of course, computing the wavelet transform by solving Equation 2.5 would be infeasible for large values of n. Indeed, the wavelet decomposition can be computed very efficiently by means of a hierarchical bottom-up technique, without solving Equation 2.5. As remarked before, each coefficient $d_{i^\star j}$, with $i^\star = \log n - 1$, affects exactly one pair of values of v, depending on the value of j. Specifically, $d_{i^\star j}$ gives a positive contribution to $v[2j]$ and a negative one to $v[2j+1]$. Therefore, defining $s_{i^\star j} = \frac{v[2j]+v[2j+1]}{\sqrt{2}}$ and $d_{i^\star j} = \frac{v[2j]-v[2j+1]}{\sqrt{2}}$ for each pair of contiguous elements $\langle v[2j], v[2j+1] \rangle$ $(0 \leq j < n/2)$, it is immediate to show that $v[2j] = \frac{s_{i^\star j}[t]}{\sqrt{2}} + \frac{d_{i^\star j}}{\sqrt{2}}$ and $v[2j+1] = \frac{s_{i^\star j}[t]}{\sqrt{2}} - \frac{d_{i^\star j}}{\sqrt{2}}$. By imposing that the contribution $\pm d_{i^\star j}$ to the pair $\langle v[2j], v[2j+1] \rangle$ comes exactly from the last $n/2$ coefficients $c_{n/2} \ldots c_{n-1}$, it is necessary that the contribution $s_{i^\star j}$ comes from the first $n/2$ coefficients $c_0 \ldots c_{n/2-1}$. These coefficients can be recursively computed by adopting the same strategy, considering the vector $[s_{i^\star 0}, \ldots s_{i^\star n/2-1}]$ as new vector of size $n/2$ to be decomposed.

Example 2.1 Consider the vector $v = [7, 5, 1, 3, 8, 2, 8, 6]$ of length $n = 8$. At the first step, v is decomposed into

$$\begin{aligned} s_{20} &= \tfrac{v[0]+v[1]}{\sqrt{2}} = 6\sqrt{2},\ d_{20} = \tfrac{v[0]-v[1]}{\sqrt{2}} = \sqrt{2}, \\ s_{21} &= \tfrac{v[2]+v[3]}{\sqrt{2}} = 2\sqrt{2},\ d_{21} = \tfrac{v[2]-v[3]}{\sqrt{2}} = -\sqrt{2}, \\ s_{22} &= \tfrac{v[4]+v[5]}{\sqrt{2}} = 5\sqrt{2},\ d_{22} = \tfrac{v[4]-v[5]}{\sqrt{2}} = 3\sqrt{2}, \\ s_{23} &= \tfrac{v[6]+v[7]}{\sqrt{2}} = 7\sqrt{2},\ d_{23} = \tfrac{v[6]-v[7]}{\sqrt{2}} = \sqrt{2}. \end{aligned}$$

The four coefficients d_{20}, d_{21}, d_{22} and d_{23} correspond to the last four coefficients of the decomposition, namely to c_4, c_5, c_6 and c_7. The remaining coefficients of the decomposition are computed by decomposing the vector

$$s_2 = [s_{20}, s_{21}, s_{22}, s_{23}] = [6\sqrt{2}, 2\sqrt{2}, 5\sqrt{2}, 7\sqrt{2}],$$

thus obtaining

$$\begin{aligned} s_{10} &= \tfrac{s_2[0]+s_2[1]}{\sqrt{2}} = 8, \quad d_{10} = \tfrac{s_2[0]-s_2[1]}{\sqrt{2}} = 4, \\ s_{11} &= \tfrac{s_2[2]+s_2[3]}{\sqrt{2}} = 12,\ d_{11} = \tfrac{s_2[2]-s_2[3]}{\sqrt{2}} = -2. \end{aligned}$$

d_{10} and d_{11} are assigned to the coefficients c_2 and c_3, respectively. Finally, $s_1 = [s_{10}, s_{11}] = [8, 12]$ is decomposed into

$$s_{00} = \tfrac{s_1[0]+s_1[1]}{\sqrt{2}} = 10\sqrt{2} \text{ and } d_{00} = \tfrac{s_1[0]-s_1[1]}{\sqrt{2}} = -2\sqrt{2}$$

and the decomposition ends, assigning s_{00} to c_0 and d_{00} to c_1. The resulting vector representing the transform is

$$c = [10\sqrt{2}, -2\sqrt{2}, 4, -2, \sqrt{2}, -\sqrt{2}, 3\sqrt{2}, \sqrt{2}].$$

By inverting the decomposition process, it is possible to reconstruct the original values of v. In fact, from $s_{00} = c_0$ and $d_{00} = c_1$ it is possible to reconstruct

$$s_1 = [\tfrac{c_0+c_1}{\sqrt{2}}, \tfrac{c_0-c_1}{\sqrt{2}}],$$

from s_1 and $d_1 = [c_2, c_3]$ it is possible to reconstruct

$$s_2 = [\tfrac{\frac{c_0+c_1}{\sqrt{2}}+c_2}{\sqrt{2}}, \tfrac{\frac{c_0+c_1}{\sqrt{2}}-c_2}{\sqrt{2}}, \tfrac{\frac{c_0+c_1}{\sqrt{2}}+c_3}{\sqrt{2}}, \tfrac{\frac{c_0+c_1}{\sqrt{2}}-c_3}{\sqrt{2}}] =$$
$$[\tfrac{c_0}{2} + \tfrac{c_1}{2} + \tfrac{c_2\sqrt{2}}{2}, \tfrac{c_0}{2} + \tfrac{c_1}{2} - \tfrac{c_2\sqrt{2}}{2}, \tfrac{c_0}{2} + \tfrac{c_1}{2} + \tfrac{c_3\sqrt{2}}{2}, \tfrac{c_0}{2} + \tfrac{c_1}{2} - \tfrac{c_3\sqrt{2}}{2}]$$

and finally from s_2 and $d_2 = [c_4, c_5, c_6, c_7]$ it is possible to reconstruct v, which results $v = (\mathbf{W} \times c^T)^T$. □

In both the decomposition and reconstruction phases, it easy to observe that the number of addition or subtraction and division performed is $2 \cdot (n-1)$, where n is the length of v, thus the complexity of performing both the DWT and the anti-transform is $\mathcal{O}(n)$. In Fig. 2.3 the algorithms for computing the wavelet transform (a) and anti-transform (b) are reported.

```
INPUT   v: a vector of length n;
OUTPUT  c: a vector of length n representing the Haar discrete wavelet transform of v;

begin
  c = new double[n];
  for (i ∈ {0..n−1}) do
    c[i]=v[i];
  l=n;
  while (l > 1) do begin
    s = new double[l/2];
    d = new double[l/2];
    for (i ∈ {0..l−1}) do begin
      s[i]=(c[2i]+c[2i+1])/√2;
      s[i]=(c[2i]−c[2i+1])/√2;
    endfor;
    for (i ∈ {0..l−1}) do begin
      c[i] = s[i];
      c[n/2+i] = d[i];
    endfor;
    l=l/2;
  endwhile;
  return c;
end;
```

(a)

```
INPUT   c: a vector of length n;
OUTPUT  v: a vector of length n representing the Haar discrete wavelet anti-transform of c;

begin
  v = new double[n];
  for (i ∈ {0..n−1}) do
    v[i]=c[i];
  l=1;
  while (l < n) do begin
    s = new double[l];
    d = new double[l];
    for (i ∈ {0..l−1}) do begin
      s[i] = v[i];
      d[i] = v[l+i];
    endfor;
    for (i ∈ {0..l−1}) do begin
      v[2i]=(s[i]+d[i])/√2;
      s[2i+1]=(s[i]−d[i])/√2;
    endfor;
    l=l·2;
  endwhile;
  return v;
end;
```

(b)

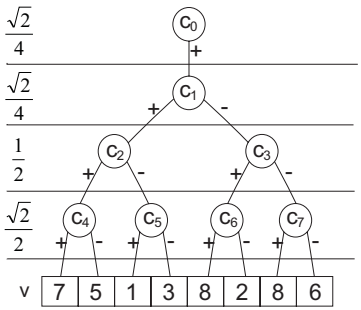
Fig. 2.3. Algorithms computing the Haar discrete wavelet transform (a) and anti-transform (b)

2.2.3 Querying wavelet coefficients

Observe that, each row of $\mathbf{W}$ (see Fig. 2.2) contains exactly $\log n + 1$ non-null values, therefore, each element of v, according to Equation 2.5, can be reconstructed by summing the weighted contribution of exactly $\log n + 1$ coefficients.

The mother wavelet function dilating strategy for defining the basis, impose an implicit hierarchy in the coefficients representing the wavelet decomposition. Therefore, they can be represented by a hierarchical structure, in particular by a perfect binary tree.

In Figure 2.4 (a) the tree representing the organization of the coefficients resulting from the transform of a vector of length 8 is depicted. Observe that,



(a)

INPUT c: a vector of length n representing the Haar discrete wavelet transform of a vector v;
i: an index between 0 and $n-1$;

OUTPUT s: the value v[i];

```
begin
  p = 1;
  s = c[0];
  inf = 0; sup = n-1;
  while (inf < sup) do begin
    med = (inf+sup)/2;
    if (inf ≤ med) then begin
      s = (s+c[p])/√2;
      p = 2p;
      sup=med;
    else begin
      s = (s−c[p])/√2;
      p = 2p + 1;
      inf=med+1;
    endif;
  endwhile;
  return s;
end;
```

(b)

Fig. 2.4. Hierarchical representation of the coefficients and weights (on the left) resulting from a wavelet transform of a vector v of size 8 (a) and algorithm for computing an equality query on the wavelet coefficients (b)

the coefficient c_0 affects all the elements in v, giving them a contribution equal to $c_0 \cdot \left(\frac{1}{\sqrt{2}}\right)^{\log n}$. All the other coefficients c_i $(1 \leq i < n)$ affects only the leaves of the subtree rooted at c_i, giving them a positive or negative contribution equal to $\left(\frac{1}{\sqrt{2}}\right)^{\log n - \lfloor \log i \rfloor}$, depending on whether the elements are leaves of the left- or the right-subtree, respectively, rooted at c_i.

From the hierarchical representation it is clear that the value $v[i]$can be reconstructed by traversing the tree from the root to the leaf $c_{\lfloor (n+i)/2 \rfloor}$, and by multiplying at each level the value of the traversed coefficient by a positive or negative constant, whose magnitude depends on the level and whose sign depends on the direction of the traversing. Therefore, an efficient algorithm can reconstruct a single value in time $\mathcal{O}(\log n)$. This algorithm is depicted in

Fig. 2.4 (b), where the coefficient tree is assumed to be represented by levels inside a vector c.

A very interesting property of the wavelet decomposition, which derives from the hierarchical characteristic of the decomposition, consists in the possibility to compute also sum range queries in time $\mathcal{O}(\log n)$. In fact, it can be shown that the sum of elements within a range $[l..h]$ depends only on the coefficients traversed from two paths starting from the root and reaching l and h. Thus, there is no need to reconstruct all the original values contained in the range $[l..h]$ in order to compute the answer to a sum-query on the same range. This is an advantage w.r.t. the other parametric techniques which, in order to avoid reconstructing all the values in a range of a query, entail either to compute the integral of the approximating function or to approximate the cumulative data distribution.

2.2.4 Wavelet coefficients thresholding

As the coefficients c representing the transformed vector v enable the exact reconstruction of v, no information has been lost, thus no summarization has been achieved. In fact, the coefficients are still represented by a vector of length n as the original data vector v. In order to summarize the data only the m coefficients of c whose "importance" is evaluated above a threshold are retained. The others are not stored and are usually implicitly assumed to be zeros. The number m depends usually on the desired compression ratio. The choice of the m coefficients to be maintained depends on a metric representing the approximation error, that has to be minimized. For example, the *SSE* could be minimized (see Equation 2.1). An important property shared by all the wavelet transforms, but only if they adopt an orthonormal basis, is that the *SSE* is minimized simply choosing the m coefficients with the larger magnitude (absolute value) [120]. This can be intuitively explained considering that the "importance" of the contribution of each coefficient to the reconstruction of the original data depends only on its value, as coefficients are all associated to vectors in the basis with the same unitary modulus. Therefore, maintaining the coefficients with larger magnitude enables maintaining the most important pieces of information.

However, different coefficient threshold strategies have been proposed, aiming at minimizing other error metrics, and they will be discussed in the following.

2.2.5 Multi-dimensional DWT

The Haar wavelet transform can be extended to the multi-dimensional data domains using two different methods, namely the *standard* and *nonstandard* multi-dimensional Haar transform. The multi-dimensional transform can be

obtained on the basis of the one-dimensional DWT. In the case of the standard transform, the multi-dimensional data are processed according to one-dimensional segment along each direction. Specifically, in the d-dimensional case, for the first dimension, all ranges of size $n \times 1 \times \cdots \times 1$, which represent one-dimensional arrays, are transformed according to the one-dimensional DWT. The resulting d-dimensional array is re-transformed by applying the DWT to alle ranges of size $1 \times n \times \cdots \times 1$, and so on, until the d-th dimension is processed. The nonstandard transform, works similarly, but apply only one DWT step at time along each dimension. That is, the first dimension is processed by applying the first step of the DWT, then the second dimension is applied the first step, and so on until the d-th dimension. Then, the process is recursively applied to the sub-array of size $(n/2)^d$ corresponding to the block $\langle [0..n/2-1], \ldots, [0..n/2-1] \rangle$.

For instance, consider a two-dimensional array D of size $n \times n$, and let $D[i, *]$ denote the i-th row and $D[*, j]$ denote the j-th column of the matrix representing D. According to the standard approach, first the one-dimensional DWT is applied to $D[0, *], D[1, *], \ldots, D[n-1, *]$, thus obtaining a new matrix D'. Then, the DWT is applied to $D'[*, 0], D'[*, 1], \ldots, D'[*, n-1]$, thus obtaining a matrix D" which correspond to the final matrix of wavelet coefficients. According to the nonstandard approach, only the first step of the DWT is applied to $D[0, *], D[1, *], \ldots, D[n-1, *]$, thus obtaining a new matrix D'. Then, only the first step of the DWT is applied to $D'[*, 0], D'[*, 1], \ldots, D'[*, n-1]$, thus obtaining a matrix D". Now, the same process is applied to the sub-array corresponding to the block $\langle [0..n/2-1], [0..n/2-1] \rangle$ of D", and so on, until the array to be processed is reduced to one only cell (a 1×1 matrix).

No one of the two techniques has ever been proved to result in better accuracy than the other (after the coefficient threshold), but it can be shown that the nonstandard one can be computed more efficiently.

The wavelet transform of a d-dimensional data array D, of size n^d, is a d-dimensional wavelet-coefficient array C of size n^d. Each coefficient of C, like in the one-dimensional case, is associated to a range of the data domain, which is called *support region*. In the one-dimensional case, the support region is a one-dimensional range (i.e., the range where the correspondent basis vector is not null), in the d-dimensional case, each cell of C gives a contributions to a d-dimensional range of values in D.

The support regions of each coefficient, in the multi-dimensional case, depends on the type (standard or nonstandard) of transform. The basis of the multi-dimensional wavelet transform, is a set of n^d d-dimensional arrays, each of size n^d and assuming non null values only in its support region. Each array of the basis is associated to exactly one coefficient of C. The support regions of the basis for a 4×4 data domain are depicted in Fig 2.5, for the standard (a) and nonstandard (b) transform. The support region of the coefficient $C[i, j]$ is that corresponding to the null null region of $W[i, j]$, which is partitioned in positive- and negative-contribution sub-regions.

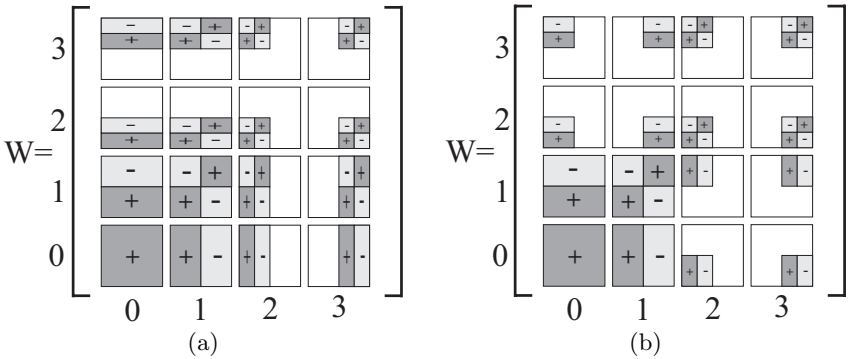


Fig. 2.5. Coefficient support region for multi-dimensional standard (a) and non-standard (b) Haar wavelet transform of a 4×4 two-dimensional array

2.2.6 Wavelet-based summarization techniques

The first work proposing the adoption of a wavelet transform to the context of approximate query answering was proposed by Matias et. al. in 1998 [99]. Their proposal consisted basically in applying the DWT to the cumulative distribution C, instead of the original vector D. According to their results, approximating C yields lower error rates than approximating D. Queries can be evaluated by accessing exactly two values on the approximate representation of C (in the one-dimensional case). They also found that the use of Haar basis yields approximations comparable to these yielded by the other techniques (i.e., histograms) which represented the state-of-the-art at that time. Instead, they found that DWT adopting linear basis (i.e., piecewise linear functions as basis) performed better, especially on range queries. Their results were probably due to the fact that the cumulative distribution is monotone nondecreasing and, intuitively, it can be locally fit better by linear than constant functions. They also proposed four heuristics for selecting the m coefficients which minimize a measure of the error on a query workload (they proposed three different measure of the error). The first heuristic consisted in selecting the m largest in magnitude coefficients. Observe that this deterministic approach is not still optimal, as their adopted basis is not orthonormal, the measure can be different from the *SSE* and the query workload can be different from the set of all the possible equality queries. The other proposed approaches consisted in choosing deterministically the $m' \neq m$ coefficients with largest magnitude, and then greedily selecting/unselecting those coefficients which yield the maximum decreasing/minimun increasing in the measure of the error, until the number of retained coefficients is equal to m. Finally, they proposed a straightforward extension to multi-dimensional contexts, by applying the standard multi-dimensional transform. This solutions entails computing the cumulative distribution of a multi-dimensional array, which can result huge

in volume and it could not fit in main memory. Another disadvantage of approximating the cumulative distributions consists in the inefficiency in query evaluation. In fact, in order to answer a range query, it is necessary to compute 2^d values, corresponding to the vertices of a d-dimensional range ($2^{d'}$ values, if d' dimensions if specified by the query) which entails the traversing of the coefficients tree along 2^d paths, while computing any range query on the decomposition of original data would entail the traversing of only 2 paths.

To overcome the problem of limited main memory, Vitter et al., in the same year proposed an I/O-efficient strategy to be adopted when disk accesses are needed in order to compute the wavelet transform. In [127] they proposed to partition the array in chunks representing blocks of cells and then storing each chunk into a secondary memory page. Indeed, their proposal was limited to a logical chunk, which assumes that cells are stored into a one-dimensional array according to a linear ordering[2] of the multi-dimensional space. According to this assumption, multi-dimensional array chunks are mapped to blocks of contiguous cells of the one-dimensional array. They showed that it is possible to compute the DWT over the multi-dimensional array by means of $\mathcal{O}(\frac{n^d}{B} \log_{M/B} \frac{n^d}{B})$ disk accesses, where n^d is the size of the multi-dimensional array to be transformed, M is the size of the main memory and B is the disk page size. They also presented a theorem which states that it is possible to reconstruct each of the original cells of the array in time $\mathcal{O}(\min\{\log n^d, m\})$, where m is the number of retained coefficients. Therefore, the complexity of answering range queries according to their proposal is $\mathcal{O}(2^d \cdot \min\{\log n^d, m\})$, as it is necessary to compute the approximate values of 2^d cells. Besides the algorithms regarding the decomposition and querying of the data, they proposed an improvement related to the accuracy. Specifically, they proposed to decompose the logarithm of the cumulative values, as this yields better approximations in terms of relative (instead of absolute) errors affecting range query estimations.

Even though [127] represents a step ahead in the study of the practical feasibility of the wavelet transform in the multi-dimensional context, it only scraped the wall representing the problems deriving from the management of physical multi-dimensional arrays. In fact, even in real cases, without applying to the high-dimensionality contexts, the multi-dimensional array could not fit also in secondary and even tertiary memory. Of course, the density of such an array would be very low (the size of the array, not the number of tuples, increases exponentially in the number of dimensions). However, the algorithms proposed in [127] do not take into account the sparsity of the array, and their complexity depends on n^d, the number of cells, even thought the majority of them contains null values.

A significant improvement was introduced in the work of 1999 [128], in which Vitter and Wang abandoned the idea of working on cumulative values,

[2] although it is not explicitly stated, from a figure in that paper depicting an example, it seems that they adopted the Z-ordering.

thus exploiting the sparseness of data. They proposed two algorithms performing the multi-dimensional DWT in time $O(\frac{N_z}{B}\log_{M/B}\frac{n^d}{B})$, where N_z is the number of non-null cells of the d-dimension al array of size n. These algorithms reduce dramatically the decomposition time, as they reduce the complexity of a factor equal to $\frac{n}{N_z}$, which in real cases can be several orders of magnitude large. The algorithms basically partition the d dimensions into groups such that the "slices" of the d-dimensional array built on the dimensions of each group can fit in main memory. Each slice is decomposed according to the standard multi-dimensional wavelet transform along the dimensions of the considered group. In order to limit the spatial complexity, which can exceeds $\mathcal{O}(N_z)$ as at each one-dimensional pass of the DWT the number of non-null wavelets coefficients can be larger than the original non-null values, they cut-off those wavelet coefficients which do not exceed a threshold. Indeed, as the standard decomposition is adopted, the coefficients are cut-off before reaching their final values, thus introducing an approximation in the transform. This approximation was not studied, neither theoretically nor experimentally, even though the authors stressed that this approximation is in the spirit of the data summarization. Another remarkable aspect of this work, deriving from working on raw data rather than cumulative values as well, consist in the range query evaluation algorithm they propose, which is optimal, as only two paths in the coefficients hierarchy (instead of 2^d) have to be traversed. A variant of the technique, working on cumulative values, was proposed by the authors in [130] together with Lim and Padmanabhan, where they proposed to compute the cumulative data on-the-fly for each slice of multi-dimensional array, just before decomposing it by means of the wavelet transform.

A different approach in transforming multi-dimensional data, which is based on the nonstandard Haar DWT, was introduced by Chakrabarti et al. in [19, 20]. By means of the nonstandard Haar DWT, they proposed to process, independently, each of the N_z non-null values and obtain $2^d - 1$ detail coefficients, in their final values, and 1 average coefficient, which is defined over a domain of size $\frac{n^d}{2^d}$ and needs to be further transformed. The selection of the m largest detail coefficients largest in magnitude can be safely accomplished on the fly (without approximation, as the choice is done according to their final value), and the remaining $\mathcal{O}(min\{\frac{n^d}{2^d}, N_z\})$ average coefficients are recursively decomposed. Their algorithm can run in time $\mathcal{O}(N_z \log N_z)$, which can decrease down to $\mathcal{O}(N_z)$ if data are organized in chunks. Despite this important improvement, the main contribution of [19, 20] is the application of the wavelet transform to a more general query approximation context. In fact, the authors described how to efficiently compute selection, projection, join and general aggregation operators in the reduced wavelet domain. Specifically, they proposed to compute equivalent operators in the wavelet domain and then to anti-transform the result. This solution represents a significant improvement in efficiency w.r.t. the other possibility, consisting in anti-transforming before computing the operators, as the the wavelet domain

consists of a compact set of data synopses, while the anti-transformed data could be significantly larger.

Further work dealing with wavelets has mainly regarded the investigation of effective strategy for the coefficient thresholding. The deterministic strategy based on keeping only the m largest in magnitude coefficients, although guarantees the lower *SSE* on reconstructed single values, can yield highly biased estimations on some regions of data, for which no coefficients could be kept, in favor of other regions which could be approximated with high accuracy. In order to avoid this problem, Garofalakis and Gibbons in [45, 46] introduced a new probabilistic thresholding scheme based on the idea of randomly rounding coefficients, so that the expected value of the rounded value coincides with the original one. More specifically, the most important coefficients are deterministically retained, while the other are rounded either up to a larger rounding value to be retained or down to zero. To this aim, each non-zero coefficient c_i is assigned a random value C_i such that C_i takes the value λ_i (and it is retained) with probability $y_i = \frac{c_i}{\lambda_i}$ and takes the value zero (and it is discarded) with probability $1 - y_i$. Then, by flipping an independent y_i-biased coin (i.e., a coin with probability of success equal to y_i and independent on the other flips), the type of rounding for c_i is chosen. It is easy to verify that C_i has expected value $E[C_i] = c_i$ and variance $Var(C_i) = c_i \cdot (\lambda_i - c_i)$. A coefficient c_i can be deterministically chosen to by retained by posing $\lambda_i = c_i$. The authors showed that this thresholding scheme yields unbiased estimation both for equality and range queries. The values λ_i can be chosen in order to minimize different error metrics. A quantization on coefficient is adopted (i.e., each λ_i can be a multiple of a fixed value $1/q$). The authors presented the dynamic-programming-based algorithms for minimizing the expected mean-squared error and the maximum relative error. In addition, they proposed a thresholding scheme not based on rounding but which aims at minimizing the maximum normalized bias. The time complexity of proposed algorithms is $\mathcal{O}(N_z q^2 m \log(qm))$, where m is the number of the retained coefficients. By means of experiments they showed that the new thresholding strategies, in particular those minimizing the maximum relative error and the maximum normalized bias perform better than the deterministic thresholding strategy on one-dimensional synthetic and real-life data sets. Although the authors described how to extend the framework to the multi-dimensional application setting, no experimental results testing the technique in that context have ever been presented. In [34], an ϵ-approximate algorithm based on the same probabilistic thresholding strategy was proposed, working in time $\mathcal{O}(N_z \log q \cdot min\{\frac{\log N_z \log R}{\epsilon}, mq\})$, where R is proportional to the maximum absolute value of wavelet coefficients. The problems possibly arising from "bad" coin-flip sequences and from the quantization in the coefficient thresholding scheme proposed in [45, 46] has been overcome by Garofalakis and Kumar in [47, 48]. It these works they introduced a deterministic wavelet thresholding scheme, implemented by means of a dynamic-programming-based algorithm, which can minimize the maximum relative/absolute error in the estimations. The complexity of the proposed

algorithm is $\mathcal{O}({N_z}^2)$ in the one-dimensional case. The extension to the d-dimensional case becomes infeasible, as the complexity becomes $\mathcal{O}\left({N_z}^{2^d}\right)$. To reduce the complexity, they introduced an ϵ-approximate algorithm, with complexity $\mathcal{O}(\frac{2^{2^d}}{\epsilon} \cdot N_z \log N_z)$.

2.3 Histograms

According to Beniger and Robyn [8], the word *histogram* was coined in in 1895 by Karl Pearson on the basis of the ancient Greek words $\iota\sigma\tau o\varsigma$, which means *mast*, and $\gamma\rho\alpha\mu\mu\alpha$, which means *graphical symbol*. By means of this word Pearson intended to denote the class of graphical representations, commonly adopted in statistics, based on the use of vertical lines of different height for representing the values of some measure associated to different items. However, the use of histograms precedes their nicknaming. In fact, the first known use of histograms dates back to 1833, when the French statistician André Michel Guerry adopted this form of graphical representation (obviously without calling it *histogram*) for depicting crime data in his book *Essai Sur La Statistique Morale de la France*. In 1846, the Belgian statistician Adolphe Quetelet in his book *Lettres sur la Théorie des Probabilités, Appliquée aux Sciences Morales et Politiques* adopted a symmetric histogram approximating a normal distribution.

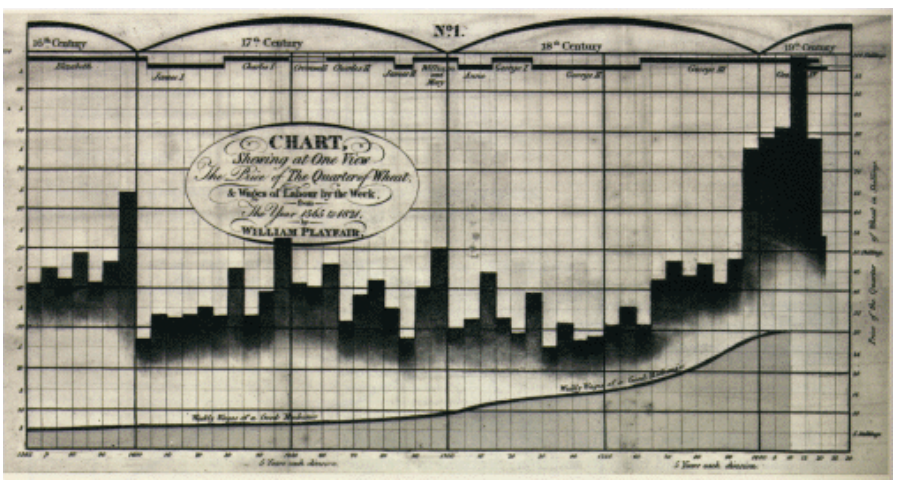


Fig. 2.6. Bar chart from *Letters on our agricultural distresses* by Playfair, 1822

A similar form of graphical representation are *bar charts*, which are based on bars, instead of lines, having a width besides a height as the lines of his-

tograms. However, the two forms of representation are strictly related, and are often confused. The use of bar charts is anterior to the introduction of histograms. In fact, in 1786 the Scottish political economist William Playfair adopted bar charts for the first time in his book *The Commercial and Political Atlas*, in which the commercial exchanges between Scotland and other countries were analyzed. A known practical and successful use of bar charts was that by Florence Nightingale, a English nurse who organized the sanitary services during the Crimea war and who founded the modern nursery assistance. She adopted bar charts in 1859 to compare mortality in the peacetime army to that of civilians, convincing the government to improve army hygiene.

Even though histograms are mainly considered a form of graphical representation, they are based on tables representing some data distribution. The idea of using this kind tables, without graphically representing them, was firstly introduced by the English statistician John Graunt, generally considered to be the founder of the science of demography. In 1662 he published the book *Natural and Political OBSERVATIONS Mentioned in a following INDEX, and made upon the Bills of Mortality* in which the age and causes of mortality in London during the bubonic plague years (from 1592 to 1636) were analyzed. In this book, he presented several tables analyzing the number of deaths grouped by several characteristics, such as the cause and the year. The English government was interested in his studies, as they represented the basis for the first statistically-based estimation of the population of London and of England (thus, it was possible to have an idea about the possible incomes due to taxes!).

Without recalling all their adoptions throughout the last three centuries, it is just worth remarking that histograms, since their origins, have been widely adopted to represent qualitatively data distributions.

Their adoption in databases was explicitly proposed for the first time in 1980 by Kooi, who in his PhD thesis [84] proposed to adopt histograms for approximating attribute value frequency distributions for supporting the selectivity estimation in query optimization. However, the idea of approximating the attribute frequency distribution by means of tables had been already proposed one year before by Merret and Otoo in [101]. Since the early 1980s, several histogram-based data approximation techniques have been proposed.

In order to describe the main ones, some basic definitions will be given.

Definition 2.1 *Given a one-dimensional data distribution D over the domain of values V, consisting of n entries represented by pairs $\langle v_i, f_i \rangle$, where $v_i \in V$ and $f_i \in [0,1]$ represents the frequency associated to v_i, $s_i = v_i - v_{i+1}$ $(0 < i < n)$ will be said* spread *of v_i and $a_i = f_i \cdot s_i$ $(0 < i < n)$ will be said* area *of v_i. The set of the frequencies $\{f_i | 0 < i \leq n\}$ will be denoted as F, the set of spreads $\{s_i | 0 < i < n\}$ will be denoted as S, and the set of areas $\{a_i | 0 < i < n\}$ will be denoted as A (spred and area are not defined for the n-th value).*

The definitions of spreads and areas make sense only if a total ordering among the values in V can be defined. In the one-dimensional case it is always possible defining a total ordering on V, which can be a natural ordering depending on the semantics of values in V or an ordering induced by an enumeration. In the d-dimensional case, where v_i is a d-tuples, a total ordering could make no-sense. Therefore, in the following, v_i will be assumed as simple values, thus focusing the attention on one-dimensional data distributions. The generalization to the multi-dimensional case will be proposed later.

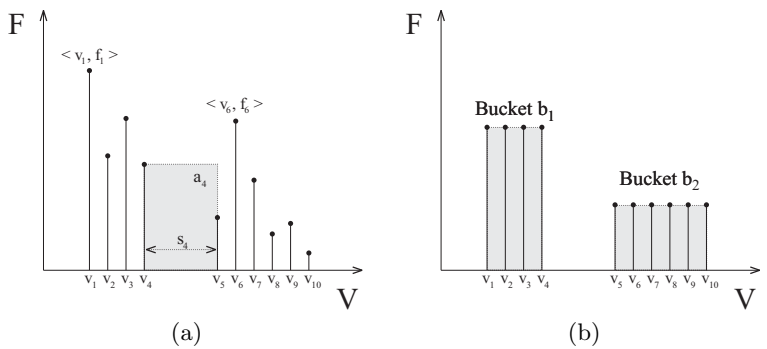


Fig. 2.7. Histogram build on a data distribution over ten values and summarization of data distribution in two buckets

In Fig. 2.7 (a) an histogram built on a set V consisting of ten values is depicted. In order to achieve data summarization, the set of pairs $\langle v_i, f_i \rangle$ is partitioned into groups of pairs called *buckets*, and for each bucket a summarizing information is stored. That is, an information summarizing both the values and the frequencies associated to the pairs in the same bucket must be stored. The most adopted strategies impose to partition the data distribution so that if a bucket contains two entries $\langle v_i, f_i \rangle$ and $\langle v_i, f_i \rangle$ then it also contains all the entries $\langle v_k, f_k \rangle$ with $v_i < v_k < v_j$. Then, for each bucket the range of its values (the minimum and the maximum value) and the sum of its frequencies are stored. Therefore, each bucket b_k can be considered as a pair $\langle \rho_k, sum_k \rangle$, where $\rho_k = [lb_k..ub_k]$ is a range defined on the value domain V, with lower bound lb_k and upper bound ub_k, and $sum_k = \sum_{i=lb_k}^{ub_k} f_i$.

2.3.1 Querying histograms

The estimation on the basis of the bucket information are usually made according to two assumptions, the first on the values and the second on the frequencies:

1. *Continuous value assumption* (*CVA*): it is assumed that any possible value between the minimum and the maximum values in the bucket are present with non-null frequency in the bucket itself;
2. *Uniform frequency distribution assumption* (*UFD*): it is assumed that each value in the bucket is associated the same frequency value.

Indeed, other strategies for estimating the value distribution have been proposed in literature. Specifically, in [108] the *point value assumption* is introduced, according to which only one value is present in the bucket (usually the lowest among the values actually contained in the bucket). In [109], a new assumption is introduced, namely the *uniform spread assumption*: values are assumed to be equidistant inside the bucket. In order to exploit this assumption, it is necessary to store the number of distinct values actually contained inside each bucket. However, according to theoretical results, this assumption does not lead to more accurate estimations than the CVA. In fact, Buccafurri et al. in [12] proved that even though both the number c of distinct values associated to non-null frequencies and the sum of frequencies s were known, an unbiased estimator of range queries would not take into account c but only s (instead, c affects the variance of the estimator).

In Fig. 2.7 (b) a partitioning of the data distribution in Fig. 2.7 (a) is depicted. The original data distribution is summarized by two buckets, the first one summarizing the first 4 pairs, and the second one summarizing the last 6 pairs. In this case, the two buckets are $b_1 = \langle [v_1..v_4], \sum_{k=1}^{4} f_i \rangle$ ($lb_1 = v_1$, $lb_1 = v_4$) and $b_1 = \langle [v_5..v_{10}], \sum_{k=5}^{10} f_i \rangle$ ($lb_2 = v_5$, $lb_1 = v_{10}$). According to the *UDF* and the *CVA* assumptions, each frequency f_i of $v_i \in \rho_k$ is approximated by the same value $\tilde{f}_k = \frac{sum_k}{|\rho_k|}$, where $|\rho_k| = ub_k - lb_k + 1$ is the size of the range ρ_k (corresponding to the estimated number of values in ρ_k according to the *CVA*). When estimating a range query, i.e. the sum of frequencies associated to values inside a range r, there are three possible cases as regards the contribution of each bucket b_k to the estimate:

1. $\rho_k \cap r = \emptyset$: the bucket gives no contribution to the query;
2. $\rho_k \cap r \equiv \rho_k$: the bucket is completely contained inside the query, and gives an exact contribution to the query;
3. $\rho_k \cap r \subset \rho_k$: the contribution of the bucket must be estimated. To this end, according to the *CVA*, the number of values in b_k and r is estimated as $|\rho_k \cap r|$, each of these values is associated an estimate $\tilde{f}_k = \frac{sum_k}{|\rho_k|}$, therefore the estimate contribution of b_k is $|\rho_k \cap r| \cdot \frac{sum_k}{|\rho_k|}$.

The three cases can be unified considering that $\frac{|\rho_k \cap r|}{|\rho_k|} = 0$ when $\rho_k \cap r = \emptyset$ and $\frac{|\rho_k \cap r|}{|\rho_k|} = 1$ when $\rho_k \cap r \equiv \rho_k$. The simple general algorithm for evaluating range queries on a histogram, represented by set of buckets grouping contiguous values and on the basis of the CVA and UFD assumptions is depicted in Fig. 2.8. Observe that the errors affecting the estimations depend exclusively on the buckets which partially overlap the query range, as they give an

approximate contribution to the estimation, while the other gives an exact contribution (0 case 1, sum_k in case 2). In the one-dimensional case, at most two buckets can gives an approximate contribution to the query estimate.

INPUT H:a histogram represented as a set of β pairs $\langle \rho_i, sum_i \rangle$ $(1 \leq i \leq \beta)$ approximating the a frequency value distribution on a set V;
r: a range defined on the same domain V;

OUTPUT s: the estimate of the sum of frequencies associated to values inside r;

begin
  s = 0;
  for $(i \in \{1..\beta\})$ **do**
    $s = s + |\rho_i \cap r| \cdot \frac{sum_i}{|\rho_i|}$;
  return s;
end;

Fig. 2.8. General algorithm for answering range queries on a histogram

The complexity of the algorithm is $\mathcal{O}(\beta)$, where β is the number of buckets, as a linear scanning of the buckets is performed, and for each bucket a constant number of operations is computed.

The accuracy of the estimates, of course, depends on the quality of the partition of the data distribution on which the histogram is based. Denoting with $\langle v_i, \tilde{f}_i \rangle$ the estimated data distribution on the histogram, the sum of squared errors $SSE = \sum_{i=1}^{n} (f_i - \tilde{f}_i)^2$ is a widely adopted metric for the errors affecting estimates on the histogram. The aim of a histogram construction algorithm is that of minimizing the measure of the errors within a given storage space bound, i.e. a limited number β of buckets. In [74] Jagadish et al. have proposed a dynamic-programming-based algorithm which can compute the V-Optimal histogram, i.e. the histogram minimizing the *SSE* of estimates, in time $\mathcal{O}(\beta \cdot n^2)$ in the case of one-dimensional data. Observe that, the *SSE* depends on the absolute error of equality query estimates. In many context it would more useful to minimize the relative error of estimates, instead of the absolute one. In [55] Guha et al. proposed a dynamic-programming-based algorithm for constructin a one-dimensional histogram which minimize the maximum relative error of equality query estimates, running in time $\mathcal{O}(n\beta \log^2 n)$. Observe that minimizing either the *SSE* of a histogram of the maximum relative error on equality queries, does not give non-trivial guarantees on errors affecting range queries estimates. In [50] Gilbert et al. proposed a pseudo-polynomial algorithm for minimizing the sum of squared errors on range queries for one-dimensional

data. Specifically, they defined the *SSE* for range queries as

$$SSE' = \sum_{a \leq b} \left(sum([a..b]) - \widetilde{sum}([a..b]) \right)^2 ,$$

where $[a..b]$ are all the possible ranges on which queries can be defined, and $\widetilde{sum}([a..b])$ is the estimation of $sum([a..b])$ on the histogram. The complexity of the proposed algorithm is $\mathcal{O}\left(n^4 \beta(sum([1..n]))^3\right)$.

2.3.2 A Histogram Taxonomy

In [109] a taxonomy of histograms was proposed by Poosala et al. They studied all the histogram aspects, enumerating the properties characterizing a wide class of histogram construction techniques. Their taxonomy included all the previously proposed histogram types and some others that they showed to represent a significant improvement w.r.t. the first ones. The following histogram properties were isolated in order to define the taxonomy:

- *Sort Parameter*: it is a parameter assigned to each entry $\langle v_i, f_i \rangle$ of the data distribution according to some function of v_i and f_i. Value (V), frequency (F) and area (A) were indicated as the most significant possible sort parameters;
- *Source Parameter*: it is the parameter which represents the most critical aspect of the data distribution in the estimation problem. Spread (S), frequency (F) and area (A) were indicated as source parameters of interest;
- *Partition Class*: this parameter specifies possible restrictions on the bucket composition. For instance, *serial* histograms [70] require that entries of the data distribution in the same bucket, ordered according to the sort parameter, are contiguous and *end-biased* histograms [70, 72] require that all but one buckets are singleton, i.e. they contain one only entry of the data distribution;
- *Partition Constraint*: this parameter represents a mathematical constraint on the source parameter and uniquely identify a histogram within the class of histograms defined by sort and source parameters and partition class. For instance, *equi-sum* partitioning constraint impose that the sums of source parameter in all buckets is the same.

The most adopted sort parameter, since the first proposals [84, 101], has been the value (V). The source parameter adopted by both [84] and [101] was the spread (S), but later, more effective techniques have focused the attention on the frequency (F) as source parameter. As regards the partition class, the majority of techniques falls in the class of serial histograms. The parameter on which the histogram-based techniques mostly differ is the partition constraints. Besides the *equi-sum* one, other proposed partition constraints have been *V-Optimal* [72], which requires to minimize the variance of subset of data distribution inside buckets, *Spline-Based* [109], which requires to

minimize the maximum absolute difference between a source parameter value and the average of source parameter values in buckets, *Max-Diff* [109], which define β buckets by separating the entries of the data distribution, ordered according the sort parameter, in correspondence of the $\beta - 1$ pairs with the $\beta - 1$ larger absolute difference in the source parameter, and *Compressed* [109], which stores separately the k entries of the data distribution and the remaining ones are partitioned according to an equi-sum histogram.

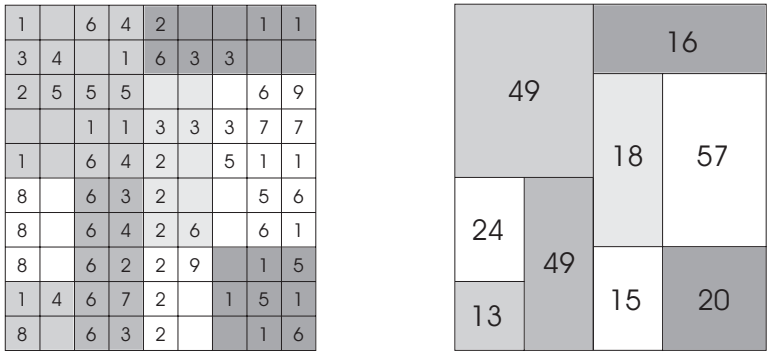
The reason why the majority of techniques adopt value as sort parameter jointly with serial partition class resides in the estimation efficiency. In fact, range queries are posed specifying a range on the value domain V. Therefore, if a bucket did not contain values which are contiguous in their domain and no implicit information were available on the set of values contained in each bucket, in order to make feasible the range query estimations, the list of all the values inside each bucket should be stored (a summary information of non-contiguous values, such as their average, would be insignificant). Therefore, summarization would regard only the frequencies, and the size of the histogram would be $\mathcal{O}(n)$, which is the same order of magnitude of the original data distribution size. Moreover, computing an estimation would require to access all the n values partitioned among the β buckets, in order to verify whether each of them is contained inside the query range. Thus, the estimation complexity would coincide with that of computing the exact answer to the query on the original data distribution.

2.3.3 Multi-dimensional Histograms

In the case of multi-dimensional data, only histograms with value as sort parameters have been studied. In this case, values are d-tuples corresponding to points of the d-dimensional data domain. Each point is associated a value f_i. The classical histogram-based multi-dimensional data summarization techniques define a partition of the data domain, consisting of hyper-rectangular blocks, and for each block its boundaries and the sum of frequencies associated to the points inside the blocks are stored. That is, the buckets can be represented as in the one-dimensional case by means of pairs $\langle b_i, sum(b_i)\rangle$, where b_i are non-overlapping multi-dimensional blocks.

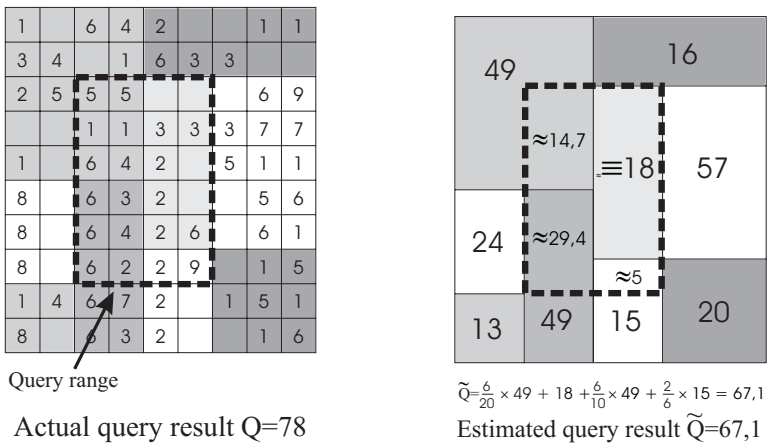
In Fig.2.9 a two-dimensional data distribution (a), represented as a two-dimensional array, and a possible histogram approximating it (b) are depicted. The basic algorithm for querying multi-dimensional histograms is the same as that for one-dimensional histograms.

In Fig.2.10 the evaluation of a query on the data distribution of Fig. 2.9 is represented. Observe that, unlike the one-dimensional case, where at most two-buckets can give an approximate contribution to a range query, in the multi-dimensional case it is not possible to give a non-trivial bound on the buckets partially overlapping the query, and thus yielding and approximation in the query estimate. From this consideration, it emerges that the problem of effectively approximating a data distribution, possibly giving error guarantees



(a) (b)

Fig. 2.9. A two-dimensional data distribution (a) and a histogram approximating it (b)



(a) (b)

Fig. 2.10. Range query on a two dimensional data distribution (a) and its estimation on a histogram (b)

on the quality of estimation, is much harder in the multi-dimensional case than in the one-dimensional one.

As regards the partitioning of data domain, in the multi-dimensional case three classes of partitions can be defined:

- *Arbitrary partition*: no restriction on the arrangement of buckets is imposed (Fig 2.11 a);

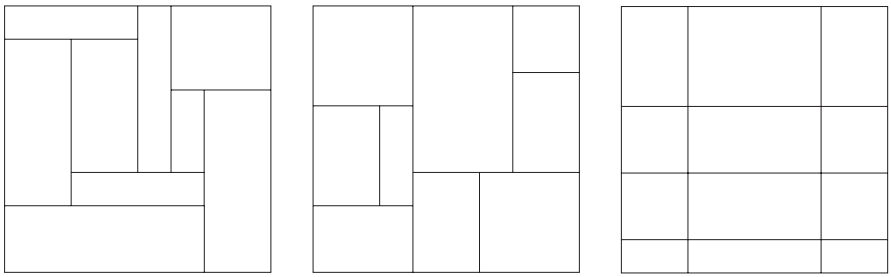


Fig. 2.11. Examples of abitrary (a), hierarchical (b) and grid-based (c) partitions of a two-dimensional data domain into 9 buckets

- *Hierarchical partition*: the partition can be obtained by recursively splitting a block, starting from the block representing the whole domain, along one dimension into two sub-blocks (Fig 2.11 b);
- *Grid-based partition*: buckets are represented by the cells of a grid defined on the data domain(Fig 2.11 c).

The arbitrary class is a super-class of the hierarchical one, which in turn is a super-class of the grid-based one.

The construction of an arbitrary partition in the multi-dimensional case minimizing an error metric such as the *SSE* has been proved in [104], by Muthukrishnan et al., to be an $\mathcal{NP}$-hard problem. Therefore, all the approaches proposed in literature, dealing with more than one dimensions, were restricted to studying the hierarchical and grid-based sub-classes of partitions.

2.3.4 Histogram-based summarization techniques

The first histogram based summarization techniques, [84] and [101], proposed to partition the attribute value domain in *equi-width* ranges, storing the sum of entry frequencies in each range. Therefore, these proposals adopted value as sort parameter, spread as source parameter, serial partition class (contiguous values are summarized in the same bucket) and equi-sum as partition constraint (the sum of the entry spreads in each bucket, corresponding to the width of the bucket, must be the same). In particular, in [101], Merret and Otoo proposed to summarize a multi-dimensional data distribution by means of tables obtained partitioning the data domain according to a grid, whose cells have equal width along each dimension. Equi-width histogram represented a dramatic improvement over the uniform distribution assumption for the overall value set, which had been mainly adopted until the end of 1970s. Thus, histograms were quickly adopted by the Ingres DBMS [85, 121] in its commercial version, and later by other DBMSs as well.

In 1981 Christodoulakis, in his PhD thesis [27], suggested to adopt variable-width tables based on the *maximal difference* criterion, aiming at

summarizing together contiguous values with similar associated frequency. According to this criterion, the difference between the maximum and the minimum frequency in the same bucket must be within a given threshold. This criterion minimizes the estimation error on certain values rather than the average error on all values, and it seems most appropriate when queries are not uniformly spread over all attribute values.

Another milestone in the history of histogram-based summarization techniques is represented by the PhD thesis of Piatetsky-Shapiro [107], wherein he proposed the *FASTSCAN* system for query optimization. The system introduced a new class of histograms, based on the *equi-height* criterion (known also as *equi-depth*), which he showed to outperform the classical equi-width histograms. In [108], a work in joint with Connell, the technique is described with theoretical analysis of the estimation error in the worst and average cases. The authors stressed that the error can be arbitrarily reduced by increasing the number of buckets (indeed, this is rather expected). The equi-height histogram, according to the previously described taxonomy, adopts value as sort parameter, frequency as source parameter, belongs to the serial partition class and adopts equi-sum as partition constraints (i.e., the sum of the source parameter, the frequency, of entries inside each bucket must be the same). Although the *equi-heigth* partitioning strategy is possibly superior to the *equi-width*, in terms of estimation accuracy, the construction of an *equi-heigth* histogram is more expensive. In fact, before choosing the boundaries of β buckets, $\beta - 1$ quantiles of the data distribution must be computed, and this operation could require multiple scans of data. To achieve a reduction in complexity, the authors described how to efficiently construct the histogram on the basis of a sample of the data distribution. Theoretical bounds on the difference between the "approximate" equi-height histogram obtained by means of a random sample of data and the exact equi-height histogram (i.e., the histogram built on the actual data distribution and containing the same sum of frequencies in all buckets) were provided by Chaudhuri et. al. in [23]. Besides the sampling, several works related to the efficient computation of quantiles could be exploited as support to the equi-height histogram construction. For instance, in 1985, Jain and Chlamtac proposed the P^2 algorithm [76], which efficiently (i.e., by means of a single scan of data and with constant auxiliary space) computes approximate quantiles. In [112] Raatikained proposed a generalization of the P^2 algorithm which enables the simultaneous efficient estimation of several quantiles. More recently, algorithms for efficiently computing approximate quantiles with approximation bounds were proposed by Alsabti et. al. [5] and Manku et al. [97].

In [79] Kamel and King proposed a technique for partitioning multidimensional data by means of texture analysis. They did not talk explicitly of histograms, but their technique can be actually considered a histogram-based one. To partition the data space into cells, their techniques aims at obtaining hyper-rectangles containing homogeneous data distribution, because, as the authors stressed, they can be summarized easily without producing large es-

timation errors. To this end, the data domain is first partitioned into cells by means of a grid. Then, for each d-dimensional cell its homogeneity is evaluated as $\sum_{i=1}^{d}\sum_{j=1}^{n_i}(marg_i[j]-\mu_i)^2$, where $marg_i[j]$ it the j-th value of the marginal distribution (see Section 1.5) on the i-th dimension of the cell, n_i is the number of values that the cell spans along the i-th dimension, and μ_i is the average of $marg_i$ values. Adjacent cells with similar evaluated homogeneity are subsequently merged together. According to the authors, buckets with similar function values have also similar texture. The merging of adjacent cells suggests a form of agglomerative clustering which takes into account either the proximity and pattern similarity of data. This has been the first approach proposing to partition a multi-dimensional data distributions according to a bottom-up strategy.

The first work talking explicitly of multi-dimensional histogram is [103], by Muralikrishna and DeWitt, who in 1988 proposed and extension of the equi-depth histogram, introduced in [107, 108], to the multi-dimensional case. Their proposal in constructing a d-dimensional histogram entails the construction of several one-dimensional equi-height histograms along each dimension separately. Specifically, at the first step, a number of bucket is created by defining a one-dimensional equi-height histogram w.r.t. the first dimension. To this end, the marginal distribution w.r.t. the first dimension can be considered. At the following steps, for each bucket previously defined, tuples inside it are further partitioned according to an equi-height histogram w.r.t. the second dimension, and so on.

In [69] Ioannidis and Christodoulakis studied the errors yielded by uniform frequency assumption on the estimation of size of the result of multi-join queries. They proved an exponential growth of the error with the query dimension (i.e., number of joins) in the worst case. The results referred to a restricted class of queries, namely multi-join queries with only one attribute participating in joins per relation. Even though this was not a work strictly related to histogram construction techniques, it was the starting point for an important series of innovative works related to histograms. In fact, a very important aspect of this work have been the study of the error obtained by storing exactly the highest frequencies and using the uniform frequency assumption for all the rest of frequencies, which was shown to dramatically reduce the errors in estimations. The work [69] posed his bases on some studies regarding the related problem of block access estimation by Christodoulakis [29, 31] (see Section1.1).

The ideas proposed in [69] were the preamble to the work of 1993 [70], where Ioannidis and Christodoulakis introduced the serial and biased (low-, high- and end-biased) classes of histograms, proving that for single- and multi-join queries over a single attribute, the optimal histogram has the frequency as sort parameter. For the first time, in the context of selectivity estimation, it was proposed construct an histogram by adopting frequency instead of value as sort parameter, that is it was proposed to group entries with contiguous frequencies in the same bucket, instead of contiguous values. Indeed, a similar

idea had been already introduced in [124] by Vander Zanden et al., in [98], in the context of the block access estimation problem (see Section 1.1).

A generalization of [69] to multi-join queries over multiple attributes was provided in [71], where Ioannidis proved that serial histograms with the frequency as sort parameter are always optimal in the case of maximum size of the join result and in the case of minimum size when the attribute value independence assumption holds. In addition, the author pointed out that the expected value of join sizes, considering every possible frequency distribution, coincides with the estimation yielded by any histogram. Therefore, it is only meaningful to compare various techniques in extreme cases, as in the average cases all the histograms behave identically. Despite the theoretical importance of the work, in practice serial histograms on sort parameters different from values are inefficient to be queried, as remarked in Section 2.3.2.

In [72], together with Poosala, Ioannidis introduced more practically feasible ideas than those proposed in [70] and [71]. They introduced the *V-Optimal* histogram for the first time, showing that minimizing the variance of absolute error on the estimation of the frequency of single values is equivalent to minimize the error on a self-join. Therefore, the V-Optimal coincides with a serial histogram with frequencies as sort parameter. However, the authors themselves pointed out that the serial histogram with frequency as sort parameter is not feasible in practice, both for computational and spatial efficiency. Therefore, the authors recommended to adopt end-biased histograms, which are a particular case, efficiently manageable, of serial histograms. In end-biased histograms, the data distribution entries with the highest and the lowest frequencies are stored each in a singleton bucket, while the other entries are all stored in a single bucket.

In 1997, after about ten years of research inactivity in the study of multi-dimensional histograms, one of most known multi-dimensional data summarization technique, namely *MHIST-p*, was introduced in [111] by Poosala and Ioannidis. The *MHIST-p* histogram is built by a multi-step algorithm which, at each step, chooses the bucket which is evaluated as the most in need of partitioning, and partitions it into p new sub-buckets along one of its dimensions. The bucket to be partitioned is selected and partitioned on the basis of its marginal distributions. Several partition constraint can be adopted by *MHIST-p* to select and split the bucket which is the most in need of partitioning. Authors found that *V-Optimal* and *Max-Diff* are the most effective ones when value is adopted as sort parameter and area as source parameter with the serial partition class (all these parameters refer to the marginal distributions, and not to the joint frequency distribution). Therefore, they chose *Max-Diff* as it is more efficient to be computed. Thus, according to *Max-Diff*, after computing the *marginal distributions* (see Section 1.5) $marg_1(b), \ldots, marg_d(b)$ 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 with the largest difference w.r.t. every other pair of adjacent values in any other marginal distribution of any other block. Then, b is split along

the dimension i into p new buckets. The authors found that $p = 2$ provides the best results. Therefore, when not differently specified, *MHIST* refers to *MHIST*-2 adopting the *Max-Diff* partition constraints on marginal distributions. In *MHIST* for each bucket its *MBR* is stored (i.e., the smallest range containing all the entries in the bucket).

In [3], Acharya et al. introduced *Min-Skew*, a technique proposed in the context of selectivity estimation in two-dimensional spatial data sets. Indeed, this technique is suitable for approximating general multi-dimensional data distribution and can be viewed as a refinement of *MHIST*. Basically, *Min-Skew* 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: it tries all possible splits along every dimension of every block, and evaluates how much the *SSE* of the marginal distribution along the splitting dimension is reduced by the split. Then, the block b and the splitting position $\langle dim, pos \rangle$ are chosen 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 other $SSE(marg_i(b'))$ (where $i \in [1..d]$) which could be obtained by performing some split along i. Also in *Min-Skew*, like in *MHIST*, buckets boundaries are represented by means of their *MBR*s.

In [44] Furtado and Madeira proposed a technique, called *Summary Grids* (*SGRID*), which for the first time since the work [79] suggested again an opposite approach to this based on iteratively partitioning the data domain in a top-down fashion. In fact, they proposed to build buckets by aggregating data in a bottom-up fashion which aims at coalescing neighbor homogeneous data. Starting from a point, a bucket is enlarged towards one of a set of possible directions until a measure of the bucket homogeneity exceeds a threshold. Outliers are summarized apart, in order to not affect bucket homogeneity. Even though this technique aims at avoiding local homogeneous regions to be prematurely split, as it could happen in *MHIST* or *Min-Skew*, its greedy strategy could enable the creation of small homogeneous buckets which could later prevent the creation of larger homogeneous buckets (under the hypothesis that buckets can not overlap).

2.3.5 Non-classical Histograms

Histogram-based summarization techniques described so far base their strategy on the partitioning of the overall data domain into non-overlapping hyper-rectangular blocks, built by an initial elaboration on the whole data distribution, usually by means of an iterative top-down partitioning strategy. However, some different approaches have been proposed in the last few years, differing from the classical histogram-based techniques for different aspects. Specifically, some techniques exploit mathematical transforms in order to enhance the histogram construction, some techniques use an approach driven by the

query workload in histogram construction, some techniques allow buckets to overlap and other techniques aim at improving the intra-bucket estimation accuracy by adopting a more complex representation of data inside buckets than the a simple summary information which is adopted in estimating query answers on the basis of the uniform distribution assumption. Some of these techniques will be briefly described in the following.

In 1936, Eckart and Young, proposed a method for approximating a matrix with another one of lower rank [37]. Their idea, based on the exploitation of the *Singular Value Decomposition* (*SVD*) of a matrix, was similarly proposed to approximate two-dimensional data distributions in [111] by Poosala and Ioannidis. The *SVD* of a real matrix $\mathbf{D}$ of size $m \times n$ is any factorization of the form

$$\mathbf{D} = \mathbf{U} \times \mathbf{\Sigma} \times \mathbf{V}, \tag{2.6}$$

where $\mathbf{U}$ in an $m \times n$ orthogonal matrix, $\mathbf{\Sigma}$ is a $n \times n$ diagonal matrix with non-negative values, and $\mathbf{V}$ is an $n \times n$ orthogonal matrix. The values on the diagonal of $\mathbf{\Sigma}$ are said *singular values* of $\mathbf{D}$, and will be denoted as σ_i (i.e., σ_i is the value of $\mathbf{D}$ at the row i and column i). Denoting by U^i the i-th column of $\mathbf{U}$ and by V_i the i-th row of $\mathbf{V}$, Equation 2.6 can written as

$$\mathbf{D} = \sum_{i=1}^{n} \sigma_i \cdot U^i \times V_i. \tag{2.7}$$

In order to achieve data summarization, the smallest singular values are approximated by zero, thus, there is no need to store the columns of $\mathbf{U}$ and the rows $\mathbf{V}$ which gives a null additive contribution to $\mathbf{D}$ according to Equation 2.7. The other vectors, i.e. those corresponding to the k largest singular values, are approximated by means of k distinct one-dimensional histograms. The estimation can be performed on stored histogram adopting Equation 2.7. However, for an equality query all the k one-dimensional histograms must be adopted. Therefore, the estimation can be inefficient, especially considering that for a range query several point values must be estimated, unless the singular value decomposition is applied to the cumulative distribution instead of the original one. This technique represents an effort in reducing the dependency between attributes, but it is limited to the two-dimensional case. It could be interesting trying to define more sophisticate thresholding strategies for choosing which singular values and related vectors retain in order the optimize some objective function, as it was made in [45, 46, 47, 48] for wavelet-based summarization techniques.

A technique which works on general multi-dimensional distribution and aims at reducing correlation among dimension was proposed by Deshpande et al. in [35]. The idea inspiring their work is that the full correlation among the attributes of a relation is unlikely to hold as well as their full independence. Thus, they proposed to reduce the data dimensionality by means of *statistical interaction models*. In particular, they adopted the *decomposable*

models, a subset of *log-linear models* which have the advantage to enable query estimation by means of a closed formula. The decomposed model is a set $\mathcal{M} = \{S_1, \ldots, S_k\}$ where S_i is a subset of the relation attributes $\{X_1, \ldots, X_d\}$ (the subset are not required to be disjoint), which can be represented by a graph having a node for each attribute X_i and an edge between each pair of nodes corresponding to a pair of attributes inside the same S_i. This graph must be chordal, that is any cycle of length greater to three has a shortcut between any two non-adjacent nodes. The model is built so that the attributes within the same subset S_i can be strongly correlated, but groups of attributes belonging to different subsets are weakly correlated. The data distribution is then projected along the attributes of each subset separately, thus obtaining k data distributions with reduced dimensionality w.r.t. the original data distribution. Each new low-dimensional data distribution is then summarized by means of *MHIST*. As *MHIST* is known to yield poor accuracy over three dimensions, the authors imposed that each subset of attributes S_i has cardinality bounded by 3. Thus, at most 3-dimensional data distribution are summarized. In order to choose a good model (i.e., a good decomposition of the attributes), they proposed a novel algorithm which starting form the empty model (corresponding to the full independence assumption) progressively introduce the edge in the graph representing the model which mostly improve a measure of the model quality, under the constraint that the new model must continue being decomposable. In order to partition the total storage space among the k histograms, two algorithms were proposed: one optimal, which is based on dynamic programming and one sub-optimal which performs progressively splits choosing at each step the histogram which mostly decrease a measure of the overall error of the k histograms.

In [1] Aboulnaga and Chaudhuri proposed a technique, called *ST-histogram* (self-tuning histogram), for building histograms based on different approach w.r.t. respect all the previously proposed histogram-based techniques. In fact, instead of creating the histogram by examining the whole data distribution, they proposed to start with any initial, possibly rough, histogram, and then to refine it by means of query feedbacks, which can be obtained at no extra cost. Thus, the histogram is built incrementally, with little overhead at each query execution, but completely avoiding the initial cost due to data scanning. Furthermore, this technique is particularly suitable for approximating evolving data distributions, as the updates on data are incrementally propagated to the histogram, without requiring the possibly time consuming operation of building a new histogram from scratch. If some previously built histograms are available, they can be exploited to define the initial *ST-histogram*. Otherwise, the initial buckets of the *ST-histogram* can be defined, without looking at data, as the cells of a grid partitioning the data domain, each one containing an equal number of tuples, under the uniform distribution assumption over the whole data domain. When a query is performed, the actual answer value is adopted as feedback to update the histogram. Specifically, the absolute error in the estimation is computed, and each bucket involved in the query is

tuned by summing a signed error proportional to the bucket contribution in the estimate, adjusted by a dumping factor to avoid oversensitive histograms. While the bucket overall frequencies are tuned at each query, bucket boundaries are tuned only periodically, as this operation can be more expensive. The bucket boundaries restructuring entails moving some grid splits from regions with low frequency to region with higher frequency. To this aim, marginal distributions are examined: on each marginal distribution adjacent cells with similar frequency are merged, and cells with higher frequency are split. The split updates on each dimension are propagated to the multi-dimensional cells accordingly. These kind of histograms, built without an initial knowledge of the overall data distribution are unlike to provide the same accuracy as the traditional approaches. The authors, in fact, by means of experiments showed that *ST-histograms* are comparable in accuracy to those provided by *MHIST* only on data without high skews.

In [56, 57] *GENHIST* (*Generalized Histogram*), a technique for approximating multi-dimensional data distributions over real domains, was proposed by Gunopulos et al. Of course, the technique can be adopted to summarize even data distribution defined over discrete domains. The main difference w.r.t. classical histograms is that buckets yielded by *GENHIST* can overlap and they are obtained by means of a bottom-up strategy. The idea underlying GENHIST is to progressively locate regions of data which exhibit a non-homogeneous distribution w.r.t. contiguous ones. At each step, GENHIST algorithm constructs a grid based on a ξ-regular partitioning of the multi-dimensional data distribution and chooses a number of cells of the grid having the largest density. The data distribution is made smoother by randomly removing from each selected cell a number of tuples, so that the density of remaining tuples in the cell is the same as the average density of the neighbor cells. Removed tuples are stored in a bucket, whose boundaries coincide with those of the corresponding cell. The value of ξ defining the grid depends on the step of the algorithm. At the first step, an input parameter is used, and at the following steps its value is iteratively decreased, thus making the regular partitioning of data coarser: at each step the grid divides the data domain into about half as many cells as the previous step. This follows from the observation that the data distribution processed at each step of the algorithm is more homogeneous than that processed at previous steps, as high density peaks have been removed, thus larger buckets suffice to approximate data in detail. As in classical histograms, on a *GENHIST* histograms range queries are evaluated on the basis of the CVA and UFD assumptions. Indeed, as buckets are "layers" of data, in the regions where two or more buckets overlap, the approximate data density is assumed to be the sum of the average data density associated to all the overlapping buckets. Thus, the density of a region, as described by the histogram, can be obtained by summing different combinations of the stored densities. This feature is actually the advantage of the possible bucket overlapping. In fact, the number of regions described

by different densities in the histogram can result much larger than the number of buckets. *GENHIST* has been proved, by means of experiments, to be definitely superior in accuracy provided in estimating range queries w.r.t. the other histogram-based summarization techniques, especially in the high-dimensional settings. *GENHIST* has been generally considered the technique for summarizing multi-dimensional data representing the state-of-the art so far.

Another technique which builds the histogram without looking at data but by means of query result feedbacks, as *ST-histogram*, is *STHoles*, proposed in [11] by Bruno et al. Besides the query-driven incremental construction of the histogram, *STHoles* differs from classical histograms also in the possibility to construct overlapping buckets. Specifically, there is the possibility than one bucket is a hole of another one. In a *STHoles* histogram buckets are organized according to a tree structure. Each bucket b is represented by its *MBR*, but its actual region is not necessarily rectangular, as it may contain rectangular holes. Each of these rectangular holes is the bounding box of some other bucket, nested in b and considered as a child of b. The frequency value associated to each bucket takes into account only the data distribution entries contained in the actual bucket region, that is the region covered by the bucket *MBR* excluding the regions covered by its holes. An *STHoles* histogram can be initialized assuming that it coincides with a singleton bucket, corresponding to the whole data domain. When a range query is issued on the database, each bucket b in the current histogram overlapping the query range r is eligible to be updated. To this aim, by inspecting the result of the query, it is possible to compute the exact number of tuples in the intersection between r and each bucket *MBR* (of course, the query is assumed to return all the tuples in r, not only the result of an aggregate operator on these tuples). If the density of tuples in the intersection between a bucket b and the query range r is very different (larger or smaller) than the overall density of b, the intersection is transformed into a new bucket, which becomes a hole of b, and the number of tuples in the new bucket are subtracted by the summary information about the number of tuples in b. The region corresponding to the intersection might not be a rectangular region, as some holes of b could be also intersected by r. In this case, the new bucket will not coincide with the intersection, but with a shrunk region of it, which enables the new bucket region not to overlap holes of b. Adding new buckets may require to remove some others, in order to not exceed the storage space bound. To this aim, a merge of two buckets is performed. Specifically, either a pair of parent-child or a pair of siblings buckets can be merged into one only bucket. The pair which minimize a measure of loss in accuracy is chosen. By means of experiments, *STHoles* is shown to provide accuracy similar to *GENHIST* up to 4 dimensions. However, this technique has the intrinsic limit to be bounded to the query workload, and queries posed onto not-yet-explored regions of data are likely to provide unacceptable errors in estimations. On the other hand,

STHoles, and *ST-Histogram* as well, are a natural and effective solution to the approximation of evolving data sets, where the construction from scratch of a more accurate histogram on static data, as *GENHIST*, would result infeasible.

In [87] Lee et al. proposed to adopt the *Discrete Cosine Transform* (DCT) to enhance the histogram performances. Their work stems from the consideration that in order to effectively approximate a multi-dimensional data distribution, a large number of small buckets it is often required. Therefore, the propose to partition data domain by means of a fine-grain grid, thus obtaining a large number of small buckets which accurately enough represent the data distribution. Then, in order to reduce the number of buckets, the DCT is applied, and the less relevant coefficients are cut to zero. According the authors, the technique should be effective as long as the data are strongly correlated, as only in this case the DCT gives benefits in compacting the information into the low-frequency coefficients. An efficient incremental maintenance of the histogram is also proposed, which is based on the linearity of the DCT: when new data are inserted, their DCT is computed and then added to the previous one representing the histogram. Deletions of data can be viewed as insertions of negative data, and this case is treated as the insertion one.

In [83] König and Weikum proposed to improve the intra-bucket estimations by adopting a linear function curve fitting model to approximate data inside buckets instead of a flat value. Their technique, tailored for one-dimensional data distribution, first partition the data domain into buckets of possibly different size. They proposed two algorithms to this aim: one optimal and one more efficient based on a greedy strategy. After bucket boundaries are defined, the underlying data distribution is locally approximated by means of a linear function. Therefore, two values per bucket must be stored, instead of only one. They also proposed how to maintain the histogram up-to-date w.r.t. the data changes. Specifically, the query answer adopted as feedback, on the basis of a weight assigned to each possible kind of query, in order to update the coefficient representing the data distribution inside the buckets involved in the query.

In [13, 14] Buccafurri et al. proposed to improve the intra bucket estimation, in the case of one-dimensional data distributions, by associating to each bucket a bit string which instead of representing a flat value (i.e., the sum of values inside the bucket) represents an approximation of the total value distribution across the bucket domain. This is achieved by means of several proposed kinds of *indices*, which approximately describes in a hierarchical fashion the actual data distribution inside each bucket. The index minimizing a sample of queries is selected for each bucket. The idea was extended to the two-dimensional case in [15], where quad-trees were proposed to partition the data domain, starting from the overall data domain and step by step partitioning the bucket with the largest *SSE* into four new buckets. Then, the data distribution of each bucket is described by one of a set of indices which best approximates the local data distribution. Of course, on the one hand

a more detailed description of the local data distribution enables providing lower error rates in query estimation, but on the other it is more expensive to be represented, thus a smaller number of buckets can be constructed within the same storage space bound. However, the proposed representation is very compact and it is shown that adopting a smaller number of buckets is often preferable if each bucket local distribution is approximated in more detail.

In [9] Blohsfeld et al. investigated the adoption of kernel estimators in representing the intra-bucket histogram distribution, finding that it yields lower errors than classical histograms. However, their approach is focused on one-dimensional continuous data distributions, and it can not be trivially generalized to discrete multi-dimensional data distributions as it is based on the study of the derivatives of data distribution. Their approach, indeed, was generalized by Gunopulos et al. in [56, 57] to the multi-dimensional continuous distribution, finding that their novel technique, namely *GENHIST*, outperforms kernel estimators.

2.4 Sampling

The adoption of sampling to infer statistical properties of large populations, when it is not possible to take a complete census, is common in many fields. Sampling-based estimations are supported by the wide statical theory related to statistical hypotheses testing. Sampling was obviously exploited also in computer science, as also digital data can be considered a "population" settled in the world represented by their domain. The main issue in adopting sampling is the need to draw a sample which is representative of the whole population, otherwise estimations on it would result biased. For instance, if a zoologist wanted to perform a statistical study on birds, results obtained on a sample made up of only penguins would be accepted for publication in a serious journal with low probability.

One of the main characteristic which differentiate sampling techniques is the possibility to select a member either only once or more times. In the former case sampling is said *with replacement*, in the latter *without replacement*. Sampling with replacement is much simpler than sampling without replacement, as each member can be selected independently on the others.

According to the strategy adopted to draw a sample from a population, sampling techniques are classified as either as *probability* or *nonprobability*. In probability sampling each member of the population has a known probability of being included in the sample, whereas in nonprobability sampling members are chosen in some non-random manner. Both the techniques can achieve the task to draw a representative sample from the population, but only when dealing with the former one probability and statistical theory can be exploited. Probability sampling techniques are the following:

- *Random sampling*: is the purest form of sampling. Each member of the population as the same probability of being included in the sample. This

kind of sampling was used since ancient times: it had always been common to cast lots[3] for fairly choosing a group of people to be assigned some task or something (of course, it assumed that lots are unbiased!);
- *Systematic sampling*: scanning the population, every n-th member is added to the sample (e.g., if $n = 5$ the 5th member, the 10th, the 15th, and so on, are selected). This technique yield an unbiased sample only if the order adopted to scan the population is random, i.e. there are no hidden patterns in it;
- *Stratified sampling*: a stratum is a subset of the population sharing a common property (e.g., sex or age when dealing with animals). The technique first identify the relevant stratums and population is partitioned according to them. Then, for each stratum, a number of members is selected by random sampling.

As regards nonprobability sampling techniques, the most common ones are the following:

- *Convenience sampling*: the members which are more convenient to choose (according to some cost function) are included in the sample;
- *Judgement sampling*: the members that are judged (by some "expert agent") to be the most representative of the population are included in the sample;
- *Quota sampling*: like in stratified sampling, stratums are first identified and then members of each stratum are chosen according to convenience of judgement sampling;
- *Snowball sampling*: members are progressively added to the sample on the basis of those previously added. Thus, the sample grows like a rolling snowball.

Of course, nonprobability techniques are not guaranteed to yield unbiased samples. For instance, snowball sampling is intrinsically biased, and nothing assure that the most convenient member to be chosen or those judged the best are really unbiasedly representative of the population. On the other hand, nonprobability techniques are less expensive to be applied than probability ones, as selecting actual random samples could be difficult. For example, if a population is very large, some members could be difficult to be reached, but nevertheless it is necessary to consider them in order to have an unbiased sample. Therefore, it is important to develop efficient techniques for generating random sampling.

In the case of sampling with replacement from a population of N_z members, drawing a random sample of size β can be simply achieved by selecting β random integers and selecting for each selected integer i the i-th member of the population. This method could be adapted to the case of sampling without replacement by storing the already selected integers and extracting

[3] Sampling by the use lots is mentioned several times in the Bible (e.g., Nehemiah 11:1)

random integers until β distinct integers are selected. This simple algorithm can perform efficiently if $\beta \ll N_z$, as the probability of selecting more times the same integer is low. However, a deterministic upper bound of the number of extractions can not be provided, and many extractions yielding duplicate integers could be required in practice.

In order to define a systematic strategy in drawing a random sample without replacement, the sample could be drawn by scanning all the members of the population, and choosing with a probability p_i the i-th member. Being m the number of members that have been already included in the sample when the choice about the i-th is going to be taken, it can be proved that adopting

$$p_{i,m} = \frac{\beta - m}{N_z - i + 1} \ (1 \leq i \leq N_z) \tag{2.8}$$

as probability to include the i-th member in the sample, at the end of the scanning a random sample of size β is obtained [40]. That is, each of the N_z has been chosen with probability β/N_z.

Obviously, this approach has complexity $\mathcal{O}(N_z)$, as one variate must be computed for almost each member of the population. In order to decrease the complexity, it is possible to define a probability function to be adopted to skip a number of members after selecting each one [125]. If the i-th member has been selected as the m-th ($m < \beta$) in the sample, the number $S(i, m)$ of elements to be skipped for including another member in the sample can be chosen according to a probability function $f_{i,m}(s) = Pr\{S(i, m) = s\}$, with $0 \leq s \leq (N_z - i) - (\beta - m)$. The mean of this probability function, in order to obtain an unbiased sampled, must be

$$E[f_{i,m}(s)] = \frac{(N_z - i) - (\beta - m)}{n - m + 1}. \tag{2.9}$$

The function $f_{i,m}(s)$ can be defined as

$$f_{i,m}(s) = \frac{\binom{(N_z - i) - s - 1}{\beta - m - 1}}{\binom{N_z - i}{\beta - m}}. \tag{2.10}$$

In this case, only β variates must be computed, instead of N_z. However, the cost of computing each variate, in order to compute the skip length after selecting each member, is $\mathcal{O}(\frac{N_z}{\beta})$, therefore the complexity of the overall sampling procedure has not been decreased. Efficient algorithms for obtaining a random sampling with complexity $\mathcal{O}(\beta)$ were proposed by Vitter in [125] and by Ahrens and Dieter in [4]. The algorithms are based on the adoption of variates easier to compute than the exact one. Specifically, in [4] the authors proposed two methods based on the adoption of geometrically distributed random variates with mean equal to the mean of function represented by Equation 2.10.

If at the end of the process less than β members have been selected, the process is repeated. In order to avoid duplicates, the sample obtained at the first step must be stored into an array of size β. Thus, these methods have spatial complexity linear in the size of the required sample. The method proposed in [125] was shown in [126] to be slightly slower than one of the two proposed in [4], but it has the advantage to yield the sample in one pass, therefore it can be run with constant space complexity (of course, this is an advantage only if there is no need to store the sample).

The use of sampling in databases as been largely proposed since the early 1980s in the context of query optimization and exploratory data analysis. As described in Chapter 1, in these contexts fast approximate estimations of range queries can be preferable to slow exact answers. A few approaches based on the adoption of a data sample as database abstract for efficiently computing approximate query answers have been proposed [2, 49]. In fact, the majority of proposed techniques exploiting sampling for efficiently estimating query answers, rather than adopting sampling for obtaining synopses summarizing data on which performing all the estimates, adopt the sampling as an on-line operation [59, 62]. That is, a sample of data is drawn when it is needed to perform an estimate. Therefore, this sampling techniques, even though are techniques supporting efficient query estimation, can not be properly considered data summarizing techniques.

The on-line sampling techniques have the advantage consisting in the possibility do adapt somehow the sampling operation during its execution (*adaptive sampling*). For example, it is possible to continue in extracting data until a desired confidence level is guaranteed with a certain probability. This possibility, of course, enables to achieve better estimations, in terms of accuracy, than those obtained by means of pre-computed data synopses, such as histograms. On the other hand, on-line sampling requires disk accesses, then it usually results much more time-consuming than the adoption of summarized representations of data that can be kept in main memory. Observe that, if members of the data population are accessed randomly and independently one another, the number of required accesses to disk blocks could be computed by Yao's formula (Equation 1.2). According to that formula, the number of accessed pages grows almost linearly with the number of retrieved tuples. A form of convenience sampling, namely *cluster sampling*, which adopt a disk page as sample unit could be adopted to decrease the number of disk access [64]. However, if tuples were not randomly distributed among pages, the obtained sample would be biased. Several studies have been proposed in order to make on-line sampling efficient and capable of providing answers within a desired degree of accuracy [89]. The common idea on which these techniques are based is to iteratively sample tuples from data until a stopping condition is reached depending on the current sample and the desired approximation degree [58, 90, 91]. Some techniques [65] are based on a two-stage sampling (*double sampling*): first a certain number of tuples is sampled, then, on the

basis of information obtained by the first sample, a certain number of tuples is further sampled in order to obtain the required estimate accuracy.

A general algorithm for answering range queries by means of a sampling-based technique is depicted in Fig. 2.12. In this algorithm a sampling source

```
INPUT    S: a sampling source;
         N_z the number of non-null points tuples;
         r: a range defined on the same domain V;

OUTPUT   s: the estimate of the sum of frequencies as-
            sociated to values inside r;

begin
  s = 0;
  n = 0;
  while (NOT reachedStopCondition()) do begin
    p = S.extractTuple();
    if (p ∈ r) then s = s + D[p];
    n = n + 1;
  endwhile;
  return (N_z / n) · s ;
end;
```

Fig. 2.12. General algorithm for answering range queries by means of sampling

S is assumed, from which at each step a tuple $\mathbf{p}$ is obtained and its associated value $D[\mathbf{p}]$ is adopted to update the current query answer s related to the sampled tuples, until a stopping condition is reached. Finally, assuming that the sample adopted is unbiased, the answer to the query is estimated as the product between the answer s estimated on the sample and the ratio between the total number of tuples N_z and sample size n. Observe that this algorithm generalize also the case of off-line sampling. In this case, the sampling source is a list representing the pre-computed sample which can be simply scanned until its end.

2.5 Histograms vs. other techniques

Among the multitude of data-summarization proposals available in literature, histogram-based techniques undoubtedly hold the primacy in numerousness. In practical applications the use of histograms is definitely the most common in approximating data distributions. The most known commercial DBMSs adopt histograms for maintaining the statistics adopted as support for the

task of query optimization. This is due to the larger flexibility of nonparametric techniques, which the histogram-based techniques belong to, and to the negligible run-time costs in which histogram-based techniques incur in estimating query answers.

As regards the comparison between parametric and nonparametric techniques, the former ones can certainly be exploited more efficiently than the latter ones in estimating queries. In fact, they assume that actual fit a well defined mathematical model, depending on a few parameters which are constant in number independently of the actual difficulty in approximating the data distribution. That is, if a linear regression model or a gaussian distribution is adopted to model a one-dimensional data distribution, only two parameters will be adopted to describe a data distribution which can be more or less large and more or less skewed. The estimations will reduce in computing either the function approximating the data distribution in one point (for equality queries) or the integral of the function in two points (for range queries). However, general data are unlikely to be approximated with good accuracy by *a priori* chosen mathematical models. Furthermore, changing the approximation model on convenience could be infeasible, unless restricting the attention to few models, thus only partially limiting the problem. The obstinacy of some scientific researchers in trying to make general phenomena, especially human-related ones, fit mathematical models has been sometimes criticized (and also mocked) by researchers of humanistic disciplines [51]. The objective difficulty in representing general data by means of mathematical models has been perceived by database researchers since the early 1980s, when nonparametric techniques began to increase their preeminence w.r.t. the parametric ones which had been the only adopted during the 1970s to approximate data distributions. Christodoulakis, with his several works [27, 28, 29, 30, 31], strongly contributed to that evolution in research, clearly showing that modelling data distributions by means of simple statistical models yields biased estimations.

The recent adoption of mathematical transforms, in particular the discrete-wavelet-based one, seems to be promising. However, the effectiveness of the techniques has been showed to be strongly related to the thresholding strategy of the coefficients yielded by the transform. Even though some sophisticate strategies have been proposed [45, 46, 47, 48], they have only been tested on one-dimensional data.

As regards the nonparametric techniques, that is the histogram- and the sampling-based ones, is has been remarked before that the sampling-based techniques which can outperform in accuracy the histogram-based ones, are those based on on-line sampling. On-line sampling, in fact, enable to adapt the sampling to satisfy accuracy requirements. However, these are not properly summarization techniques, and they are not suitable to perform very fast estimations, which are needed especially in evaluating the cost of a large number of possible query execution plans, as they may require several disk accesses. Furthermore, in exploratory data analysis, once a random sample belonging to a region of data satisfying some conditions is retrieved, if a larger

region has to be studied, a new sample must be retrieved, as a random sample built on a set S of data is not generally also a random sample of a superset of S.

On the basis of these considerations, in the following of this thesis histogram-based techniques will be deeply investigated. Specifically, an improvement in the class of histogram-based techniques which construct the histogram according to a hierarchical binary partition of the data domain will be proposed in Chapter 3, and a new approach based on the exploitation of cluster analysis to partition data will be presented in Chapter 4.

3

Hierarchical Binary Histograms

As introduced in Section 2.3, constructing the optimal histogram (i.e., the histogram having buckets with the lowest variance) summarizing a data distribution defined over a d-dimensional domain (with $d \geq 2$) within a storage space bound has been proved to be an $\mathcal{NP}$-hard problem [104]. Therefore, the majority of the approaches proposed in literature aims at constructing "good" histograms by imposing some constraints to the partitioning scheme of the data domain. In particular, several of the best known histogram-based summarization techniques for multi-dimensional data, such as the first proposed *equi-depth* approach [103], and the more recent *MHIST* [111] and *MinSkew* [3], builds partitions of the data domain that belong to the class of the hierarchical binary partitions.

In this chapter the class of histograms based hierarchical binary partitions is deeply investigated and it will be shown how to exploit its characteristics in order to improve the histogram performances.

Specifically, the hierarchical partition scheme underlying the histogram is exploited in order to obtain a lossless compression affecting positively the accuracy and, surprisingly, even the efficiency of query answering. Then, the adoption of a further restriction on the partitioning is investigated, which enables a more compact representation of bucket boundaries. Basically, this restriction consists in constraining the split of a bucket to be laid onto a regular grid defined on the bucket itself. Several heuristics guiding the histogram construction are also proposed, and a thorough experimental analysis comparing the accuracy of histograms resulting from combining different heuristics with different representation models (both the new compression-based and the traditional ones) is provided. The best accuracy turns out from combining the grid-constrained partitioning scheme with one of the new heuristics. Histograms resulting from this combination are compared with state-of-the-art summarization techniques, showing that the proposed approach yields lower error rates and is much less sensitive to dimensionality, and that adopting the new compression scheme results in improving the efficiency of query estimation.

3.1 Hierarchical Binary Partitions

Before introducing the class of hierarchical-binary-partition-based histograms, the *hierarchical binary partition* must be introduced. This kind of partition can be obtained by recursively applying an operation called *binary split*.

Definition 3.1 (Binary Split) *Given a block $b = \langle \rho_1, \ldots, \rho_d \rangle$, let x be a coordinate on the i-th dimension of b such that $lb(\rho_i) \leq x < ub(\rho_i)$. Coordinate x divides the range ρ_i of b into $\rho_i^{low} = [lb(\rho_i)..x]$ and $\rho_i^{high} = [(x+1)..ub(\rho_i)]$, thus partitioning b into the 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 $\langle b^{low}, b^{high} \rangle$ is said to be the* binary split *of b at the* splitting position *$\langle i, x \rangle$, where i and x are said to be the* splitting dimension *and the* splitting coordinate*, respectively.*

Informally, a binary 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. The set of applied binary splits, defines a hierarchy which can be represented by a binary tree whose leaves represents a partition of the data domain.

Definition 3.2 (Hierarchical Binary Partition) *Let D be a d-dimensional data distribution with volume n^d, a* hierarchical binary partition *P of D is a binary tree such that:*

1. *the root of P is the block $\langle [1..n], \ldots, [1..n] \rangle$;*
2. *for each internal node p of P, its children represent a binary split of p itself.* □

The set of nodes, and the set of leaves of the binary partition P will be denoted, respectively, as $Nodes(P)$ and $Leaves(P)$.

An example of hierarchical binary partition on a two-dimensional data distribution is shown in Fig. 3.1.

3.2 Flat Binary Histograms

In this section a formal abstraction of classical histograms based on binary partitions, such as *MHIST* [111] and *Min-Skew* [3], is provided. This class of histograms is referred as *Flat Binary Histograms* (*FBH*), 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 3.3 (Flat Binary Histogram) *Let D be a d-dimensional data distribution and P be a hierarchical binary partition of D, the* Flat Binary Histogram *on D based on P is the set of pairs:*

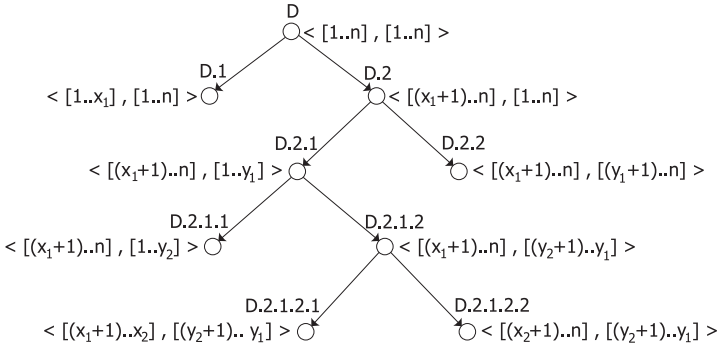


Fig. 3.1. A hierarchical binary partition

$$FBH = \{\langle b_1, sum(b_1)\rangle, \ldots, \langle b_\beta, sum(b_\beta)\rangle\},$$

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

In the following, given the flat binary histogram:

$$FBH = \{\langle b_1, sum(b_1)\rangle, \ldots, \langle b_\beta, sum(b_\beta)\rangle\},$$

the blocks $b_1, \ldots, b_\beta$ will be said to be the *buckets* of *FBH*, and the set $\{b_1, \ldots, b_\beta\}$ will be denoted as $Buckets(FBH)$.

The blocks $b_1, \ldots, b_\beta$ will be said to be *buckets* of *FBH*, and the set $\{b_1, \ldots, b_\beta\}$ will be denoted as $Buckets(FBH)$.

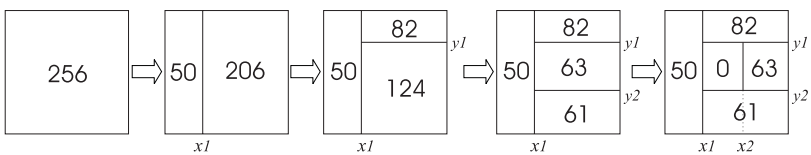


Fig. 3.2. Constructing an *FBH* based on the hierarchical binary partition of Fig. 3.1

Fig. 3.2 shows how the two-dimensional flat binary histogram corresponding to the binary partition of Fig. 3.1 can be obtained by progressively performing binary splits. The histogram consists of the following set:
$\{ < \langle [1..x_1], [1..n]\rangle\ , 50 >,$
$< \langle [x_1{+}1..n], [1..y_2]\rangle, 61 >,$
$< \langle [x_1{+}1..x_2], [y_2{+}1..y_1]\rangle, 0 >,$
$< \langle [x_2{+}1..n], [y_2{+}1..y_1]\rangle, 63 >,$
$< \langle [x_1{+}1..n], [y_1{+}1\ ..\ n]\rangle, 82 > \}.$

A flat binary histogram can be represented by storing, for each bucket of the partition, both its boundaries and the sum of its elements. In the following it will be assumed that integer values will be represented by means of 32-bit words. In order to exploit as much as possible the available storage space, usually the boundaries of the buckets are not represented explicitly (this would require $2d$ words for each bucket): the data domain is linearized and the boundaries of the bucket $b = \langle \rho_1, \ldots, \rho_d \rangle$ are translated into the one-dimensional coordinates in this linearized space of the two points $p_1 = \langle lb(\rho_1), \ldots, lb(\rho_d) \rangle$ and $p_2 = \langle ub(\rho_1), \ldots, ub(\rho_d) \rangle$, i.e. the nearest and the farthest corner of the bucket w.r.t. the point $\langle 0, \ldots, 0 \rangle$, respectively. The amount of storage space needed to represent p_1 and p_2 depends on the size of the data domain. In the following, ξ will denote the number of words needed to encode the coordinates of a multi-dimensional point adopting the above-explained strategy [1]. Therefore the amount of storage space needed to store an *FBH* with β buckets is given by

$$size(FBH) = 32 \cdot (2 \cdot \xi + 1) \cdot \beta \text{ bits.} \tag{3.1}$$

MHIST and *Min-Skew* algorithms use a different representation of buckets: instead of storing the ranges delimiting the leaves of the binary partition, they store, for each leaf, the coordinates of its *MBR* (*minimal bounding rectangle*, see Section 2.3.4). For instance, consider the case that D is a two-dimensional data distribution with two points in it, placed at the ends of a diagonal. According to this representation model, splitting D will lead to two single-point *MBR*s. W.r.t. the naive representation model introduced above for *FBH*, this aims at a higher accuracy in approximating D, and introduces no spatial overhead. In fact, representing the coordinates of the *MBR* inside a bucket b has the same cost as representing the boundaries of b, but the information provided by the *MBR* on where non null elements are located inside b is more accurate.

In Section 3.3 an alternative representation scheme is proposed, which does not enable *MBR*s to be stored, but allows bucket boundaries to be represented more efficiently, so that a larger number of buckets can be stored within the same storage space bound.

3.3 Hierarchical Binary Histograms

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 space consumption (i.e., $2 \cdot \xi \cdot |Buckets(FBH)|$ words) of an *FBH* is due to

[1] $\xi = \left\lceil \frac{\log_2 n^d}{32} \right\rceil = \left\lceil \frac{d \cdot \log_2 n}{32} \right\rceil$. Observe that for high-dimensional data a single 32-bit word is unlikely to suffice to represent a point in a d-dimensional space, as $d \cdot \log_2 n > 32$ generally holds.

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. It is rather expected that exploiting this characteristic would improve the spatial efficiency of the representation. Specifically, the boundaries of two buckets of an *FBH* corresponding to a pair of siblings in the underlying binary partition can be derived by the boundaries of their parent together with the splitting position generating them. For instance, consider two buckets b_i, b_{i+1} which correspond to a pair of siblings in the hierarchical partition underlying the histogram; b_i, b_{i+1} can be viewed as the result of splitting a block b of the multi-dimensional space along one of its dimensions and, therefore, they have 2^{d-1} coinciding vertices. However, in *FBH* histograms adopting this naive representation scheme, these coinciding vertices are stored twice, as the buckets are represented independently of each other. Therefore, the simple *FBH* representation paradigm defined in Section 3.2 does not exploit the hierarchical nature of the partition.

The idea underlying *Hierarchical Binary Histogram* consists in explicitly storing the splitting position of the binary split generating the partition, thus avoiding the explicit storing of bucket boundaries. Storing the structure of the partition enables the boundaries of the buckets (which correspond to the leaves of the partition tree) to be retrieved from the partition itself. Moreover, as storing the partition tree is less costly (in terms of amount of storage space) than storing bucket boundaries (as it will be explained in the following), some storage space can be saved and invested to obtain finer grain buckets.

Definition 3.4 (Hierarchical Binary Histogram) *Given a d-dimensional data distribution D and a hierarchical binary partition P of D, the* Hierarchical Binary Histogram *on D based on P is the pair $HBH = \langle P, S \rangle$, where S is the set of pairs $\{\langle p, sum(p) \rangle \mid p \in Nodes(P)\}$.* □

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 $Leaves(P)$.

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 one bit per node is needed 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, the splitting position which defines the ranges corresponding to its children. Therefore, each non leaf node can be stored using a string of bits, having length $32 + \lceil \log_2 d \rceil + 1$, where 32 bits are used to represent the splitting coordinate, $\lceil \log_2 d \rceil$ 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

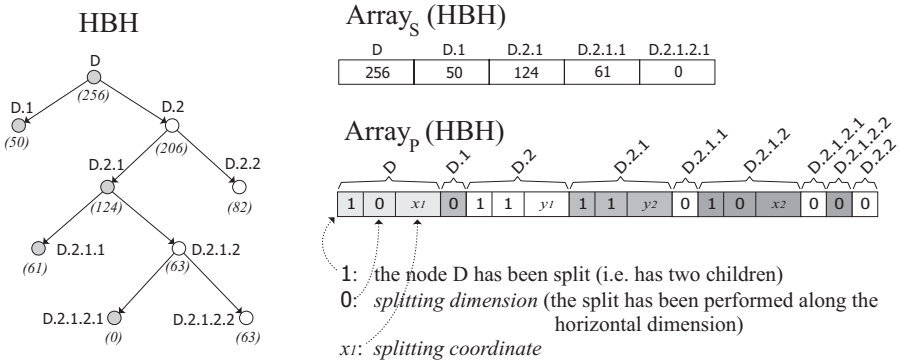


Fig. 3.3. Representation of an *HBH*

bits (denoted as $Array_P(HBH)$) consisting of the concatenation of the strings of bits representing P nodes.

The pairs $\langle p_1, sum(p_1)\rangle, \ldots, \langle p_m, sum(p_m)\rangle$ of S (where m is the cardinality of $Nodes(HBH)$) 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 $Array_P(HBH)$. 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. The sums associated to non-derivable nodes are stored into the array $Array_S(HBH)$.

The application of this representation paradigm to the *HBH* shown on the left-hand side of Fig. 3.3[2] is shown on the right-hand side of the same figure, where non-derivable nodes are colored in grey, whereas derivable nodes are white. Leaf nodes of *HBH* are represented in the array $Array_P(HBH)$ by means of a unique bit, with value 0. As regards non-leaf nodes, the first bit of their representation is 1 (meaning that these nodes are split); the second bit is 0 if the node is split along the horizontal dimension, 1 otherwise.

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, it is expected that null blocks are very likely to occur when partitioning the multi-dimensional space. This leads us to adopt an ad-hoc compact representation

[2] Observe that this is the representation of the same partition of a two-dimensional data domain depicted in Fig. 3.2

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

- if the node is not a leaf, it is represented using a $32 + \lceil \log_2 d \rceil + 1$ bits, where 32 bits are used to represent the splitting coordinate, $\lceil \log_2 d \rceil$ 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 $Array_S(HBH)$ represents the sums of all non-null non-derivable nodes.

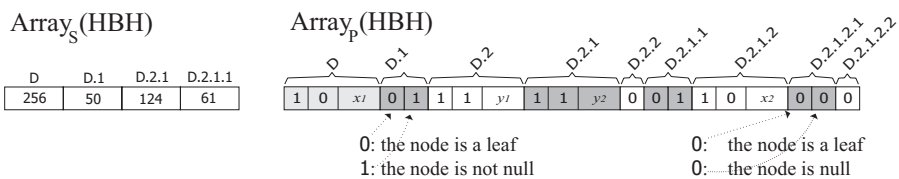


Fig. 3.4. Efficient representation of an *HBH*

A possible representation of the *HBH* shown on the left-hand side of Fig. 3.3 according to this new model is provided in Fig. 3.4. In particular, both non-leaf nodes and derivable leaf nodes are stored in the same way as in Fig. 3.3, 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.

The size of the storage space needed to represent an *HBH* with β buckets according to this model is the sum of four contributions:

1. the number of bits needed to represent the structure of the partition tree, that is $2 \cdot \beta - 1$;
2. the number of bits representing the splitting position for each non-leaf node, that is $(\beta - 1) \cdot (\lceil log\ d \rceil + 32)$;
3. the number of bits associated to all non-derivable leaf nodes to indicate if they are null or not. This number ranges from 1 to $\beta - 1$, which correspond to the cases that all but one leaf nodes are derivable, and all but one leaf nodes are non-derivable;
4. the number of 32-bit words needed to represent $Array_S(HBH)$, that is one word for each non-null non-derivable node. This number ranges from 1 to β.

According to the physical representation model presented above, it can be easily shown that the maximum size of an *HBH* with β buckets is

$$size^{max}(HBH) = \beta \cdot (67 + \lceil log\ d \rceil) - (34 + \lceil log\ d \rceil) \text{ bits}, \tag{3.2}$$

which corresponds to the case that all but one leaf nodes are non-derivable, and all non-derivable nodes are not null. Observe that 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. These can be easily proved considering that the storage space needed to represent *FBH* with β buckets is $(64 \cdot \xi + 32) \cdot \beta$, and $64 \cdot \xi + 32 > 67 + \lceil log\ d \rceil$ for any $\xi \geq 1$ and for any practically significant value of d.

Moreover, the information stored in an *HBH* is "better" organized than the corresponding *FBH*, as *HBH* represents the sums associated with all the nodes (not only the leaves) of the partition tree, which can be exploited to evaluate queries more efficiently, as it will be clearer later.

Remark. Observe that the physical representation model introduced above cannot be used to represent the coordinates of the *MBR*s inside buckets. This is due to the fact that *MBR*s of two sibling nodes of a binary partition in general do not coincide with node boundaries, because the two partitions can be shrunk to eliminate any null spaces around.

3.4 Grid Hierarchical Binary Histograms

In the previous section it has been shown how the binary partition scheme underlying a hierarchical binary histogram can be exploited to reduce the amount of memory needed to represent bucket boundaries. In this section, further constraints are introduced 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, approximating data in more detail.

By analyzing the storage space needed to represent a binary split, it turns out that the splitting coordinate representation, which employs 32 bits, it is likely to overwhelm the $\lceil \log d \rceil$ bits needed to represent the splitting dimension. This is due to the fact than the number of dimensions of data distribution can be several orders of magnitude less than the size of each dimension. If the splitting coordinate is constrained to assume one of a set of fixed coordinates, the number of bits needed to represent it can be dramatically reduced. On the basis of this considerations, it is possible to define a new class of histograms. A *Grid Hierarchical Binary Histogram* (*GHBH*) is a hierarchical binary histogram whose internal nodes cannot be split at any position: every split of a block is constrained to be laid onto a grid, which divides the block into a number of equal-size sub-blocks. This number is a parameter of the partition, and it is the same for every block of the partition tree.

Definition 3.5 (k-degree Binary Split) *Given a block* $b =< \rho_1, \ldots, \rho_d >$, *a* binary split of degree k *is a binary split at the position* $\langle i, x \rangle$ *with* $1 \leq i \leq d$ *and* $x = lb(\rho_i) + \left\lceil j \cdot \frac{size(\rho_i)}{k} \right\rceil - 1$ *for some* $j \in [1 \,..\, k-1]$.

Informally, a binary split of degree k is constrained to assume one of the $k-1$ coordinate that partition the splitting dimension into k portions of equal size. On the basis of this constrained split, it is possible to define the constrained hierarchical binary partition, and a new class of hierarchical binary histograms.

Definition 3.6 (k-degree Grid Hierarchical Binary Partition) *Given a* d*-dimensional data distribution* D, *a* grid hierarchical binary partition of degree k *of* D *is a hierarchical binary partition* P *of* D *such that, for each non-leaf node* p *of* P, *the pair of children of* p *is a binary split of degree* k *of* p. □

Definition 3.7 (k-degree Grid Hierarchical Binary Histogram) *Given a* d*-dimensional data distribution* D, *a* Grid Hierarchical Binary Histogram of degree k *on* D *is a hierarchical binary histogram* k*-GHBH* $= \langle P, S \rangle$ *on* D *where* P *is a grid hierarchical binary partition of degree* k *of* D. □

In the following, *GHBH* will be adopted as an acronym of grid hierarchical binary histogram without specifying the degree k of the partition when k is not relevant. Fig. 3.5 shows an example of a two-dimensional 4-*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 coordinate. In fact, for a grid binary partition of degree k, the splitting coordinate can be derived from the order of the "notch" which has been chosen among the $k-1$ possible notches on the splitting dimension on which the splitting coordinate is constrained to be laid. Therefore, the splitting coordinate can be stored using $\lceil \log_2(k-1) \rceil$ bits, instead of 32 bits. In the following, degree values which are a power of 2 will be considered, so that

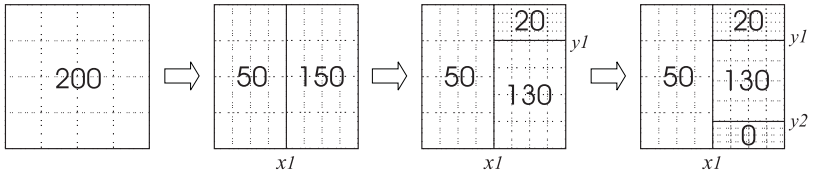


Fig. 3.5. Constructing a 4-*GHBH*

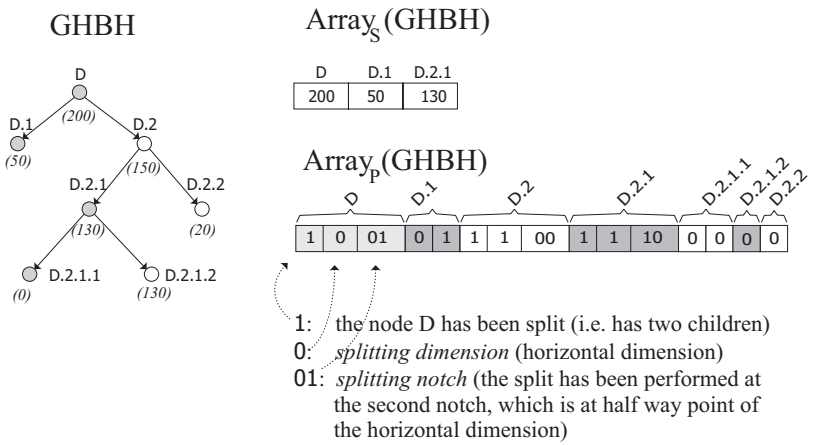


Fig. 3.6. Representing the 4-*GHBH* of Fig. 3.5

the space consumption needed to store the splitting coordinate will be simply denoted as $\log_2 k$. Fig. 3.6 shows the representation of the grid hierarchical binary histogram of Fig. 3.5.

The maximum storage space consumption of a grid hierarchical binary histogram with β buckets can be obtained from the analogous formula for hierarchical binary histograms by reducing the space needed to store the splitting coordinates from $(\beta - 1) \cdot 32$ to $(\beta - 1) \cdot log\ k$, thus obtaining

$$size^{max}(GHBH) = \beta \cdot (35 + \lceil log\ d \rceil + log\ k) - (2 + \lceil log\ d \rceil + log\ k) \text{ bits.} \quad (3.3)$$

3.5 Spatial efficiency of representation models

In this section the results of Equations 3.1, 3.2, and 3.3, stating the size of an *FBH*, and the maximum size of an *HBH* and a *GHBH* w.r.t. the number of buckets, will be extended and formally proved. Specifically, the efficiency of the different physical representation models is compared by evaluating the number of buckets of a histogram H of type *FBH*, or *HBH* or *GHBH* saturating a storage space bound B. In the following, given a storage space bound B expressed in bits, a histogram H will be said to be B-*maximal* if $size(H) \leq B$

and no split can be performed on any bucket of H, otherwise the storage space consumption of H would exceed B.

Type of Histogram	Minimum and maximum number of buckets
FBH	$\beta^{\min}_{FBH} = \beta^{\max}_{FBH} = \left\lfloor \frac{B}{32\cdot(2\cdot\xi+1)} \right\rfloor$
HBH	$\beta^{\min}_{HBH} = \left\lfloor \frac{B+\lceil \log d \rceil+34}{67+\lceil \log d \rceil} \right\rfloor$, $\beta^{\max}_{HBH} = \left\lfloor \frac{B+\lceil \log d \rceil+2}{35+\lceil \log d \rceil} \right\rfloor$
k-*GHBH*	$\beta^{\min}_{GHBH} = \left\lfloor \frac{B+\log k+\lceil \log d \rceil+2}{35+\log k+\lceil \log d \rceil} \right\rfloor$, $\beta^{\max}_{GHBH} = \left\lfloor \frac{B+\log k+\lceil \log d \rceil-30}{3+\log k+\lceil \log d \rceil} \right\rfloor$

Table 3.1. Minimum and maximum number of buckets for *B-maximal* histograms of various types

Proposition 3.1 *Let D be a d-dimensional data distribution, B a storage space bound, and T a type of histogram (where T is either FBH, HBH or k-GHBH). The number of buckets β_T of a B-maximal histogram H of type T on D is in the range $[\beta_T^{min}..\beta_T^{max}]$ where β_T^{min} and β_T^{max} are reported in Table 3.1, where ξ is the number of 32-bit words needed to represent a point of the linearized data domain ($\xi = d$ if no linearization is adopted).*

Proof.
1. T=*FBH*.
The size of an *FBH* with β buckets is $size(FBH)=(2\cdot\xi+1)\cdot 32\cdot\beta$ bits, so that $size(FBH) \leq B$ holds for all values of $\beta \leq \left\lfloor \frac{B}{32\cdot(2\cdot\xi+1)} \right\rfloor$. Therefore the latter bound on β is the number of buckets of a B-maximal *FBH*.
2. T=*HBH* or T=k-*GHBH*.
An *HBH*, as well as a k-*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). Let $size_T^{min}(\beta)$ and $size_T^{max}(\beta)$ denote, respectively, the minimum and the maximum space consumption of any histogram of type T having β buckets. The upper bound on the number of buckets of a B-maximal histogram of type T is obtained as the largest value of β which satisfies the inequality $size_T^{min}(\beta) \leq B$. Similarly, the lower bound on the number of buckets of a B-maximal histogram of type T is obtained as the largest value of β which satisfies the inequality $size_T^{max}(\beta) \leq B$.
According to the physical representation of an *HBH* described in Section 3.3, the size of an *HBH* H with β buckets can be expressed as the sum of four contributions:
$size_{HBH}(H) = (2\cdot\beta-1)+(\beta-1)\cdot(\lceil log\ d \rceil+32)+ndl(H)+32\cdot ndn^{+}(H)$, where $ndl(H)$ and $ndn^{+}(H)$ stand for the number of non-derivable leaves of H and,

respectively, the number of non-null non-derivable nodes of H. Analogously, $ndl^+(H)$ and $ndl^0(H)$ will denote the number of non-null non-derivable leaves and, respectively, the number of null derivable leaves of H. As $ndl(H) = ndl^+(H) + ndl^0(H)$ and $ndn^+(H) = \beta - ndl^0(H)$, then $size_{HBH}(H) = (2 \cdot \beta - 1) + (\beta - 1) \cdot (\lceil log\ d \rceil + 32) + 32 \cdot \beta + ndl^+(H) - 31 \cdot ndl^0(H)$. Similarly, the size of a k-*GHBH* H with β buckets is
$size_{GHBH}(H) = (2 \cdot \beta - 1) + (\beta - 1) \cdot (\lceil log\ d \rceil + \log\ k) + 32 \cdot \beta + ndl^+(H) - 31 \cdot ndl^0(H)$.

The expressions for $size_T(H)$ (for either T=*HBH* and T=*GHBH*) have minimum value when $ndl^+(H) = 0$ and $ndl^0(H) = \beta - 1$, which occurs for a histogram of type T with β buckets where all but one leaves are non-derivable and null. Likewise, the expressions for $size_T(H)$ have maximum value when $ndl^+(H) = \beta - 1$ and $ndl^0(H) = 0$, which occurs for a histogram of type T with β buckets where all but one leaves are non-derivable and not null. Thus the minimum and maximum storage consumption of an *HBH* and a *GHBH* having β buckets are, respectively:
$size^{min}_{HBH}(\beta) = \beta \cdot (35 + \lceil log\ d \rceil) - \lceil log\ d \rceil - 2$;
$size^{min}_{GHBH}(\beta) = \beta \cdot (3 + \lceil log\ d \rceil + log\ k) - \lceil log\ d \rceil - log\ k + 30$;
$size^{max}_{HBH}(\beta) = \beta \cdot (67 + \lceil log\ d \rceil) - \lceil log\ d \rceil - 34$;
$size^{max}_{GHBH}(\beta) = \beta \cdot (35 + \lceil log\ d \rceil + log\ k) - \lceil log\ d \rceil - log\ k - 2$.
As said above, β^{max}_{HBH}, β^{max}_{GHBH}, as well as β^{min}_{HBH}, β^{min}_{GHBH}, are straightforward. □

Observe that while all possible B-maximal histograms of type *FBH* have the same number of buckets (for a given B and a given size of the data domain), this does not hold for *HBH* and *GHBH*. This is due to the fact that the buckets of an *HBH* (or, equivalently, a *GHBH*) have a different storage space consumption depending on the underlying data distribution. Therefore bounds β^{min}_{HBH}, β^{min}_{GHBH}, β^{max}_{HBH}, β^{max}_{GHBH} reported in Table 3.1 have been computed by considering the case that the available storage space B is equal to the maximum and minimum storage space consumption of an *HBH* and a *GHBH* histogram, respectively.

Comparing the ranges defining the possible number of buckets of the different types of histogram, 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 the denominator of β^{min}_{HBH} (i.e. $67 + \lceil \log_2 d \rceil$) is less than the denominator of β_{FBH} (i.e. $32 \cdot (2 \cdot \xi + 1)$) even for $\xi = 1$. Analogously, the constraint on the splitting position of a *GHBH* further increases the number of buckets that can be represented within B, as it can be assumed that $\log_2 k < 32$, thus $67 > 35 + \log_2 k$.

In order to give an idea of the benefits (in terms of number of buckets) introduced by the efficient representation models of *HBH* and *GHBH*, consider the case of an 8-dimensional data distribution. In this scenario, it is likely that ξ is at least 2 (as $\xi = \left\lceil \frac{d \cdot \log_2 n}{32} \right\rceil$, a single 32-bit word suffices to encode the coordinates of the data domain only if $n \leq 16$, that is a rather specific case).

Therefore the number of buckets of an *HBH* is between 2 and 4 times the number of buckets of a *FBH* on the same data; considering the case of 8-*GHBH*, the number of buckets is between 4 and 18 times that of an *FBH* on the same data.

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*.

3.6 Constructing V-Optimal histogram

As introduced in Section 2.3, one of the most important issues when dealing with multi-dimensional histograms is building the histogram which approximates "best" the original data distribution, while being constrained to fit in a given storage space bound. The *SSE* of a histogram is a widely used metric to measure its "quality", in terms of accuracy of the estimated answers. Independently on the class of the underlying partition, the *SSE* of a histogram consisting of the buckets $\{b_1, \ldots, b_\beta\}$ is defined as $\sum_{i=1}^{\beta} SSE(b_i)$, where the *SSE* of a single bucket is given by $SSE(b_i) = \sum_{\mathbf{j} \in b_i} (D[\mathbf{j}] - avg(b_i))^2$. The histogram having minimum *SSE* w.r.t. all other histograms of the same size is said *V-Optimal*.
Given a space bound B, the histogram on D which has minimum *SSE* among all histograms on D whose size is bounded by B is said to be *V-Optimal* (w.r.t. B). This notion of optimality can be trivially specialized to the case of histograms based on binary partitions.

Definition 3.8 (V-Optimal binary-partition-based histogram) *Let D be a d-dimensional data distribution, B a storage space bound, and T a type of histogram (where T is either FBH, HBH, or k-GHBH). A histogram $H^\star$ of type T on D is said to be* V-Optimal *w.r.t. B if the following conditions hold:*

1. $size(H^\star) \leq B$;
2. $SSE(H^\star) = min_{H' \in \mathcal{H}_T(D,B)} \{SSE(H')\}$

where $\mathcal{H}_T(D, B)$ is the set of all histograms of type T on D whose size is less than or equal to B. □

The following theorem states the computational complexity of constructing the V-Optimal *FBH*, *HBH*, and *GHBH*.

Theorem 3.1 *Let D be a d-dimensional data distribution, B a storage space bound, and T a type of histogram (where T is either FBH, HBH, or k-GHBH). A V-Optimal histogram $H^\star$ of type T on D w.r.t. B can be computed in the complexity bounds reported in Table 3.2.*

Type of Histogram	Complexity bound for the V-Optimal histogram
FBH	$\mathcal{O}(d \cdot \frac{B^2}{\xi^2 \cdot 2^d} \cdot n^{2d+1})$
HBH	$\mathcal{O}\left(d \cdot \frac{B^2}{2^d} \cdot n^{2d+1}\right)$
k-*GHBH*	$\mathcal{O}\left(d \cdot \frac{B^2}{2^d} \cdot k^{d+1} \cdot n^d\right)$

Table 3.2. V-Optimal histogram construction complexity

Proof.
1. *T=FBH*. 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 $\gamma = (2 \cdot \xi + 1) \cdot 32$, 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 $S \geq \gamma \ \wedge \ (S < 2{\cdot}\gamma \ \vee \ vol(b){=}1)$;

3. $SSE^*(b, S) = min\{SSE^*(b^{low}, S1) + SSE^*(b^{high}, S2) \mid$
   $\langle b^{low}, b^{high} \rangle$ *is a binary split on* b,
   $S1{>}0,\ S2{>}0,\ S1{+}\ S2{=}S\}$,
   otherwise.

The 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 $\mathcal{O}(d{\cdot}n{\cdot}\frac{B}{\xi})$ values computed at the previous steps, as the possible binary splits of a block are $\mathcal{O}(d{\cdot}n)$ and there are $\mathcal{O}(\frac{B}{\xi})$ possible ways to divide S into two halves which are multiple of γ.
The number of different $SSE^*(b, S)$ to be computed are $\mathcal{O}(\frac{B}{\xi} \cdot \frac{n^{2d}}{2^d})$, as the number of sub-blocks of D are $\mathcal{O}(\frac{n^{2d}}{2^d})$, and the number of possible values of S are $\mathcal{O}(\frac{B}{\xi})$.
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 $\mathcal{O}(n^{2d})$. It follows that the overall cost of the dynamic programming algorithm is $\mathcal{O}(d \cdot \frac{B^2}{\xi^2 \cdot 2^d} \cdot n^{2d+1})$.
2. *T=HBH*. The problem of finding the V-Optimal *HBH* can be formalized

and solved following the same approach as the one just described for *FBH*. 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^* as either a left-hand child or a right-hand child of some node. In fact, due to the physical representation paradigm (Section 3.5), the storage consumption of an *HBH* constructed on b is different in these two cases. Intuitively enough, this leads to a recursive formulation of the V-Optimal problem which is different from that described for *FBH*. The minimum *SSE* of any *HBH* H on b having $size(H) \leq S$ must be defined both in the case that b is considered as a left-hand child node (which will be denoted by $SSE^*_{left}(b,S)$) and a right-hand child node (which will be denoted 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*. The main differences are that in the non-recursive cases (i.e. the cases such that no *HBH* can be constructed or no split can be performed on b) more complex conditions on the storage space must be expressed, as the storage space consumption of b depends 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 $\langle b^{low}, b^{high}\rangle$ 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 $\mathcal{O}(B\cdot\frac{n^{2d}}{2^d})$ values of $SSE^*_{left}(b,S)$ and $\mathcal{O}(B\cdot\frac{n^{2d}}{2^d})$ values of $SSE^*_{right}(b,S)$, where each one is computed in time $\mathcal{O}(d\cdot n\cdot B)$.

3. T=k-*GHBH*. The problem of finding the V-Optimal k-*GHBH* 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 coordinate 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 both $SSE^*_{left}(b,S)$ and $SSE^*_{right}(b,S)$ needed to compute $SSE^*_{left}(D,B)$ exhibits a different complexity bound w.r.t. the case of *HBH* as:

1. The cost of computing a single value of $SSE^*_{left}(b,S)$ or $SSE^*_{right}(b,S)$ is reduced to $\mathcal{O}(d\cdot k\cdot B)$, since all the possible binary splits of degree k on a block are $d\cdot k$ (instead of $d\cdot n$).
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 $\mathcal{O}(n^d\cdot\frac{k^d}{2^d})$ (instead of

$\mathcal{O}(\frac{n^{2d}}{2^d})$). Thus the number of values of $SSE^*_{left}(b, S)$ or $SSE^*_{right}(b, S)$ to be computed is $\mathcal{O}(n^d \cdot \frac{k^d}{2^d})$ for each S in $[0..B]$.

3. The cost of computing the *SSE* of all the $\mathcal{O}(n^d \cdot \frac{k^d}{2^d})$ blocks is $\mathcal{O}(n^d \cdot k^d)$.

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

Results for *FBH*s in Theorem 3.1 can be viewed as an extension of the results presented in [104], 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[3]. Observe that this result does not hold for arbitrary partitions, where the problem of finding the V-Optimal histogram has been shown to be $\mathcal{NP}$-hard in the two-dimensional case [104]. In the one-dimensional case the classes of arbitrary and hierarchical partitions coincide, and thus Theorem 3.1 is consistent with that of [74], where a polynomial-time algorithm for constructing a V-Optimal histogram on a one-dimensional data distribution has been proposed.

Comparing results for *FBH*s, *HBH*s and *GHBH*s in Theorem 3.1, it can be observed 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. However, the two complexity bounds have the same polynomial degree w.r.t. the volume 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 k-*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.

3.7 Greedy algorithms for histogram construction

From the complexity bounds reported in Table 3.2 it is possible to draw the conclusion that V-Optimal hierarchical histograms can be built in time polynomial w.r.t. the size of the domain of the input data distribution. In particular, both *FBH** and *HBH** can be constructed in nearly quadratic time w.r.t. n^d, whereas k-*GHBH** in linear time (since the grid degree k can be assumed as a constant). Indeed, for high-dimensionality scenarios the size of the domain is so large that finding the V-Optimal becomes unfeasible. In order to reach the goal of minimizing the *SSE*, in favor of simplicity and speed,

[3] Indeed [104] 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.

a greedy approach for constructing the histogram is adopted, accepting the possibility of obtaining a non-optimal solution. As it will be shown later, this approach can work in linear time w.r.t. N_z (the number of non-null points inside D), which is generally much smaller than n^d (especially in the case of high-dimensionality data).
The proposed approach can be viewed as an extension of the standard greedy strategy adopted by MHIST and Min-Skew. 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 a 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, the 32 bits used to represent the sum of its elements are saved. 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 by experimental analysis are reported in Table 3.3.

Two of them (namely *Max-Red*^{marg} and *MaxDiff*) are not new, as they were used by other techniques (*Min-Skew* and *MHIST*, respectively) to drive the histogram construction. Criteria denoted as *marginal* (marg) investigate *marginal distributions* of blocks. The marginal distribution of a block b along the i-th dimension, denoted by $marg_i(b)$, has been defined in Section 1.5. In the following, the term $marginalSSE$ will be used to denote $SSE(marg_i(b))$ for some $i \in \{1..d\}$.

The resulting algorithm is shown in Fig. 3.7. It uses a priority queue q where nodes of the histogram are ordered according to their need to be partitioned. At each step, the node at the top of the queue is extracted and split, and its children are in turn enqueued. Before adding a new node b to the queue, the function $Evaluate(G, b)$ is invoked, G being the adopted greedy criterion. This function returns both a measure of the need of b to be partitioned (denoted as *need*), and the position $\langle dim, coord \rangle$ of the most effective split, according to the adopted criterion G.
For instance, if $G =$*Max-Var/Max-Red*, the function returns the *SSE* of b into *need*, and the splitting position which yields the largest reduction of *SSE* into $\langle dim, coord \rangle$. Otherwise, if *Max-Red* criterion is adopted, the value of *need* returned by *Evaluate* on b is the maximum reduction of *SSE* which can be obtained by splitting b, and the returned pair $\langle dim, coord \rangle$ defines the position corresponding to this split.

Criterion	**The node b to be split, and the splitting position** $\langle dim, coord \rangle$
Max-Var/ Max-Red	the leaf node b having maximum *SSE* is chosen, and split at the position $< dim, coord >$ 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 leaf node, the marginal *SSE* along its dimensions are evaluated, and the node 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 coordinate $coord$ 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 leaf node is reduced by trying all possible splits. b and $< dim, coord >$ is splitting 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 the buckets of the histogram)
Max-Red^{marg} (used by *Min-Skew*)	all possible splits along every dimension of all leaf nodes are performed, and the corresponding reductions of marginal *SSE* (along the splitting dimensions) are evaluated. b and $< dim, coord >$ are the bucket and the position such that the reduction of $SSE(marg_{dim}(b))$ obtained by splitting b at $< dim, coord >$ 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
MaxDiff (used by *MHIST*)	the leaf node b is chosen whose marginal distribution (along any dimension i) 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 leaf node. Then b is split along the dimension i by putting a boundary between e_j and e_{j+1}.

Table 3.3. Splitting strategies

The function *BinarySplit* takes the following arguments: a bucket b of the histogram, the chosen splitting position and the available storage space B. It returns the pairs of sub-blocks $\langle b^{low}, b^{high} \rangle$ obtained by performing the specified binary split of b. Moreover, it evaluates the storage space consumption of adding b^{low} and b^{high} as children of b (the storage space needed to store these new buckets depending on the histogram type). If the sum of this storage space consumption with the current size of H is smaller than or equal to the space bound B, then buckets b^{low}, b^{high} are actually inserted into H; otherwise, H is not updated and the next invocation of *overflow*( ) will return *true*: this ends the histogram construction.

INPUT D: a multi-dimensional data distribution;
B: available amount of storage space for representing the histogram;
T: the type of histogram to be built ($T \in \{FBH, HBH, GHBH\}$);
G: the greedy criterion to be adopted;

OUTPUT H: a histogram of type T on D within B;

begin
$q :=$ **new** Queue();
$b_0 := \langle [1..n], \ldots, [1..n] \rangle$;
$H :=$ **new** Histogram(T, b_0);
$\langle need, dim, coord \rangle$ = Evaluate(G, b_0);
$q.Insert(< b_0, \langle need, dim, coord \rangle >)$;
while (!H.overflow( )) **do begin**
$< b, \langle need, dim, coord \rangle > = q.GetFirst(\ )$;
$\langle b^{low}, b^{high} \rangle = H$.BinarySplit$(b, dim, coord, B)$;
$q.Insert(\langle b^{low}$, Evaluate$(G, b^{low}) \rangle)$;
$q.Insert(\langle b^{high}$, Evaluate$(G, b^{high}) \rangle)$;
endwhile;
return H;
end

Fig. 3.7. Greedy algorithm

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

The complexity of Greedy Algorithm strictly depends not only on the type of histogram to be built, but also on the adopted data model. For instance, a d-dimensional data distribution can be stored into a d-dimensional array (where each cell is associated to a point of the d-dimensional space), or by adopting a sparse model, where only non null data are stored as d-tuple. In the latter case, D will be a set of N_z tuples $\langle x_1, \ldots, x_d, val \rangle$, where $x_1, \ldots, x_d$ are the coordinates and val the value of non-null points. In the following, complexity bounds and workspace analysis are provided for the greedy algorithms constructing different types of histograms (*FBH*, *HBH*, and *k-GHBH*) adopting different data models (sparse and non-sparse). Moreover, the use of pre-computed auxiliary data structures is investigated as a support for the evaluation of greedy criteria.

3.7.1 Greedy criteria evaluation for *FBH* and *HBH* construction

The complexity of computing function *Evaluate* on a block b is now evaluated for the case of both the *FBH* and *HBH* construction, when different greedy criteria G are used, and when either the sparse data model or the non-sparse one are adopted. It will be shown that the order of magnitude of the computational complexity of $Evaluate(G, b)$ does not depend on the criterion G. This is due to the fact that the *SSE* of a block, as well as its reduction due to a split, can be computed by scanning marginal distributions, as explained in the following.

Max-Red

Let $b = \langle \rho_1, \ldots, \rho_d \rangle$ be the block of D on which function *Evaluate* is invoked. Denoting the binary split of b at the position $\langle i, j \rangle$ by $\langle b^l, b^h \rangle$, it can be shown that the reduction of $SSE(b)$ due to this split is given by:

$$Red(b,i,j) = SSE(b) - \big(SSE(b^l) + SSE(b^h)\big) = \\ = \frac{vol(b^l) \cdot vol(b^h)}{vol(b)} \cdot \left(\frac{sum(b^l)}{vol(b^l)} - \frac{sum(b^h)}{vol(b^h)}\right)^2 \tag{3.4}$$

As $sum(b^l) = \sum_{k=0}^{j-lb(\rho_i)} marg_i(b)[k]$, and $sum(b^h) = sum(b) - sum(b^l)$, then $Red(b, i, j)$ can be computed by accessing $marg_i(b)$. In particular, notice that all possible splits along the dimension *dim* can be evaluated progressively, starting from $j^{min} = lb(\rho_i(b))$ to $j^{max} = ub(\rho_i(b)) - 1$. That is, denoting as $b^l(k)$, $b^h(k)$ the sub-blocks of b obtained by performing the binary split at the k-th position, comparing all possible splits along the dimension i can be accomplished by first computing $marg_i(b)$, and then scanning it once, as $sum\left(b^l(k)\right) = sum\left(b^l(k-1)\right) + marg_i^b[k]$.

The cost of constructing all marginal distributions is either $\mathcal{O}(d \cdot n^d)$ (non-sparse data model) or $\mathcal{O}(d \cdot N)$ (sparse data model). The cost of scanning all marginal distributions to find the most effective splitting position is $\mathcal{O}(d \cdot n)$, so that the complexity of $Evaluate(Max\text{-}Red, b)$ is bounded by either $\mathcal{O}(d \cdot n^d)$ (non-sparse data model) or $\mathcal{O}(d \cdot N + d \cdot n)$ (sparse data model).

Max-Red$^{\mathbf{marg}}$

The reduction of $SSE(marg_i^b)$ due to a split of b at the position $\langle i, j \rangle$ can be shown to be (by applying the definition of *SSE*):

$$Red^{marg}(b,i,j) = Red(b,i,j) \cdot P_i, \tag{3.5}$$

where P_i is the ratio between the volume of b and the size of its i-th dimension. Therefore the computation of $Red^{marg}(b, i, j)$ can be accomplished

within the same bound as $Red(b, i, j)$, as well as the cost of computing the splitting position yielding the largest reduction of marginal *SSE* has the same complexity bound as computing the splitting position corresponding to the largest reduction of *SSE*. This implies that $Evaluate(Max\text{-}Red^{marg}, b)$ can be computed within the same bound as the case that *Max-Red* is adopted.

Max-Var / Max-Red

The value of $SSE(b)$ (which is returned as *need*) is given by:

$$SSE(b) = sumSquare(b) - \frac{(sum(b))^2}{vol(b)} \tag{3.6}$$

where $sumSquare(b)$ is the sum of the squares of all values contained in b. This implies that $SSE(b)$ can be computed by accessing all non null elements inside b. Therefore the cost of evaluating $SSE(b)$ is $\mathcal{O}(n^d)$ (non-sparse data model), or $\mathcal{O}(N)$ (sparse data model).
The most effective splitting position can be evaluated in the same way as the case that *Max-Red* is adopted, and this cost dominates the cost of computing $SSE(b)$.
Therefore $Evaluate(Max\text{-}Var\ /\ Max\text{-}Red, b)$ can be computed within the same bound as the case of *Max-Red*.

Observe that, when *Max-Var/Max-Red* is adopted, the strategy used by Greedy Algorithm can be modified to make the computation of the histogram more efficient. Instead of computing the most effective splitting position when a bucket is inserted into the queue q, the value of $\langle dim, coord \rangle$ is evaluated only when the bucket is extracted from q. In fact, when a new bucket is generated and inserted into q, its position inside the queue depends only on its *SSE*; similarly, the bucket which is most in need of partitioning is chosen only on the basis of its *SSE*. Therefore, computing the most effective splitting position $\langle dim, coord \rangle$ for a bucket b is useful only in the case that b is extracted from the queue. By using this strategy, the most effective splitting position is evaluated half many times as the previous strategy, as the buckets of the returned histogram (which correspond to the leafs of the underlying partition tree) have never been chosen to be split during the algorithm execution.

Max-Var$^{\mathbf{marg}}$ /Max-Red$^{\mathbf{marg}}$

In this case, the marginal distributions of b must be constructed to compute both the value of *need* (that is, the maximum variance of the marginal distributions) and the most effective splitting position. The value of *need* can be computed by scanning all marginal distributions, and the reductions of marginal variance can be evaluated in the same way as the case that *Max-Red*$^{\text{marg}}$ is adopted.
Therefore, computing $Evaluate(Max\text{-}Var^{marg}/Max\text{-}Red^{marg}, b)$ has the same complexity bound as previous cases.

MaxDiff

After constructing the marginal distributions, they are scanned in order to find the largest difference between two contiguous values of the same marginal distribution. Therefore, even this criterion has the same complexity bound of the previous cases.

3.7.2 Greedy criteria evaluation for *GHBH* construction

The main difference w.r.t. the construction of an *HBH* is that splitting positions in a *GHBH* are constrained by the grid, so that the number of possible splits to be compared when processing the block b extracted from q is $d \cdot k$ (instead of $d \cdot n$, as in the *HBH* case). The computation of $Red(b, i, j)$ and $Red^{marg}(b, i, j)$ corresponding to all the $d \cdot k$ splitting positions can be efficiently accomplished after pre-computing d temporary data structures. Differently from the case of *HBH*, these temporary data structures are not the marginal distributions of b, but consist in the marginal distributions of $grid(b)$, which is constructed as follows. $grid(b)$ is a bucket containing k^d elements, where each cell contains the sum of the elements of b located in the corresponding cell of the k-th degree grid overlying b. The marginal distributions of $grid(b)$ will be denoted as $k\text{-}marg_1, \ldots, k\text{-}marg_d$. Fig. 3.8 shows the k-marginal distributions associated to a bucket b w.r.t. a 4th degree grid.

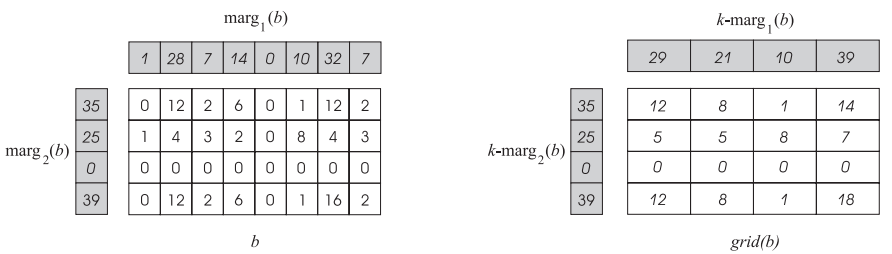


Fig. 3.8. k-marginal distributions of a bucket

Let $\langle i, j \rangle$ be an admissible splitting position for the bucket b, and $x, x{+}1$ be the corresponding cells of $k\text{-}marg_i$ (i.e. the contiguous cells of $k\text{-}marg_i$ which would be separated by performing the split). Then, the reduction of $SSE(b)$ due to this split can be computed using the values $vol(b^l)$, $vol(b^h)$, $sum(b^l)$ and $sum(b^h)$, as explained for the construction of an *HBH* (see formulas (3.4) and (3.5)). Specifically,
$sum(b^l) = \sum_{0 \leq k \leq x} k\text{-}marg_i[k]$ and $sum(b^h) = sum(b) - sum(b^l)$.

Obviously, constructing the k-marginal distributions has either cost $\mathcal{O}(d \cdot N)$ (sparse data model) or $\mathcal{O}(d \cdot n^d)$ (non-sparse model), but their scanning has cost $\mathcal{O}(d \cdot k)$ (instead of $\mathcal{O}(d \cdot n)$, as in the *HBH* case). Therefore, by applying the same reasoning explained in the previous section, it is easy to show that

$Evaluate(G, b)$ has cost $\mathcal{O}(d \cdot N + d \cdot \alpha)$ and $\mathcal{O}(d \cdot n^d)$ for the two data models, respectively, where $\alpha = k$ for all greedy criteria G except *Max-Var*^{marg}/*Max-Red*^{marg}. When *Max-Var*^{marg}/*Max-Red*^{marg} is adopted, $\alpha = n$: in fact, in order to apply this criterion, it is necessary not only to access the d k-marginal distributions to establish the most effective split, but it is also necessary to access the d marginal distributions (which have size n) in order to compute the maximum marginal *SSE*, that corresponds to the value *need* of the bucket.

3.7.3 Using pre-computation for evaluating greedy criteria

Each invocation of $Evaluate(G, b)$ can be accomplished more efficiently if the array F of *partial sums* and the array F^2 of *partial square sums* are available. F and F^2 have volume $(n+1)^d$, and are defined on the multi-dimensional range $\langle [0..n], \ldots, [0..n] \rangle$ as follows:

- $F[i_1, \ldots, i_d] = \begin{cases} 0, \text{if } i_j = 0 \text{ for some } j \in [1..d] \ ; \\ \\ sum(\langle 1..i_1, \ldots, 1..i_d \rangle), \text{otherwise.} \end{cases}$
- each element $F^2[i_1, \ldots, i_d]$ is either 0 (if $i_j = 0$ for some $j \in [1..d]$) or the sum of all the values $(D[j_1, \ldots, j_d])^2$ where $1 \leq j_k \leq i_k$ for each $k \in [1..d]$, otherwise.

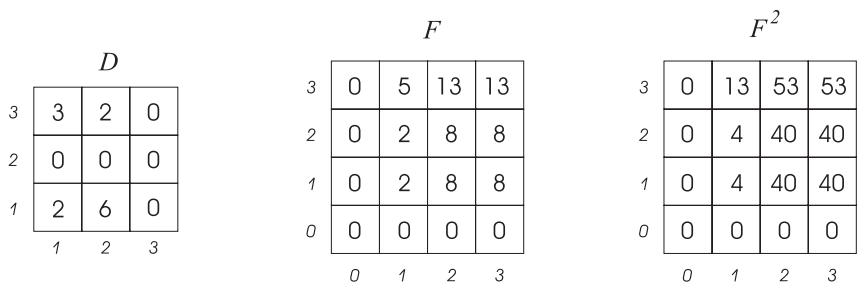


Fig. 3.9. Arrays of partial sums and partial square sums

Fig. 3.9 shows an example of arrays of partial sums and partial square sums.
By using F and F^2 both the *SSE* of a block b and the reduction the *SSE* due to a split of b can be computed efficiently, as both $sum(b)$ and $sumSquare(b)$ can be evaluated by accessing 2^d elements of F and F^2, instead of accessing all the elements of b. For instance, in the two-dimensional case depicted in Fig. 3.9, $sum(\langle [2..3], [2..3] \rangle) = (-1)^0 \cdot F[3,3] + (-1)^1 \cdot F[1,3] + (-1)^1 \cdot F[3,1] + (-1)^2 \cdot F[1,1] = 13 - 5 - 8 + 2 = 2$. In general, given a block $b = \langle [l_1..u_1], \ldots, [l_d..u_d] \rangle$, the values of $sum(b)$ and $sum^2(b)$ can be evaluated as follows:

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

and

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

In these expressions:

- $\overline{b} = \langle [l_1 - 1..u_1], \ldots, [l_d - 1..u_d] \rangle$;
- $vrt(\overline{b})$ is the set of vertices [4] of $\overline{b}$;
- $\mathbf{uv}(b) = \langle u_1, \ldots, u_d \rangle$ is the "upper" vertex of b;
- $\mathcal{C}(\imath, \mathbf{j}) = \sum_{k=1}^{d} f(i_k, j_k)$, where: $f(a,b) = \begin{cases} 1, \text{if } a \neq b; \\ 0, \text{if } a = b. \end{cases}$

Then, for any splitting position $\langle i, j \rangle$, once $sum(b^l)$, $sum(b^h)$, and $sumSquare(b)$ have been computed, either $Red(b, i, j)$, $Red^{marg}(b, i, j)$ and $SSE(b)$ can be evaluated using formulas (3.4), (3.5), (3.6).

3.7.4 Complexity of Greedy Algorithm

In this section the time complexity of Greedy Algorithm is analyzed for all the possible combinations histogram type, adopted greedy criterion and adoption of either the sparse or non-sparse model or pre-computation.

	Sparse model	***Non-Sparse model***	***Using pre-computation***	
			Cost of pre-computation	*Cost of computation*
FBH	$O\left((d \cdot N + d \cdot n) \cdot \beta_{FBH}^{max}\right)$	$O\left(n^d \cdot \beta_{FBH}^{max}\right)$	$O\left(2^d n^d\right)$	$O\left(2^d \cdot d \cdot n \cdot \beta_{FBH}^{max}\right)$
HBH	$O\left((d \cdot N + d \cdot n) \cdot \beta_{HBH}^{max}\right)$	$O\left(n^d \cdot \beta_{HBH}^{max}\right)$	$O\left(2^d n^d\right)$	$O\left(2^d \cdot d \cdot n \cdot \beta_{HBH}^{max}\right)$
k-GHBH	$O\left((d \cdot N + d \cdot \alpha) \cdot \beta_{GHBH}^{max}\right)$	$O\left(n^d \cdot \beta_{GHBH}^{max}\right)$	$O\left(2^d n^d\right)$	$O\left(\left(2^d \cdot d \cdot \alpha + \log \beta_{GHBH}^{max}\right) \cdot \beta_{GHBH}^{max}\right)$

Table 3.4. Complexity bounds of Greedy Algorithm

Theorem 3.2 *Given a d-dimensional data distribution D with volume n^d containing exactly N non-null points, the time complexity of the greedy algorithms computing a histogram of type T (where T is either FBH, HBH or k-GHBH) on D, adopting either the non-sparse data model or pre-computation, are reported in Table 3.4, where $\alpha = n$ if* Max-Var^{marg}/Max-Red^{marg} *criterion is adopted, and $\alpha = k$ for all the other greedy criteria.*

[4] Formally a point $\mathbf{x} = \langle x_1, \ldots, x_d \rangle$ belonging to the block $b = \langle \rho_1, \ldots, \rho_d \rangle$ is said to be a *vertex* of b if for each $i \in [1..d]$ x_i is either $lb(\rho_i)$ or $ub(\rho_i)$.

Proof. Complexity bounds when pre-computation is not used were obtained by multiplying the maximum number of iterations of Greedy Algorithm (which are $\mathcal{O}(\beta_T^{max})$) for the cost of each iteration. The cost of each iteration of Greedy Algorithm is dominated by the cost of evaluating the greedy criterion G on a bucket b, that is by the cost of computing $Evaluate(G, b)$ (which has been computed in Section 3.7.1 for *FBH* and *HBH*, and in Section 3.7.2 for *GHBH*). In the case that pre-computation of F and F^2 is performed, the cost of Greedy Algorithm is given by the sum of three contributions:

1. $PreComp$: the cost of pre-computing F and F^2;
2. C^U: the cost of all the updates to the priority queue;
3. C^E: the cost of computing the function $Evaluate$ for all the nodes to be inserted in the queue.

These contributions can be computed as follows:

1. Both F and F^2 can be constructed "incrementally", by accessing only once each cell of the multi-dimensional array corresponding to D and accessing $2^d - 1$ cells of F and F^2 computed at the previous steps. For instance, in the two dimensional case:
   $F[\langle i,j\rangle] = D[\langle i,j\rangle] + F[\langle i-1,j\rangle] + F[\langle i,j-1\rangle] - F[\langle i-1,j-1\rangle]$, and $F^2[\langle i,j\rangle] = (D[\langle i,j\rangle])^2 + F^2[\langle i-1,j\rangle] + F^2[\langle i,j-1\rangle] - F^2[\langle i-1,j-1\rangle]$.
   These formulas con be easily generalized to the multi-dimensional case, so that the cost of computing F and F^2 is given by: $PreComp = \mathcal{O}(2^d \cdot n^d)$.
2. As to term C^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 Greedy Algorithm is equal to the number of buckets it produces. Thus, denoting as β the number of buckets of the histogram produced by the greedy algorithm, the overall cost C^U of the updates to the priority queue is $\mathcal{O}(\beta \cdot log(\beta))$.
3. Let $C(Evaluate(G, b))$ denote the cost of computing function $Evaluate$ on the block b w.r.t. the greedy criterion G; moreover the binary histogram produced by Greedy Algorithm is denoted by H. Thus the term C^E is given by $\sum_{b \in Nodes(H)} C(Evaluate(G, b))$.

As shown in Section 3.7.3, the *SSE* of a block and the reduction of *SSE* due to a split 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 (see formulas (3.6) and (3.4)). Clearly, also the reduction of $SSE(marg_{dim}(b))$ due to the split of b along any point on dim can be computed in $\mathcal{O}(2^d)$, as it can be derived from the reduction of $SSE(b)$ due to the same split (see formula (3.5)). 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 F, can be done in $\mathcal{O}(2^d \cdot n)$. Therefore, for all the proposed greedy criteria G

but *Max-Var*^{marg}/ *Max-Red*^{marg}, $C(Evaluate(G,b)) = \mathcal{O}(2^d \cdot \eta)$, where η is the number of reductions of *SSE* or marginal *SSE* which have to be computed. In particular, $\eta = d \cdot n$ for *FBH* and *HBH*, whereas $\eta = d \cdot k$ for k-*GHBH*.
In the case that *Max-Var*^{marg}/ *Max-Red*^{marg} is adopted, the cost of computing the d marginal *SSE*s of the block is $\mathcal{O}(2^d \cdot d \cdot n)$ for either *FBH*, *HBH* and k-*GHBH*, and dominates the cost of computing the reductions of marginal *SSE*.
To sum up, when pre-computation is adopted, $C(Evaluate(G,b))$ is $\mathcal{O}(2^d \cdot d \cdot n)$ for *FBH* and *HBH*, and $C(Evaluate(G,b))$ is $\mathcal{O}(2^d \cdot d \cdot \alpha)$ for k-*GHBH* (where $\alpha = k$ for all the greedy criteria G but *Max-Var*^{marg}/ *Max-Red*^{marg}, for which $\alpha = n$).
Therefore, C^E is given by:

1. in the case that T=*FBH* or *HBH*,
   $C^E = \sum_{b \in Nodes(H)} C(Evaluate(G,b)) = \mathcal{O}(\beta_T^{max} \cdot 2^d \cdot d \cdot n)$
2. in the case that T=k-*GHBH*, $C^E = \mathcal{O}(\beta_{GHBH}^{max} \cdot 2^d \cdot d \cdot \alpha)$

Observe that in the case of *FBH* and *HBH* under any greedy criterion, as well as in the case of k-*GHBH* under *Max-Var*^{marg}/ *Max-Red*^{marg}, term C^U is negligible w.r.t. C^E. In fact, the number of buckets β is not greater than n^d, which implies $\log \beta \leq d \cdot \log n < 2^d \cdot d \cdot n$. In the case of k-*GHBH*, when a criterion different from *Max-Var*^{marg}/ *Max-Red*^{marg} is adopted, it can happen that the inequalities $\log \beta < 2^d \cdot d \cdot k$ do not hold, even though, in practical cases, the number of buckets rarely exceeds the value $2^{2^d \cdot d \cdot k}$. □

The bounds in Table 3.7.3 assume that all steps of Greedy Algorithm have the same complexity. Indeed, it is unlikely that this case occurs, since as the histogram construction goes on, smaller and smaller buckets are generated, and each of these buckets contain fewer tuples than buckets generated at previous steps. Therefore it is rather expected that, after the very first steps, Greedy Algorithm deals with buckets whose volume is much smaller than n^d, whose marginal distributions have size much smaller than n, and containing fewer tuples than N.

Experimental results comparing the efficiency of the three different approaches (the ones based on the sparse data model, the non-sparse one, and pre-computation, respectively) are provided in Section 3.9.8.

3.7.5 Workspace size for Greedy Algorithm

Implementing Greedy Algorithm with the adoption of pre-computation becomes unfeasible for high-dimensionality data, due to the explosion of the spatial complexity: the space needed to store F and F^2 grows exponentially as dimensionality increases, even if the number of non-null values remains nearly the same. In real-life scenarios, it often occurs that $N \ll n^d$, especially for high-dimensionality data: as dimensionality increases, data become sparser and the size of the data domain increases much more dramatically w.r.t. the

number of non-null data.
On the contrary, Greedy Algorithm under the sparse data model is much less sensitive on the increase of dimensionality, also from the point of view of the workspace size. In this case, Greedy Algorithm can be implemented by associating to each element of the queue not only the boundaries of the corresponding bucket b, but also the set of tuples belonging to b. Thus, when a bucket b is chosen and split into b^{low} and b^{high}, tuples of b are distributed among b^{low} and b^{high}; then the triplet $\langle need, dim, coord \rangle$ associated to the new bucket b^{low} [resp. b^{high}] is computed by scanning only the tuples belonging to b^{low} [resp. b^{high}]. That is, the partition underlying the histogram is used as an index to locate the tuples contained in the buckets. Therefore, the algorithm workspace (i.e. the storage space needed to store q) is $\mathcal{O}(d \cdot N)$ (instead of $\mathcal{O}(n^d)$, as in the case of pre-computation), since each non null point belongs to exactly one bucket of the partition at each step of the algorithm.

The complexity bounds reported in Table 3.4 show that Greedy Algorithm works in linear time w.r.t. both the number of non-null points inside D and the size of the dimensions of D. Notice that, as these bounds hold for all the considered greedy criteria, the idea of working on the one-dimensional marginal distributions of blocks does not provide a relevant benefit on the efficiency of the histogram construction w.r.t. investigating the actual multi-dimensional distributions of blocks in the greedy criterion.

3.8 Estimating range queries

As explained above, the adoption of the compressed tree-based representation model of *HBH* and *GHBH* enables a larger number of buckets to be stored within the same storage space bound w.r.t. the flat representation model of *FBH*. Intuitively enough, this is likely to slow down the evaluation of selectivity estimation on either *HBH* and *GHBH* w.r.t. *FBH*, due to the larger amount of information which is represented in the former two classes of histogram. However, In order to study in more detail the impact of the hierarchical representation on the efficiency of estimating queries, it is first shown how queries are estimated on *HBH* and *GHBH*.

Let q be a query defined on the range r, and $Sum(r)$ be the range-sum query asking for the selectivity of q. Basically $Sum(r)$ is estimated by navigating the partition tree of the histogram and summing the contributions of the nodes which overlap the query range. The visit starts from the root, which corresponds to the whole data domain. When a node u is being visited, three cases may occur:

1. *the range corresponding to u is external to r*: no contribution to the estimate is given by u;
2. *the range corresponding to u is entirely contained into r*: the contribution of u is $Sum(u)$, i.e. the sum associated with u;

3. *the range corresponding to u partially overlaps r*: if u is a leaf, linear interpolation is performed for evaluating which portion of the sum associated to the node lies onto r. If $Vol(u)$ denotes the volume of the range corresponding to u and $Vol(r \cap u)$ the volume of the intersection between r and the range of u, then the contribution of u to the query estimate is: $\frac{Vol(r \cap u)}{Vol(u)} \cdot Sum(u)$. Otherwise, if u is not a leaf, the contribution of u is the sum of the contributions of its children, which are recursively evaluated.

The volume of the range corresponding to each node is evaluated on the basis of the boundaries associated with the node. These boundaries are computed during the navigation, according to the splitting position stored in the representation of the parent node.

According to the linearized representation of the binary partition adopted in this approach, the encoding of each node u in $Array_P$ is followed by the encoding of the whole tree rooted at its left-hand child node u_{left}. Therefore, while the tree rooted at u_{left} can be directly accessed from u, linearization results in no mechanism for navigating directly from a node u to its right-hand side child node u_{right}. This implies that in order to access u_{right} starting from u, the bit string corresponding to u_{left} must be skipped. As the length of the encoding of the sub-tree rooted at u_{left} is not fixed (since it depends on the depth of the sub-tree), it is necessary to scan the encoding of this sub-tree in order to reach the position in $Array_P$ where the encoding of u_{right} starts from. However, scanning the representation of the sub-tree rooted at u_{left} results in an overhead only if u_{left} does not contain any node overlapping the query range (otherwise, it should be scanned in order to compute its contribution to the query estimate anyhow). Indeed, even if this is the case, the cost of accomplishing this scanning is unlikely to slow down the query estimation time significantly. In fact, the tree rooted at u_{left} can be rapidly scanned by accessing only the bits encoding its structure, that is the bits saying whether the corresponding nodes are leaves or not. Consider two nodes u_1, u_2 such that u_2 immediately follows u_1 according to the prefix visit of the partition tree. Then the distance between the encodings of u_1 and u_2 is exactly the length of the bit encoding of u_1. This length is a constant depending on whether u_1 is a leaf or an intermediate node. That is, if u_1 is a leaf this distance can be either 1 or 2: it is 1 if u_1 is the right-hand child of its parent node, whereas it is 2 if u_1 is the left-hand child of its parent node (see nodes $D.1$ and $D.2$ in Fig. 3.6). Otherwise, if u_1 is an intermediate node, the distance between the starting points of the encodings of u_1 and u_2 in $Array_P$ is equal to the length of the encoding of the splitting position and the splitting dimension of u_1 (i.e. $\lceil \log_2 d \rceil + 32$ for *HBH*, and $\lceil \log_2 d \rceil + \log_2 k$ for *GHBH*). Therefore, when the tree rooted at u_{left} does not overlap the query range, its encoding is scanned by skipping the fragments of $Array_P$ encoding the splitting positions and the splitting dimensions of its nodes. The bits encoding the splitting position of nodes must be unpacked only for those nodes giving a non-null contribution to

the query estimate, that is those nodes belonging to branches of the partition tree overlapping the query range.

Therefore a histogram of type *HBH* or *GHBH* can be navigated by rapidly scanning the branches of the partition tree corresponding to portions of the data domain which are not involved in the query, and executing the complete unpacking of node boundaries only for those branches overlapping the query range. Moreover, observe that storing the sums at intermediate nodes of the tree further enhances the efficiency of the estimation: if the range of an intermediate node u is completely contained inside the range of the query, the estimated answer is increased by the sum associated with node, and the tree rooted at u is rapidly scanned, without decoding the boundaries of the nodes descending from u and without accessing their sums.

As regards *FBH*, its flat representation model does not provide any mechanism for efficiently locating the buckets involved in the query. Thus estimating $Sum(r)$ requires accessing all the buckets of the histogram to check whether they overlap r. In order to check whether a bucket b overlaps r it is necessary to comute $\mathcal{O}(d)$ intersections: first it is checked whether b intersects r along dimension 1; if this is the case, the same verifying is accomplished for dimension 2, and so on, until a dimension where b and r do not overlap is found, or all dimensions have been checked. Therefore $\mathcal{O}(b \cdot d)$ intersections between one-dimensional ranges should be computed to determine the list of buckets giving non null-contribution to the query answer.

Observe that for either *HBH* and *GHBH* the number of one-dimensional intersections which must be computed to locate the buckets involved in the query is less than the latter amount (for the same value of b). In fact for each intermediate node visited in the partition tree at most one 1-dimensional intersection must be accomplished, according to the splitting dimension and the splitting position stored in the node representation. Thus the number of intersections to be performed is at most b (as the number of intermediate nodes is the same as the number of leaves). Moreover, the value b is an upper bound: as explained above, intersections must be computed only for those branches descending from nodes overlapping the query range.

The above-reported reasoning does not suffice to state that accomplishing query estimates on *HBH* and *GHBH* is more efficient than the case of *FBH*, as the number of buckets stored by *HBH* and *GHBH* is larger than that of *FBH* within the same storage space bound (as explained in Section 3.5). Moreover the cost of the bit-wise decoding of the tree structure should be taken into account. The relevance of this overhead in a practical setting cannot be measured but experimentally. Section 3.9.7 provides several experiments comparing the efficiency of answering queries when the *FBH* flat representation model and the *GHBH* hierarchical representation model are employed, respectively. Overall, experimental results show that adopting the compressed hierarchical representation model results in better accuracy and efficiency of query answering.

3.9 Experimental analysis

In this section a thorough experimental analysis investigating several issues related to the performances, in terms of accuracy and efficiency, provided by histograms based on binary partitions is presented. The attention will be focused on different issues, in order to study progressively the impact on the histogram accuracy of the following contributions: the specific tree-based representation model of *HBH*, the grid-constrained partition scheme of *GHBH*, and the heuristics used to accomplish the construction of the histogram. Finally the efficiency of query evaluation using the compressed representation model will be studied, comparing it with the *FBH* representation model.

The experiments were conducted on both synthetic and real-life data. In the following two sections the synthetic data generator and the real-life data sets used to perform experiments are described.

3.9.1 Synthetic data

The adopted synthetic data generator is similar to those of [45] and [128]. It takes as argument the following parameters: $n_1, \ldots, n_d, T, m, l_{min}, l_{max}, z_{min}, z_{max}$. Data are generated by creating m dense regions inside a d-dimensional array D with volume $n_1 \times \cdots \times n_d$. These dense regions will be denoted as $r_1, \ldots, r_m$. Each r_i is characterized by its center c_i, its width l_i and the *skew parameter* z_i (as it will be clear later, z_i determines the data distribution inside r_i). The coordinates of the centers $c_1, \ldots, c_m$ are generated according to a uniform distribution on the domain of D; the widths $l_1, \ldots, l_m$ are randomly chosen between l_{min} and l_{max}, and $z_1, \ldots, z_m$ are randomly chosen between z_{min} and z_{max} (in the experiments, where it is not differently stated, $z_{min} = 0.5$ and $z_{max} = 2.5$). Initially D is empty (i.e. it contains only null points), and at the end of the generation process it will contain a number of points whose sum is equal to T. In particular, T is divided into T_{noise} and T_{reg}. The latter is further divided into m portions $T_1, \ldots, T_m$ according to a uniform distribution. Each T_i represents the sum of the points inside region r_i. Region r_i is populated in two steps:
1. a number T_i of points inside the range of r_i (namely, $p_1, \ldots, p_{T_i}$) are generated. Each of these points p_j is obtained by first generating its distance δ_j from the center c_i and then randomly choosing p_j among the points having this distance from c_i. The value of δ_j is chosen according to a Zipf distribution on $[0..l_i]$ with parameter z_i.
2. for each p_j (with $j \in [1..T_i]$) the value of $D[p_j]$ is increased by one. Thus, each cell p in r_i will contain the number of occurrences of p in the sequence $p_1, \ldots, p_{T_i}$.

As regards T_{noise}, it is used to simulate noise in the data distribution: it is distributed among randomly generated points inside D (in the experiments $T_{\text{noise}} = 0.05 \cdot T$).

This generation paradigm results in a data distribution where, for each r_i, the higher z_i, the more "concentrated" around c_i the data points: as z_i increases, points having large distances from c_i are less probable to be generated.

As explained in [45] and [128], data-sets generated by using this strategy well represent many classes of real-life distributions.

3.9.2 Real life data

Experiments on two real-life data sets were performed. The first data set will be referred to as *Census* and was 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, containing 150 943 tuples, was extracted. 8 attributes have been chosen:

Age, *Parent's line number*, *Major Occupation*, *Marital Status*, *Race*, *Family Type*, *Public Assistance Type*, *School Enrollment*. The corresponding 8-dimensional array has about $4.6 \cdot 10^7$ cells, and contains 19129 non-null values (the density is about $4.2 \cdot 10^{-4}$).

The other data set will be referred to as *Forest Cover*. It was obtained from the U.S. Forest Service and is available at the UCI KDD archive site. It consists of 581 012 tuples having 54 attributes. Among these, 10 attributes are numerical. As in [56, 57], the tuples projected on these numerical attributes are considered, thus obtaining a 10-dimensional data distribution. The corresponding 10-dimensional array has about $4.4 \cdot 10^{28}$ cells (the density is about $1.3 \cdot 10^{-23}$).

3.9.3 Experimental plan

Given a data distribution D, let r be a range inside the domain of D. $Sum(r)$ will denote the range-sum query asking for the sum of values stored in the cells of D inside r. In the experiments D represents a frequency joint distribution, thus the value of $Sum(r)$ represents the selectivity of a query defined on the range r. Let $Sum'(r)$ the estimate of $Sum(r)$ evaluated on the histogram. The *absolute error* of the estimate is defined as: $e^{abs} = |Sum(r) - Sum'(r)|$. The *relative error* is defined as: $e^{rel} = \frac{e^{abs}}{Sum(r)}$. Observe that relative error is not defined when $S_i = 0$. Indeed, for queries with very low exact answer, the absolute error of the estimate gives a better idea of the estimate accuracy than the relative error.

The accuracy of the various techniques was evaluated by measuring the absolute error and the relative error of the estimates of range-sum queries computed by accessing the histogram. The impact of a number of parameters on the accuracy was considered: the amount of storage space used to represent the histogram, the selectivity of queries, as well as several characteristics of

the input data (such as dimensionality and domain size). The sensitivity to each of these parameters was analyzed by varying it and fixing the other ones. In particular, in order to generate groups of queries with the same selectivity on a data distribution D, the following strategy was used. First, a number of distinct points were randomly selected inside D. Then, for each of these points p, a set of queries was generated starting from the query whose range coincided with p and by progressively enlarging the query volume. This resulted in queries centered in p with increasing selectivity. Finally, such obtained queries were grouped by their selectivity.

The following sections are organized as follows.

First, in Section 3.9.4 the impact of adopting different greedy criteria under different representation models is studied. To this aim, *FBH* and *HBH* are compared, when all greedy criteria of Table 3.3 are used, in order to establish which combination yields the best accuracy. In this section a 4-dimensional synthetic data is used, with volume $400 \times 400 \times 400 \times 400$ containing one million tuples distributed among 30 dense regions. All the greedy criteria combined with the two different representation models on the *Forest Cover* data-set are also compared.

Then, in Section 3.9.5 the impact due to the grid-partitioning by comparing *HBH* and *GHBH* is investigated. This analysis will show that the best performances among all different combinations *type of histogram*/*greedy criterion* is provided by histograms of type *GHBH* constructed by adopting *Max-Var*/*Max-Red*. Moreover, the sensitivity of the accuracy provided by *GHBH* w.r.t. different characteristics of the data distribution, such as the size of dimensions and the data skewness for different data dimensionality, is studied.

In Section 3.9.6 the accuracy provided by *GHBH* using *Max-Var*/*Max-Red* with the state of the art is compared. In particular, experiments studying how the accuracy of the various techniques is affected by data dimensionality are presented.

Section 3.9.7 provides experiments testing the impact of the adoption of the compression-based representation model of *GHBH* on the efficiency of estimating queries.

Finally, in Section 3.9.8 experiments showing the construction time of the histogram are presented. These experiments aims at comparing the practical efficiency of the histogram construction when either sparse or non-sparse data model, or pre-computation is adopted.

3.9.4 Comparing *FBH* and *HBH* under different greedy criteria

As explained in Section 3.3, the tree-based representation models of *HBH* and *GHBH* are an alternative to the *MBR*-based one of *FBH*. On the one hand, the tree-based representation model yields a larger number of buckets w.r.t. the *MBR*-based one (within the same storage space bound); on the other hand, buckets represented by means of their *MBR*s are likely to provide a more accurate description of the underlying data distribution. Therefore it is worth

investigating which of these alternatives yields the best accuracy and, in more detail[5]. Therefore it was studied how the accuracy provided by the different representation models of *FBH*, *HBH*, *GHBH* depends on the particular greedy criterion adopted to guide the histogram construction.

In order to establish how using different greedy criteria and employing different representation models affect the histogram effectiveness, the accuracy of histograms obtained using different combinations *greedy criterion / representation model* was studied. In particular, this section provides experimental results studying how error rates change when the same greedy criterion is used with either the flat representation model of *FBH* and the tree-based representation model of *HBH*. The impact of the use of the grid-constrained partitioning of *GHBH* will be considered in more detail in the following section.

Observe that the combinations $Max\text{-}Red^{marg}/FBH$ and $MaxDiff/FBH$ coincide with the techniques *Min-Skew* and *MHIST*, respectively. However, the aim of this section is not to provide a complete comparison between the new compression approach and these well-known techniques: further experiments comparing the new approach with *Min-Skew* and *MHIST* will be reported in Section 3.9.6.

Diagrams in Fig. 3.10 (a,b) were obtained on a synthetic 4-dimensional data distribution with volume $400 \times 400 \times 400 \times 400$ containing one million tuples distributed among 30 dense regions (with $l_{min} = 30$ and $l_{max} = 80$), whereas diagrams in Fig. 3.10(c,d) were obtained on the 10-dimensional Forest Cover data set. The accuracy of the various criteria is evaluated w.r.t. the *storage space* available for the compressed representation.

The main results which turn out from diagrams in Fig. 3.10 are the following:

1. for both *FBH* and *HBH*, *Max-Var/Max-Red* provides lower error rates than all other criteria; moreover, this criterion exploits the amount of storage space more effectively: as the available storage space increases, error rates decrease more rapidly w.r.t. the other criteria;
2. the accuracy of histograms built by employing any criterion other than *Max-Var/Max-Red* and *Max-Red* is nearly the same when either the *FBH* or *HBH* representation model is adopted. On the contrary, histograms constructed adopting either *Max-Var/Max-Red* or, to a lesser extent, *Max-Red* benefit from the use of the *HBH* representation model.

The main result is that greedy criteria behave differently depending on whether the *FBH* representation model is used or not; interestingly, some greedy criteria benefit from the adoption of this model, whereas other criteria yield lower error rates when the tree-based representation model of *HBH* is

[5] Obviously, only *FBH* adopting *MBR*s to represent the leaves of the partition tree will be considered: otherwise, the comparison of *FBH* with *HBH* and *GHBH* (in terms of accuracy) is rather expected.

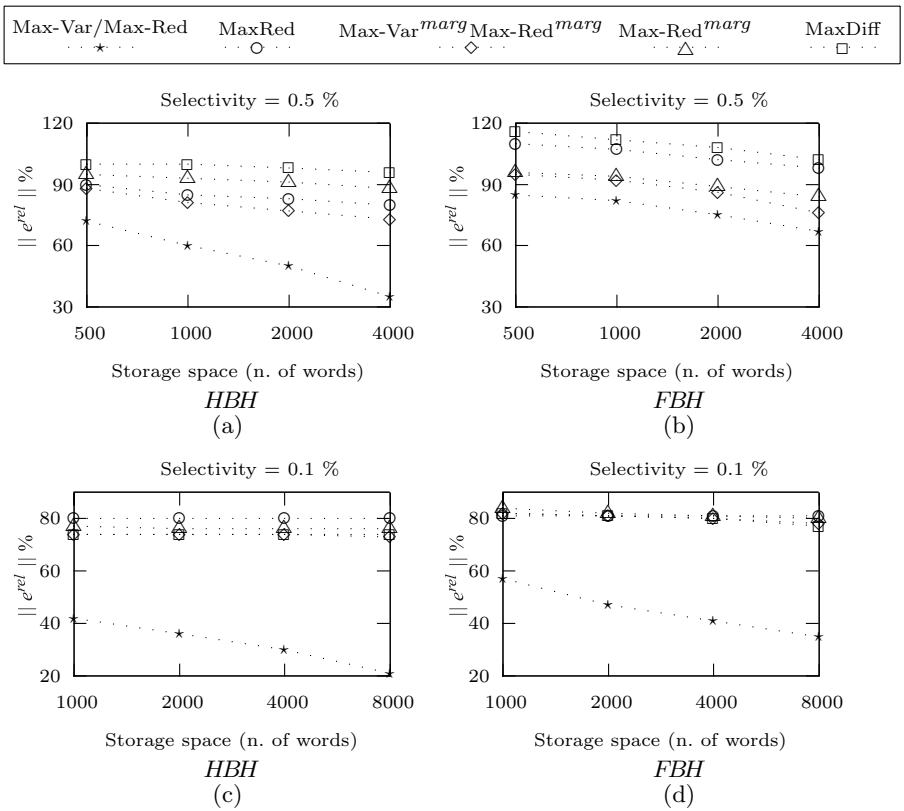


Fig. 3.10. Comparing combinations *greedy criterion / representation model* on 4D-synthetic data (a,b) and Forest Cover (c,d)

adopted.
For both *FBH* and *HBH*, *Max-Var/Max-Red* provides lower error rates than all other criteria, but its accuracy is higher under the tree-based representation model. Moreover, this criterion exploits the amount of storage space more effectively: as the available storage space increases, error rates decrease more rapidly w.r.t. the other criteria.

In order to explain why employing different greedy criteria results in different error rates, a two-dimensional data distribution was considered and it was studied how data is partitioned depending on the adopted criterion for both *HBH* and *FBH*. Partitions resulting from different combinations *greedy criterion / representation model* are depicted in Fig. 3.11. In order to explain why different combinations *greedy criterion / representation model* result in different error rates, let consider a two-dimensional data distribution and study how data is partitioned depending on the adopted combination. Indeed this is not exhaustive, since error rates in higher dimensionality settings depend also on

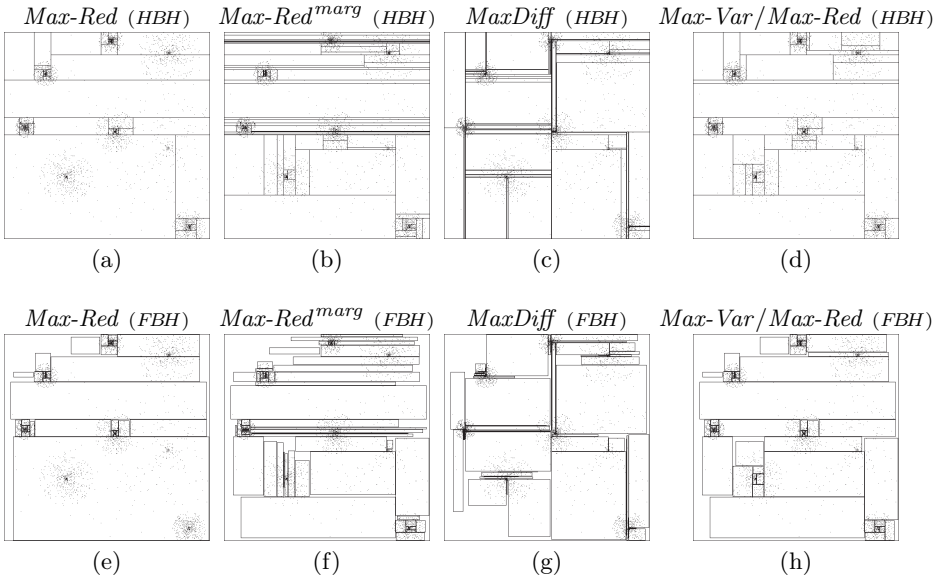


Fig. 3.11. Partitions resulting from different combinations *greedy criterion / representation model*

issues which do not significantly affect the accuracy in the two-dimensional scenario (for instance, *Max-Var/Max-Red* and *Max-Var*^{marg} */Max-Red*^{marg} yield about the same accuracy in the 2D scenario, but significantly differ from one another on higher dimensionality data).

By analyzing diagrams in Fig. 3.11, the following considerations can be drawn:

- *Max-Red* fails in building effective partitions, as it tends to progressively split "small" dense regions. This is made clear in Fig. 3.11(a) and (e): some clusters of data are rather disregarded, even if the amount of storage space would suffice to perform enough splits to isolate them; indeed, many splits are "wasted" to partition the core of other dense regions. This behavior can be explained by analyzing Equation 3.4 which expresses the reduction of *SSE* of a block b due to the binary split $\langle b^l, b^h \rangle$. From that formula, it turns out that splitting a small dense block b_1 can result in a larger reduction of *SSE* w.r.t. splitting a larger block b_2, even if $SSE(b_2) \gg SSE(b_1)$ [6]. The use of *MBR*s does not improve error rates of *Max-Red*. In fact, on the one hand *MBR*s do not prevent the criterion from generating several small buckets in a few dense regions. On the other hand, the use of *MBR* reduces the number

[6] As a hint, consider two buckets b_1, b_2 with $vol(b_1) \ll vol(b_2)$ containing the same data values distributed differently. Then $SSE(b_2) \gg SSE(b_1)$ holds, but the reduction of *SSE* due to any binary split $\langle b_1^l, b_1^h \rangle$ of b_1 is likely to be larger w.r.t. any binary split $\langle b_2^l, b_2^h \rangle$ of b_2.

of splits, so that a larger number of dense regions are likely to be disregarded by the partition;
- the behavior of *Max-Red*^{marg} can be explained as for *Max-Red*. In this case, the criterion tends to choose blocks having small size along one of their dimensions, and split them along this dimension, as this yields the maximum reduction of (marginal) *SSE*. This explains the shape of the partitions generated by *Max-Red*^{marg} shown in Fig. 3.11(b) and (f), where rectangular blocks are split along their smallest dimension, and the obtained blocks are recursively split along the same dimension. Differently from *Max-Red*, criterion *Max-Red*^{marg} can provide better performances (in terms of accuracy) when *MBR*s are used. In fact, *Max-Red*^{marg} performs several parallel splits making some dimensions of blocks to be never partitioned (for instance, in Fig. 3.11(b) several buckets are obtained by never splitting the horizontal dimension of the data domain). This results in blocks putting together dense regions with null ones, and these regions are never separated from one another, as a split along the largest edge of blocks would be necessary. In this case *MBR*s can improve the accuracy of the partition, as they fit the shape of buckets on the underlying distribution, thus isolating dense regions from null ones. Moreover, for the same reason, the adoption of *MBR* can reduce the differences in length between the edges of each block, thus limiting the number of parallel splits performed recursively on the same block.
- *MaxDiff* generally yields higher accuracy when *MBR*s are employed. However error rates are not satisfactory in both cases. As shown in Fig. 3.11(c) and (g), adopting *MaxDiff* results in partitions which poorly describe the underlying data, as this criterion is unable to separate dense regions from null ones. In fact there is no reason to assume that largest differences in marginal values arise correspondingly to the boundaries of dense regions.
- *Max-Var*/*Max-Red* succeeds in locating dense regions where a finer-grain partition of data is needed: as shown in Fig. 3.11(d) and (h) this criterion is fairer in choosing the region to be split w.r.t. *Max-Red*, *Max-Red*^{marg}, and *MaxDiff*. This explains why it outperforms all the other criteria for both *HBH* and *FBH*.

In Fig. 3.11 the partition obtained with *Max-Var*^{marg}/*Max-Red*^{marg} is not shown, as it was very similar to that of *Max-Var*/*Max-Red*. Indeed differences in error rates between these two criteria arise more significantly in higher dimensionality settings. As dimensionality increases, marginal distributions contain less and less information on the internal content of blocks: in fact, the total size of marginal distributions of a block (i.e. the sum of the lengths of all marginal distributions of the block) grows linearly with d (it is $\mathcal{O}(d \cdot n)$), whereas the volume of blocks grows exponentially with d (it is $\mathcal{O}(n^d)$). Therefore, investigating the content of marginal distributions to decide whether a block needs to be partitioned is likely to provide less reliable information as dimensionality increases. As a matter of fact, isolated dense regions of the multi-dimensional space can collapse into a single mono-

dimensional dense region, when projected on each dimension (for instance, consider two adjacent circular dense regions located at the ends of a diagonal in the two-dimensional space). Therefore, it is unlikely to succeed in isolating the dense multi-dimensional regions by only looking at their projections (i.e. the marginal distributions) on space dimensions.

The attention is now focused on explaining why only *Max-Var/Max-Red* and *Max-Red* benefits from the adoption of *HBH* representation model. As regards *Max-Var/Max-Red*, the reason for this is that *MBR*s do not help this criterion in isolating dense regions from null ones: therefore, *Max-Var/Max-Red* can exploit significantly the increase in number of buckets due to the *HBH* representation model, investing a larger number of buckets to approximate dense regions in more detail (in fact, error rates for *Max-Var/Max-Red* are very sensitive to an increase in storage space, as shown in diagrams in Fig. 3.10). As regards *Max-Red*, on the one hand *MBR*s do not prevent this criterion from generating several small buckets in a few dense regions. On the other hand, the use of *MBR* reduces the number of splits, so that a larger number of dense regions are likely to be disregarded by this criterion (for instance, the dense region at the bottom right-hand part of the data distribution in Fig. 3.11(e) is not partitioned when *MBR*s are used).

On the contrary, all other criteria are not effective in assigning distinct dense regions to distinct buckets: in this case, *MBR*s do provide a more accurate description of the data underlying buckets, but this positive benefit of *FBH* representation model turns out to be counterbalanced by the smaller number of buckets w.r.t. the *HBH* representation model.

Remark. In [35] a different version of *MHIST* consisting in adopting a partition-tree-based representation scheme instead of *MBR*s was proposed. This basically consists in combining the *Max-Diff* criterion with the *HBH* representation model. As it emerges from experiments, that naif adoption of a tree-based representation scheme is not likely to achieve significant benefits w.r.t. the original *MHIST*, which combines the *Max-Diff* criterion with the *FBH* representation model.

Due to the higher level of accuracy provided by *Max-Var/Max-Red* under the tree-based representation model, and since this criterion can be evaluated as efficiently as the other ones (as stated in Theorem 3.2), in the following only *HBH* using this criterion will be considered.

3.9.5 Comparing *HBH* with *GHBH*

In this section it is studied how the introduction of a grid constraining block splits affects the accuracy of histograms. *GHBH* and *HBH* are first compared under the same greedy criterion (thus *Max-Var/Max-Red* is adopted, under

which *HBH* yields the best accuracy), then the impact of adopting different greedy criteria to guide the construction of a *GHBH*is briefly discussed.

HBH vs *GHBH* under *Max-Var/Max-Red*

Histograms of type *GHBH* with different grid degrees have been tested. In the following, the term *GHBH*(x) will be used to denote a *GHBH* which employs x bits to store the splitting position.

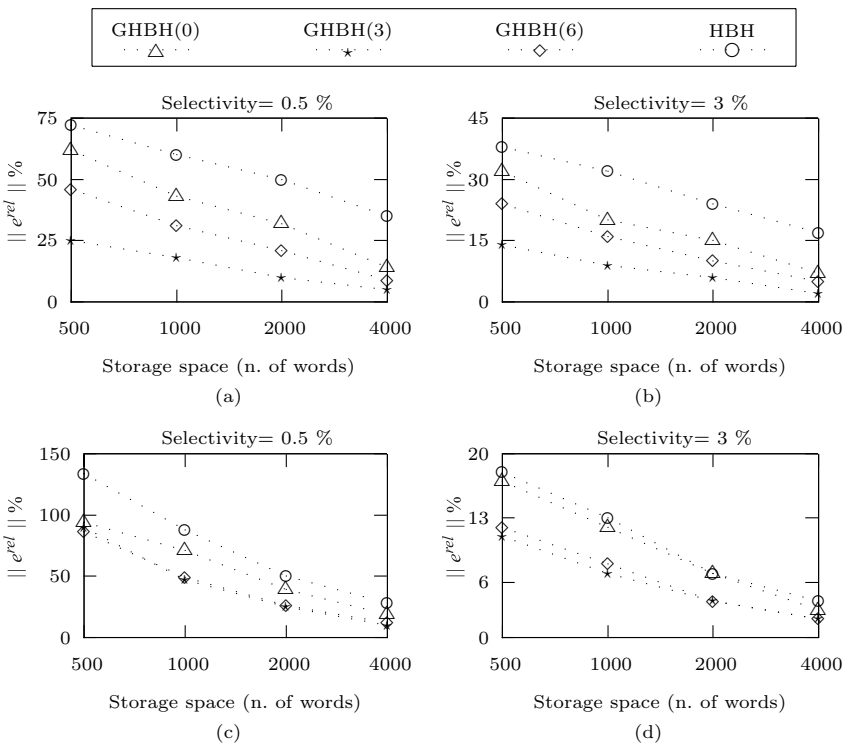


Fig. 3.12. *HBH* vs *GHBH* on 4D-synthetic data (a,b) and real-life data (c,d)

Diagrams in Fig. 3.12 (a,b) were obtained on the same 4-dimensional synthetic data distribution as Fig. 3.10, whereas diagrams in Fig. 3.12 (c,d) were obtained on Census data set by performing samples of queries whose selectivity is respectively 0.5% and 3%.

From diagrams in Fig. 3.12 it emerges that *GHBH* yields higher accuracy than *HBH*. Although the *HBH* algorithm is able to perform more effective splits at each step (as splits are not constrained by the grid), the number of buckets generated by *GHBH* algorithms is larger.

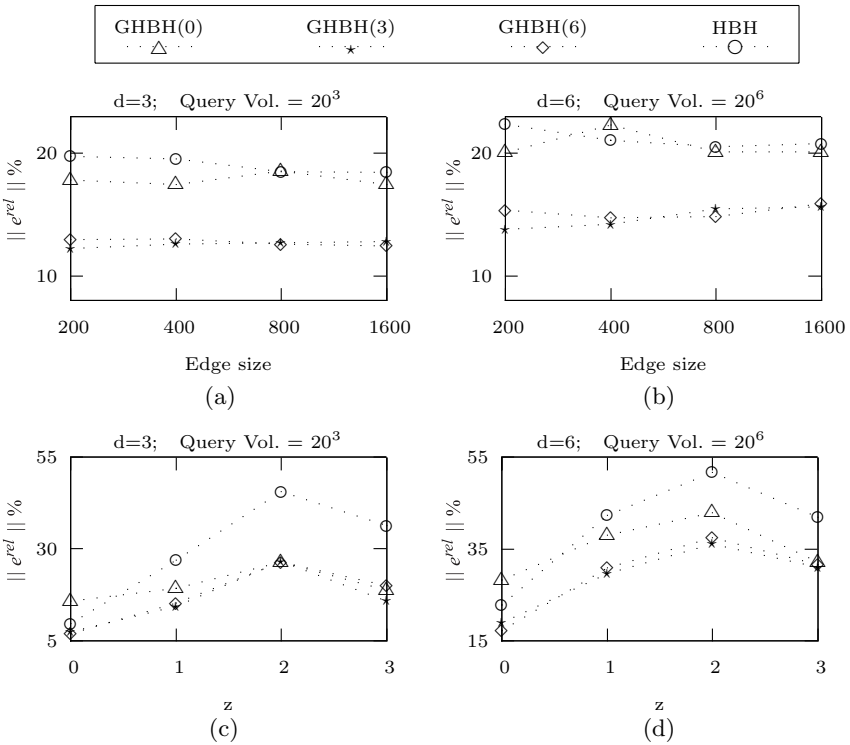


Fig. 3.13. Error rates of *HBH* and *GHBH* versus length of dimensions (a,b) and data skewness (c,d)

In order to investigate in more detail how the optimal grid degree depends on the characteristics of data, experiments studying how the accuracy of *GHBH*s adopting grids with different degrees is affected by either the size of the domain and the data skewness were performed. As regards the former issue, intuition would suggest that, as the volume of data increases, in order to keep the same accuracy in partitioning data, a higher-degree grid should be adopted. To analyze this aspect, some experiments investigating how changing the grid degree affects the effectiveness of isolating dense regions distributed on larger and larger domains were performed.

To this aim, *GHBH*s with different grid-degrees on data distributions having the same dimensionality were tested, increasing volume, and containing the same dense regions differently distributed in the data domain. Diagram 3.13(a) depicts error rates on 3-dimensional cubic data sets with increasing lengths of dimensions, from 200 to 1600. These data sets will be denoted as D^n, where n is the length of each dimension. $D^{200}, \ldots, D^{1600}$ were generated by first creating 30 dense regions, and then distributing randomly the centers of these regions in the different data domains. For instance, each dense region r_i (with

$i \in [1..30]$) inside D^{400} contains the same distribution of values as the region r'_i inside D^{200}, but the center c_i of r_i has different coordinates w.r.t. the center c'_i of r'_i (c_i and c'_i being randomly selected points inside D^{400} and D^{200}, respectively.). For each data set D^n a query set QS^n is generated as follows. QS^{200} contains, for each dense region r_i of D^{200}, 1000 hypercubic queries intersecting r_i. The centers of these queries are characterized by their relative coordinates to c_i. QS^{400} contains, for each $q \in QS^{200}$ involving r_i, a query q' involving r'_i with the same volume as q, and whose center has the same relative coordinates to c'_i as q does to c_i. Query sets QS^{800}, QS^{1600} have been constructed analogously. Evaluating error rates w.r.t. these query sets allows us to establish whether the optimal grid degree for a *GHBH* depends on the domain size. Diagram 3.13(b) was obtained analogously, but for 6-dimensional data.

Diagrams in Fig. 3.13(a,b) show that the effectiveness of adopting a particular grid degree is almost unaffected by the size of the domain.

Diagrams in Fig. 3.13(c,d) show how error rates depend on data skewness. Diagram in Fig. 3.13(c) depicts error rates for 3-dimensional data distributions with volume $400 \times 400 \times 400$ having 30 dense regions (having the same value z of the skew parameter), whereas diagram in Fig. 3.13(d) was obtained on 6-dimensional data distributions with volume 400^6 with 30 dense regions. As for diagrams 3.13(a,b) a query set consisting of queries with the same volume overlapping dense regions was considered. Observe that data density decreases as skewness increases: as z gets larger, dense regions become sparser, since tuples are generated with a decreasing probability of being far from the center of the region. That is, when $z = 0$ the distances of generated tuples from the center are uniformly distributed, while as z becomes larger, large distances from the center become less and less probable. Going to the limit, dense regions collapse to a unique cell.

From diagrams in Fig. 3.13(c,d) it turns out that "intermediate" values of z yield the largest error rates. Indeed, if skew is either very low or very high, dense regions will be rather uniform, or collapse, respectively. In both cases, isolating the dense region into a few buckets suffices to have good accuracy, whereas for intermediate values of z dense regions need more and more splits to be accurately described by the partition.

From these results, it is possible to draw the conclusion that the use of constraints on the splitting position 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 homogeneous 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 the experimental results, it emerges that *GHBH*(3) (using binary splits of degree 8) generally gives the

best performances in terms of accuracy, and as the number of bits used to define the grid increases, the accuracy decreases. However, *GHBH*s with small degree values do not exhibit large differences in error rates. Therefore, even if a value for the grid degree yielding the best accuracy in any setting cannot be found, this is not a limit of the approach, as any low-degree grid provides error rates which are close to those of the "optimal" degree. In the rest of the paper, all results on *GHBH* will be presented by using 3 bits for storing splitting positions.

HBH vs *GHBH* under different greedy criteria

It is worth noting that the improvement of *HBH* accuracy obtained by introducing the grid constraint is not simply orthogonal to any greedy criterion of Table 3.3. Although the benefits of using grids only when *Max-Var/Max-Red* is used were discussed, from several experiments it turned out that, in the case that another greedy criterion is adopted, the improvement in accuracy when moving from *HBH* to *GHBH* is not so relevant. This result is rather expected, and related diagrams are not reported. In fact, it is worth noting that *Max-Var/Max-Red* is the only criterion which improves significantly as the available storage space becomes larger (see Fig. 3.10), whereas the other criteria are less sensitive to this parameter.

Therefore, it is unlikely that the other criteria exploit the increase of the number of available splits due to the introduction of grids as *Max-Var/Max-Red* does. Therefore in the following only *GHBH* adopting *Max-Var/Max-Red* will be considered.

3.9.6 *GHBH* versus other techniques

The effectiveness of *GHBH* is compared with the state-of-the-art techniques for compressing multi-dimensional data. *GHBH* was compared with MHIST [111], Min-Skew [3], GENHIST [56, 57] and two wavelet-based techniques proposed in [127] and [128]. The experiments were conducted at the same storage space.
We have considered the two wavelet-based techniques presented in [128] (that will be referred as WAVE1) and in [127] (WAVE2). The former applies the wavelet transform directly on the source data, whereas the latter performs a pre-computation step. First, it generates the partial sum data array of the source data, and replaces each of its cells with its natural logarithm. Then, the wavelet compression process is applied to this array.

Diagrams of Fig. 3.14 were obtained on four-dimensional synthetic data with volume $8 \times 16 \times 256 \times 1024$, with $T = 35\,000$ having 30 dense regions (the obtained density is about 0.1%). In particular diagrams (a,b) show how the accuracy of the techniques changes as the storage space increases, whereas diagrams (c,d) depict error rates w.r.t. the selectivity of queries.

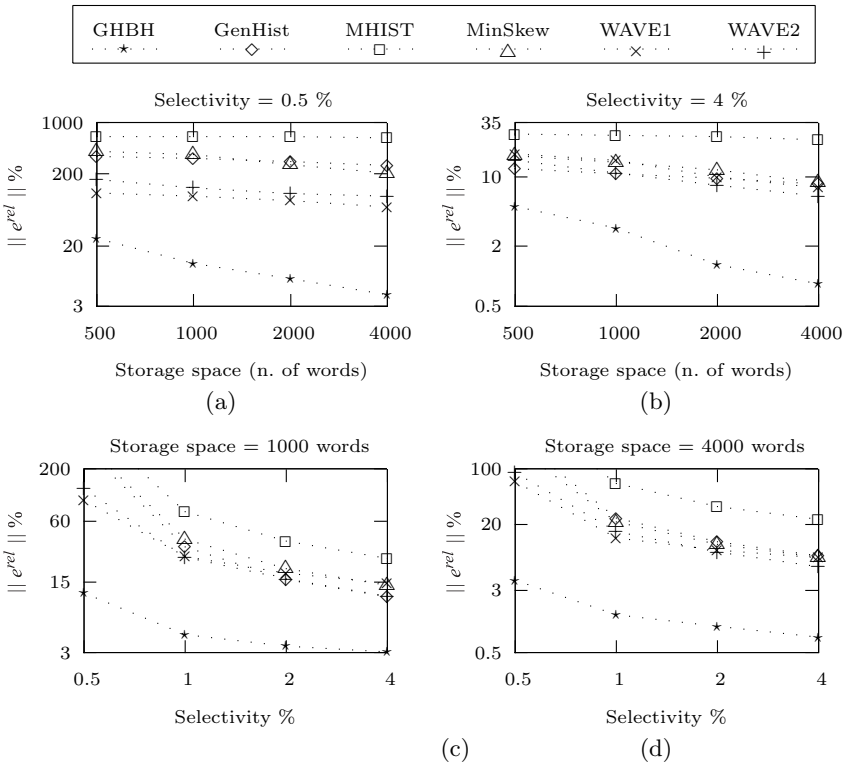


Fig. 3.14. Comparing techniques on synthetic data

GHBH exploits the increase of storage space better than the other techniques. Relative error rates of all techniques increase as query selectivity decreases: this can be easily explained by considering that as selectivity decreases (i.e. query answers become smaller in value) even a small difference between the actual query answer and the estimated one can lead to a large relative error.

Diagrams of Fig. 3.15 were obtained on the 10-dimensional Forest Cover data set. On this data set wavelet techniques were not considered as the adopted prototype does not support data sets with so large volumes. Diagram of Fig-3.15(c) refers to very low selectivities and reports absolute errors. In this case relative errors have not been considered as, for small selectivities, relative error is likely be high even if a reasonable approximation is obtained: for instance, while a 300% error rate on a query with selectivity 1 corresponds to a good accuracy of the estimate, the same error rate on a query whose selectivity is high (w.r.t. N) makes the inaccuracy of the query estimate intolerable. Thus relative error may not be indicative of the actual accuracy. In diagram (h) relative error rates at higher selectivities are reported. The query workload

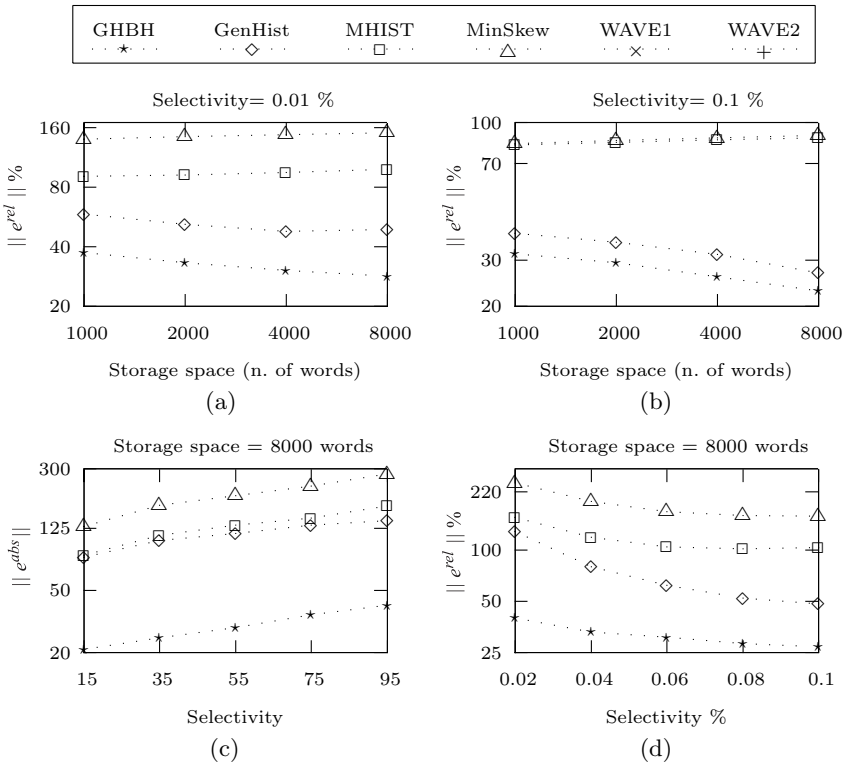


Fig. 3.15. Comparing techniques on Forest Cover

for obtaining diagrams (c,d) was constructed by first randomly generating 10 000 query centers in the data domain; then, for each of these centers, queries with increasing selectivity were generated by progressively enlarging the query volume, till 0.1% selectivity is reached. Finally, the results on the accuracy of the answers were grouped by the query selectivity. Observe that queries having the same selectivity may have very different volumes. In particular, in the case of Forest Cover Type data set, as data are very sparse, even queries with very large volumes can have very low selectivity.

Diagrams of Fig. 3.16 were obtained on the 8-dimensional Census data set. In particular, in this case a broader range of selectivities is considered, as this data set is less sparse than Forest Cover and thus queries with very low selectivities are unlikely to occur.

All the diagrams show that *GHBH* adopting *Max-Var/Max-Red* outperforms the other techniques on all the examined data sets.

Remark. From all the diagrams presented in this section it emerges that state-of-the-art techniques yield acceptable accuracy only on high selectivity queries, while excessive error rates (above 100 %) are obtained on low selec-

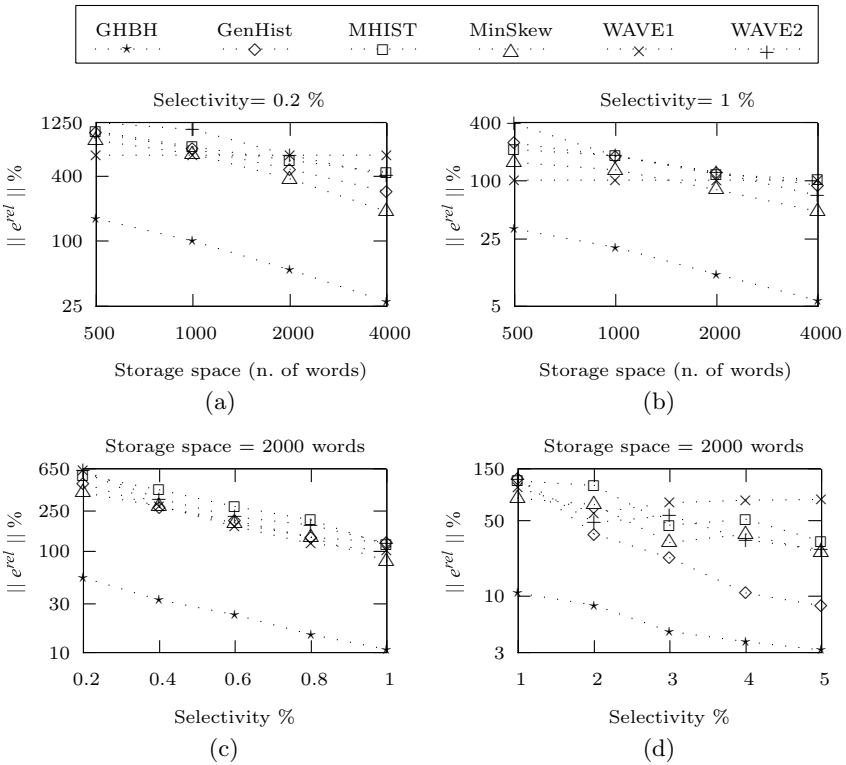


Fig. 3.16. Comparing techniques on Census

tivity queries. It is worth noting that in some contexts low selectivity queries are those most "meaningful". For instance, when data are very sparse, high selectivity queries are mainly characterized by a very wide range. This is especially clear in the high-dimensional context. For example, 0.1 % selectivity range queries on the 10-dimensional Forest Cover Type data set have an average volume equal to $4 \cdot 10^{25}$, which means that, for each dimension, the edge of the query range has to be about half the size of the data domain. Therefore, state-of-the-art techniques are not suitable for most of the significant queries in the high-dimensionality contexts. *GHBH*, instead, enables acceptable approximations even when the query selectivity is very low, which is the typical case, as explained before, occurring when dealing with a large number of dimensions. In the following it is investigated in detail how the accuracy of the various techniques is affected by an increase in dimensionality of the input data.

Diagrams 3.17(a,b) refer to synthetic data. These diagrams were obtained by starting from a 10-dimensional data distribution (called D^{10}) containing 10^{20} cells (the size of each dimension is equal to 100), where about 53000 non

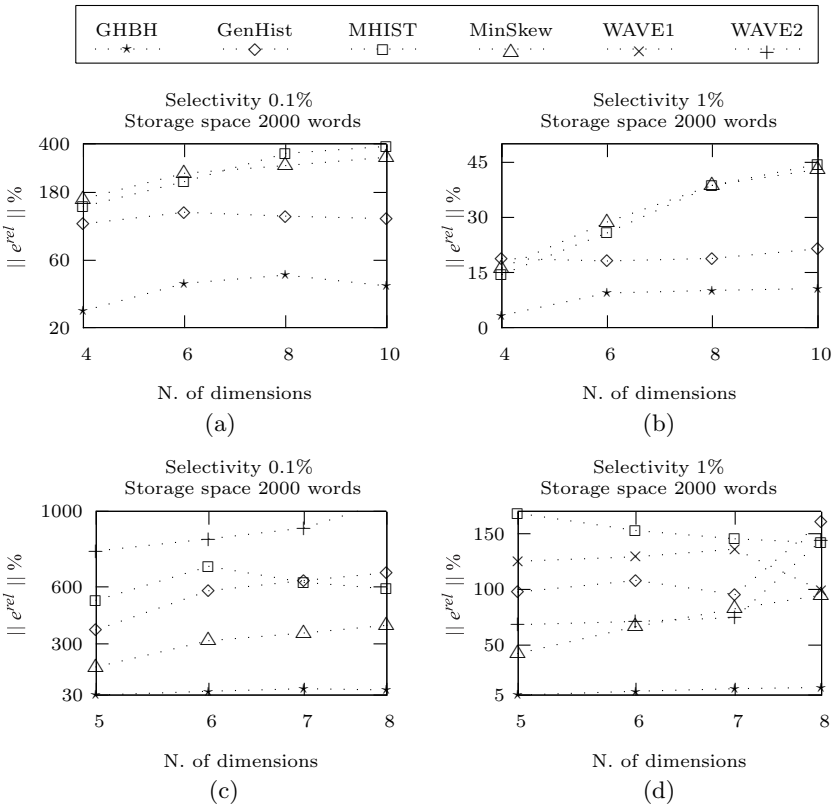


Fig. 3.17. Sensitivity to dimensionality on synthetic data (a,b) and Census data (c,d)

null values (density $\simeq 5.3 \cdot 10^{-16}$) are distributed among 1000 dense regions. The data distributions with lower dimensionality (called D^i, with $i \in 4..9$) were generated by projecting the values of D^{10} on the first i of its dimensions. By means of this strategy, a sequence of multi-dimensional data distributions was created, with increasing dimensionality (from 4 to 10) and with decreasing density (from $3.9 \cdot 10^{-4}$ to $5.3 \cdot 10^{-16}$). Diagrams 3.17(a,b) were obtained by considering, for each D^i, a sample of range queries whose selectivity is respectively 0.1% and 1%. Both these diagrams were obtained using a storage space of 2000 words. The same kind of experiments were performed on Census data set. In this case the 8-dimensional data set described in 3.9.2 was projected on the first i of its dimensions ($i = 5..7$). In Fig. 3.17(c) and (d) results obtained on samples of queries having selectivity 0.1% and 1% (respectively) are depicted.

Both WAVE1 and WAVE2 were not considered on synthetic data, as the adopted prototype does not work on data sets so large in volume. Error rates

for WAVE1 are not reported in the diagram in Fig. 3.17(c) as they were out of scale. Diagrams in Fig. 3.17 show that, for both synthetic and real-life data, error rates of every technique tend to increase as dimensionality increases, but *GHBH* accuracy gets worse very slightly and outperforms all the other techniques at all considered dimensionalities.

3.9.7 Query estimation times

In this section experiments comparing the efficiency of estimating queries on histograms adopting the *FBH* and the *GHBH* representation models are presented: the estimation time is studied versus *a)* the amount of storage space used to represent the histogram; *b)* the volume of queries; *c)* the data dimensionality.

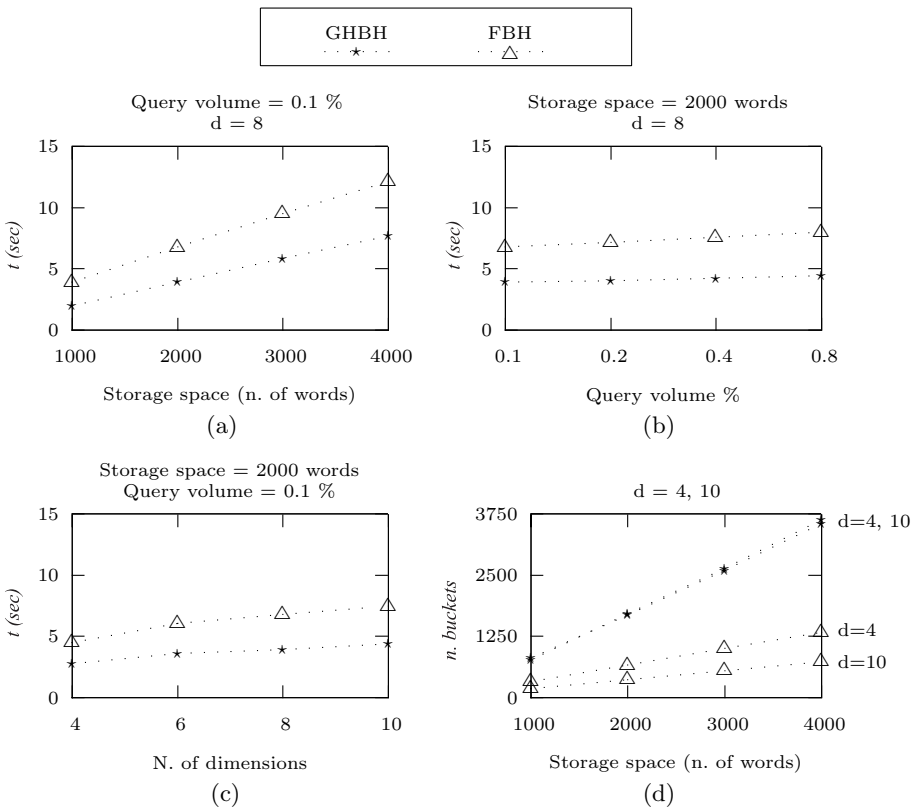


Fig. 3.18. Comparing query evaluation times (a-c) and number of buckets on synthetic data

Diagrams in Fig. 3.18 were obtained by considering the same synthetic data sets $D^4, \ldots, D^{10}$ used in the experiments of Fig. 3.17(a,b). In particular

diagram 3.18(a) depicts query estimation times versus the histogram size for a bulk of 10000 queries issued on histograms of type *FBH* and *GHBH* constructed on D^8 (all the queries of the bulk having volume that is 0.1% of the volume of the whole data domain). As expected for both *FBH* and *GHBH* query estimation times increase as the storage space used for representing histograms increases, since the amount of data that must be accessed to estimate queries gets larger. Diagram 3.18(b) refers to the same data distribution as diagram 3.18(a) and depicts query estimation times versus query volumes, for histograms consuming 2000 words (bulks of queries each consisting of 10 000 queries of the same volume were considered).

Diagram 3.18(c) depicts query execution times versus data dimensionality. In this experiment both the amount of storage space and the query volume (in terms of percentage of the data domain volume) are kept constant. This diagram shows that the increase in dimensionality slightly affects the query estimation times.

Overall, diagrams (a-c) show that estimating queries on histograms adopting the compression-based representation model of *GHBH* is faster than the case of histograms of type *FBH* consuming the same amount of storage space. This behavior turns out to be independent on the invested amount of storage space, the query volume, and the data dimensionality.

The relevance of this result is strengthened by considering that a *GHBH* consists of a much larger number of buckets than an *FBH* built within the same storage space bound. Diagram 3.18(d) depicts the number of buckets obtained by the histograms of type *FBH* and *GHBH* constructed on the 4-dimensional and the 10-dimensional data distributions of diagram 3.18(c) (that is, the data distributions having the lowest and the highest dimensionality, respectively). In the examined cases, adopting the *GHBH* representation model yields a number of buckets which ranges from 2.5 times to 5 times the number of buckets of an *FBH* (in the case of 4-dimensional data and 10-dimensional data, respectively). Observe that the amount of buckets of a *GHBH* is almost unaffected by data dimensionality, whereas the number of buckets of an *FBH* decreases as dimensionality increases. This is in agree with the formulas reported in Table 1, as it can be easily observed that both β^{min}_{GHBH} and β^{max}_{GHBH} are weakly sensitive to d, whereas β_{FBH} is inversely proportional to ξ.

Analogous results were obtained on real-life data distributions. In Fig. 3.19 results obtained by performing the same kind of experiments on Census datasets used in Fig. 3.17(c,d) are reported. Also in this context, *GHBH* representation allows always a more efficient query evaluation w.r.t. *FBH* one, for various values of storage space (Fig. 3.19(a)), query volume (Fig. 3.19(b)) and number of dimensions (Fig. 3.19(c)), and yields a larger number of buckets (Fig. 3.19(d)).

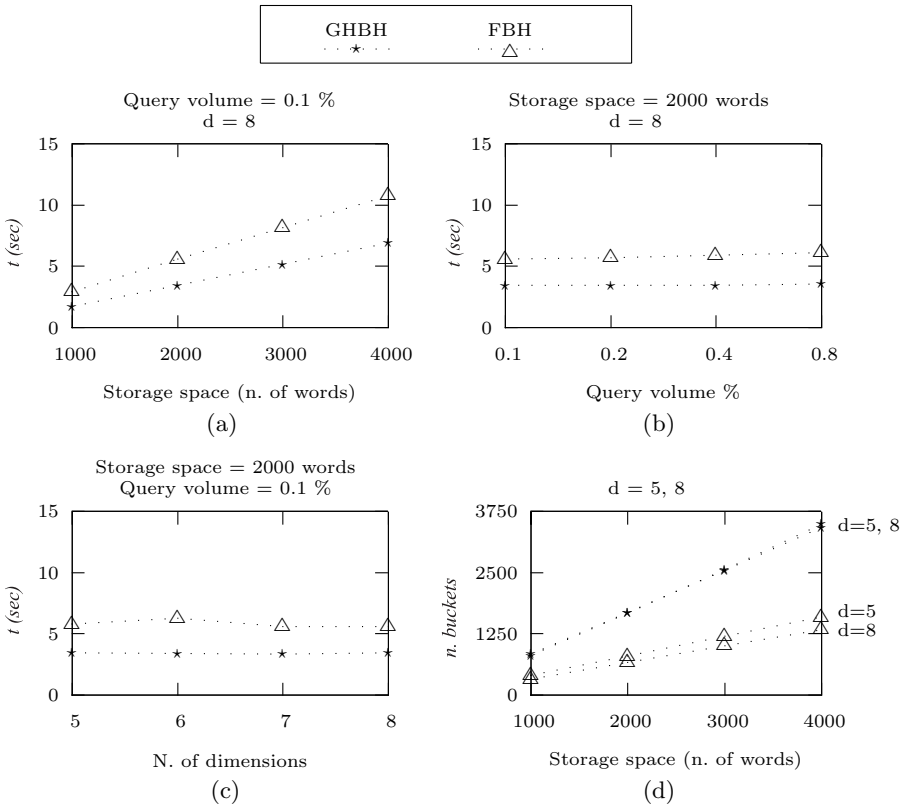


Fig. 3.19. Comparing query evaluation times (a-c) and number of buckets on Census

3.9.8 Histogram construction times

In this section is is shown how the execution times of Greedy Algorithms constructing a *GHBH* depend on several parameters, such as the storage space (i.e. number of buckets), the density, the volume, the dimensionality of D, and the grid degree. In particular, the execution times are presented when either the sparse data model, or the non-sparse one, or pre-computation is adopted.

Diagrams in Fig. 3.20 have been obtained for greedy algorithms adopting the *Max-Var/Max-Red* constructing an 8-*GHBH*.

Experimental results reported in Fig. 3.20 are basically consistent with the complexity bounds of Table 3.4, and can be summarized as follows:

- when the sparse model is used, the execution time of Greedy Algorithm is sensitive only on the number of non-null values in D (it grows linearly with N - see Fig. 3.20(a)), but is almost independent on either the data domain volume - Fig. 3.20(b) - and the dimensionality - Fig. 3.20(d);
- otherwise (if either the non-sparse data model is adopted or pre-computation is performed) the execution time of Greedy Algorithm is unaffected by an

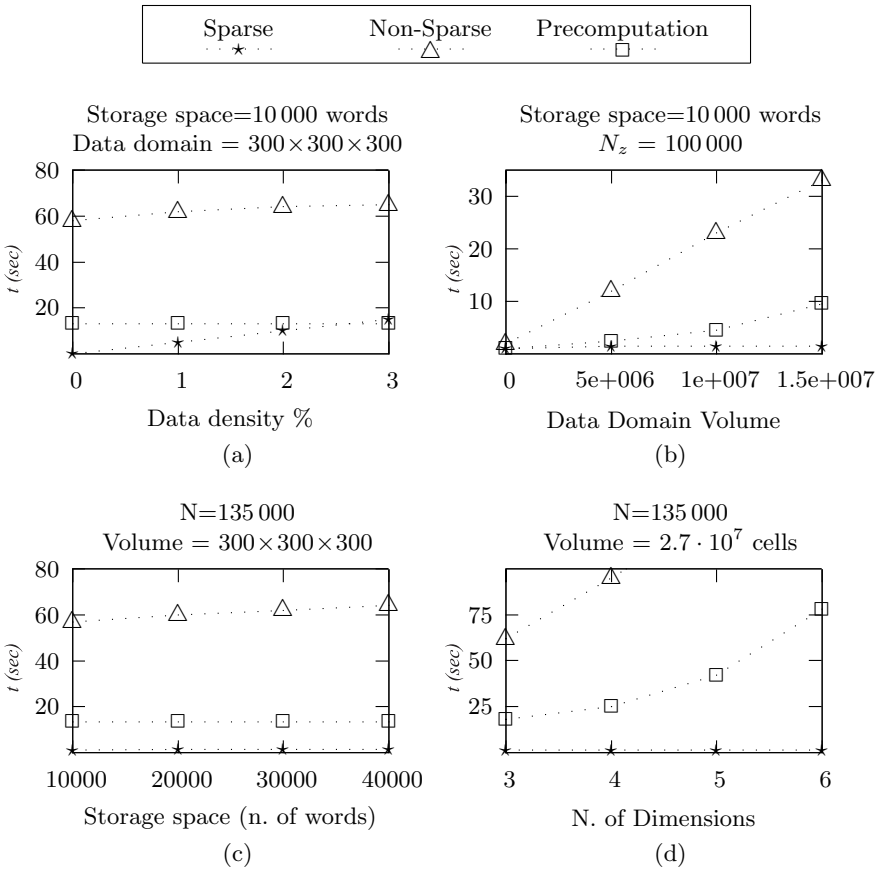


Fig. 3.20. Comparing efficiency of algorithms

increase in N, but it worsens dramatically as either d or n^d increases; in particular, the algorithm using pre-computation is faster than the one adopting the non-sparse model without pre-computation;

- when the sparse data model is used, if data density is smaller than a threshold, Greedy Algorithm is faster w.r.t. the case of non-sparse model or pre-computation. For instance, in the case depicted in Fig. 3.20(a), if data density is smaller that $\rho^* = 3\%$ the adoption of the sparse data model provides better performances than the use of pre-computation. Indeed, the exact value of the data density ρ^* where the execution times of the different approaches (based on either the sparse data model or the use of pre-computation) are about the same depends on a lot of parameters. In particular, it is worth noting that as the dimensionality increases, algorithms performing pre-computation or adopting the non-sparse data model slow down dramatically (as shown in Fig. 3.20(d)) so that the data density threshold gets a much larger value.

However, in practical scenarios, especially for data distributions with high-dimensionality, data density is so small that algorithms based on the sparse data model do perform much better than the others.

Fig. 3.20(c) shows that execution times of greedy algorithms is not very sensitive w.r.t. the size of the available storage space. This can be explained as follows:

- in the case that pre-computation is performed, the cost of the pre-computation step dominates the construction of the histogram. As explained in Section 3.7.3, the pre-computation step makes *Evaluate* more efficient to be performed, so that even if the number of buckets to be built increases, the computational overhead needed to compute them is almost negligible (i.e. the number of invocations of *Evaluate* increases, but each invocation is fast to be accomplished);
- in the other cases, the largest portion of execution times is devoted to the computation of the "first" steps, which involve very large buckets. For instance, at the first step of the greedy algorithms, the function *Evaluate* has to scan all values of D, that is either n^d values (non-sparse data model) or N values (sparse data model) must be accessed. As the construction process goes on, the buckets to be processed become smaller and smaller, so that the cost of performing further splits is almost negligible.

4

Clustering-based Histograms

In the previous chapter an effective technique for constructing histograms based on binary hierarchical partitions has been proposed. The technique, namely *GHBH*, has been shown to outperform the other histogram-based techniques and also some wavelet-based techniques to which it has been compared. The superiority of the techniques, besides to the very efficient representation models that it exploits, is due to the greedy criterion called *Max-Var/Max-Red*, which is capable of locating dense regions (see Fig. 3.11), task in which other greedy criteria, adopted by state-of-the-art techniques such as *MHIST* and *Min-Skew*, fail. If the greedy criterion guiding the partitioning of data domain is not suitable for distinguishing dense from sparse regions, the partitioning will yield buckets with high variance, that do not enable to estimate accurately range queries on the basis of the uniform distribution assumption. However, even though *Max-Var/Max-Red* does not suffer of this problem, there is an intrinsic limitation in all the techniques which base the data summarization on the construction of partitions by means of binary splits applied in a top-down fashion. In fact, when the data domain is very large, which is more and more likely as dimensionality of data increases, a top-down splitting process of the data domain, within a significantly bounded number of splits, it is unlikely to "reach" the dense regions, in order to isolate them. Thus, the possibility of summarizing together dense and sparse regions persists. For instance, consider two data distributions D^2 (of size n^2) and D^{10} (of size n^{10}), where the same number of data points are distributed, respectively, on a two-dimensional and ten-dimensional domain. If the same number of buckets is used to partition D^2 and D^{10}, buckets of D^{10} are likely to be much larger in volume than those of D^2. Therefore, the aggregate information associated to buckets of D^{10} is less localized than buckets of D^2 (as the aggregate value associated to each bucket is spread onto a larger volume), thus providing a poorer description of the actual data distribution.

In order to give an idea about the problems deriving from summarizing together dense and sparse regions, consider the bucket shown in Fig. 4.1. As the values inside the bucket are summarized by means of their sum, estimat-

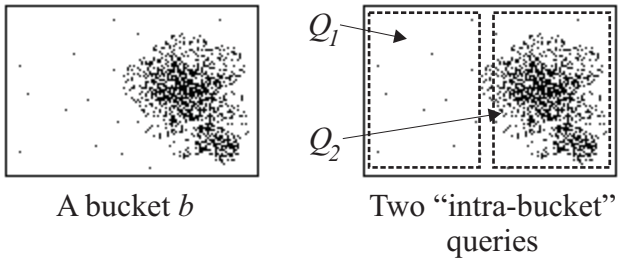


Fig. 4.1. Queries involving a inhomogeneous bucket

ing either Q_1 and Q_2 by adopting the uniform distribution assumption yields a high error rate, since the total sum is assumed to be homogeneously distributed inside b. In fact, this assumption is far from being true: most of the sum of b is concentrated in the dense cluster on the right-hand side of b.

Improving the ability of distinguishing dense regions can result in more accurate partitions, as this prevents buckets like that of Fig. 4.1 from being constructed. The problem of searching homogeneous regions is very close to the *data clustering* problem, that is the problem of grouping database objects into a set of meaningful classes. This issue has been widely studied in the data mining context, and several algorithms accomplishing data clustering have been proposed. Briefly, the clustering techniques can be classified in *partitional*, *hierarchical*, and *locality-based*. The partitional techniques the data point partitioning into clusters aims at optimizing an objective function, such as the distance between points. Each cluster is represented by either the mean of its points (k-means [93]), or by one of its points chosen as representative (k-medoid [80]). CLARANS [105] is an extension of traditional k-medoid algorithms yielding higher accuracy in locating clusters (even it could converge to a local optimum), but it is not well-suited for large databases as it may require multiple scans of the data points. Hierarchical clustering techniques construct a hierarchy of clusters by adopting either a top-down strategy (divisive hierarchical techniques) or a bottom-up one (agglomerative hierarchical techniques). BIRCH [137] and CURE [54] are examples of hierarchical algorithms. The former first populates a special data structure, namely *CF Tree*, where summary information of sub-clusters of object is stored, and then it runs an agglomerative algorithm on the previously generated sub-clusters. It is known to be unsuitable for distributions consisting of arbitrary shaped clusters or cluster having different sizes. On the contrary, CURE succeeds in identifying clusters having complex shapes and different sizes, and outperforms BIRCH on large databases. It uses a combination of random sampling and partitioning, where the clusters are characterized by a set of representative points, unlike BIRCH which uses a single centroid. Locality-based techniques group points according to some local property. The most representative technique in this class is DBSCAN [38], which is a *density-based* technique. A cluster is built

progressively by starting from a *core-point* (i.e., a point having a dense neighborhood), inserting all of its neighbors into the cluster, and then expanding the cluster from all core-points inserted at the previous step. OPTICS [6] is an extension of DBSCAN producing an augmented ordering of points representing its density-based clustering structure. A detailed survey of the existing clustering techniques can be found in [81].

In the following, a new clustering-based histogram construction technique, called *CHIST*, is proposed. The proposed technique enhances the histogram construction by adopting a clustering technique for locating dense regions, thus overcoming the problem of summarizing dense and sparse regions together. Then, this technique is extended to the case that data to be summarized are dynamic. In this scenario, re-executing the clustering of data at each data update is not feasible, due to the inefficiency of this task. Thus, a strategy for exploiting an incremental clustering approach (where the clustering is updated at each bulk of updates without reprocessing the whole data) is adopted in order to efficiently propagate data updates to the histogram. Finally, an experimental analysis of the technique performances is presented, both in terms of accuracy to range query estimates and histogram updating efficiency.

4.1 *CHIST*: Clustering-based Histogram

The proposed technique works in three steps. At the first step clusters of data and outliers (i.e. points which do not belong to any cluster) are located. At the second step, these clusters and the set of outliers are treated as distinct layers, and each layer is summarized by partitioning it according to a grid-based paradigm. At the last step the histogram is constructed by "assembling" all the buckets obtained at the previous step.

The three phases of thus approach are described in detail in the following sections. The description of the algorithm is provided by assuming a d-dimensional data distribution D, as described in Section 1.5. The amount of available storage space for the representation of the histogram will be denoted as B.

4.1.1 Step I: clustering data

The adopted algorithm for grouping data into dense dense clusters is DBSCAN [38], which is the most representative of density-based clustering algorithms. Even though the presented approach can be viewed as orthogonal to any clustering technique, DBSCAN has been chosen as it seems to be the most suitable for supporting the histogram construction. In fact, due to its density-based bottom-up clustering strategy, it can not fail in locating dense regions.

The idea underlying DBSCAN is that points belonging to a dense cluster (except those points lying on the border of the cluster) have a dense neighborhood. A point p is said to have a dense neighborhood if there are at least

MinPts distinct points whose distance from p is less than *Eps* (both *Eps* and *MinPts* are parameters crucial for the definition of clusters). Points with a dense neighborhood are said to be *core points*. DBSCAN scans input data searching for core points. Once a core point p is found, a new cluster C is created, and both p and all of its neighbors are grouped into C. Then C is recursively expanded by including the neighbors of all core points put in C at the last step. When C cannot be further expanded, DBSCAN searches for other core points to start new clusters, until no more core points can be found. At the end of the clustering, points which do not belong to any cluster are classified as *outliers*. Fig. 4.2 depicts an example of clustering obtained by DBSCAN.

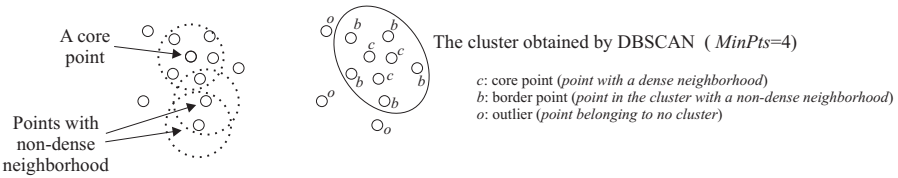


Fig. 4.2. Running DBSCAN on a set of points in a two-dimensional space

4.1.2 Step II: summarizing data into buckets

At this step the input data distribution is viewed as a superimposition of layers. Each layer is either a cluster or the set of outliers. In the following L_0 will denote the layer consisting of outliers, and $L_1, \ldots, L_c$ will denote the layers corresponding to dense clusters. L_0 will be said to be the *outlier layer*, whereas $L_1, \ldots, L_c$ will be said to be *cluster layers*. Each layer is represented by means of its *MBR* (i.e. the minimal hyper-rectangle containing all non-null points of the layer.

The different layers are summarized separately by partitioning their *MBR*s into buckets. This aims at preventing the construction of buckets where dense and sparse regions are put together, which, as explained before (see Fig. 4.1), can yield poor accuracy. The summary of the whole data distribution will be the superimposition of the summaries of all layers.

The summarization of layers is accomplished by a multi-step algorithm which, at each step, summarizes a single layer by partitioning it according to a grid and storing, for each bucket defined by this grid, both its *MBR* and the sum of its values (obviously, the cells of this grid which do not contain any data point result in empty *MBR*s which are not stored). The *MBR*s of buckets obtained from the summarization of cluster layers will be said to be *c-buckets*, whereas the *MBR*s of the buckets constructed by partitioning L_0 will be said to be *o-buckets*.

Indeed, layer L_0 is processed after the summarization of all the cluster layers. In particular, before summarizing the outlier layer, all outliers are scanned in order to locate those lying onto the range of some c-bucket. Each outlier o which lies onto some c-bucket is removed from L_0 and "added" to one c-bucket whose range contains the coordinates of o[1]. This enables the c-buckets to be viewed as "holes" of L_0, in the sense that, after performing this task, there are no points lying onto the range of some c-bucket which belong to L_0. As it will be clear in the following, this will be exploited in the physical representation of the histogram to improve its accuracy.

The overall available storage space to summarized data, must be partitioned among the layers in order to summarize each of them. Let B_i be the amount of memory which is left from the $i-1$ previous summarization steps (at the first step, B_1 coincides with the initial amount of storage space B). The portion of B_i which is invested to summarize L_i is denoted as $B(L_i)$ and is computed by comparing the need of being partitioned of L_i with all remaining layers $L_{i+1}, \ldots, L_c, L_0$. The need of being partitioned of a layer L is estimated by computing its *SSE* (denoted as $SSE(L)$), thus $B(L_i) = B_i \cdot \frac{SSE(L_i)}{SSE(L_0)+\sum_{j=i}^{c} SSE(L_j)}$.

L_i is partitioned according to a regular grid and, for each cell of the grid containing at least one point, the coordinates of its *MBR* and the sum of the values occurring in it are stored. The grid on a layer L_i is constructed as follows.

Denoting the amount of storage space needed to store a bucket by W[2], the number of buckets produced by the grid on L_i can be no more than $nb = \lfloor \frac{B(L_i)}{W} \rfloor$. Thus, if t_j is the number of divisions of the grid along the j-th dimension of L_i, it should be $\prod_{j=1}^{d} t_j = nb$.

A grid could be easily constructed by means of an equi-partitioning strategy, i.e. by partitioning all edges of the *MBR* into the same number of portions. That is, it could be possible to choose $t_1 = t_2 = \cdots = t_d = \sqrt[d]{nb}$. Indeed, this choice is likely to result in a grid whose cells are hyper-rectangles having edges with large differences in length, unless the *MBR* of the layer has edges with about the same size. A partition with buckets having large differences in length (see Fig. 3.11) is likely to provide poor accuracy in query estimates, as it is less effective in preserving the locality of information. This is due to the fact that summarizing the points inside a bucket into a single value means spreading each point onto the whole range of the data domain delimited by the bucket itself. Therefore, each point can give a contribution to cells of the domain whose distance is bounded by the maximum diagonal d_{max} of the bucket. Observe that, each possible partitioning of the *MBR* of L_i into a

[1] If more than one c-bucket contains o, one of these c-buckets is randomly selected to incorporate o. Adding an outlier o to a c-bucket b means removing o from L_0 and adding the value of o to $sum(b)$.

[2] $2 \cdot d$ 32-bit words will be adopted for storing bucket boundaries, and one 32-bit word for storing the sum-aggregate

given number of cells nb by means of a grid yields cells having the same volume (i.e. $vol(L_i)/nb$)[3]. The value of d_{max}, for buckets having the same volume, is minimized when the length of edges of the bucket is the same. For instance, consider the two buckets b' of size $10 \times 10 \times 10$, and b" of size $100 \times 5 \times 2$. These two buckets have the same volume, but very different values of the maximal diagonals. In fact, for b' it results $d'_{max} = \sqrt{10^2 + 10^2 + 10^2} \cong 17.1$ and for b" $d"_{max} = \sqrt{100^2 + 5^2 + 2^2} \cong 100.1$. Therefore, after performing summarization, the information of a point at a vertex of the range of b' is kept more localized than that of a point at a vertex of the range of b". Moreover, it is quite intuitive that a bucket is more and more likely to overlap a given range query as its maximal diagonal gets larger. Then, range queries estimations are likely to involve a larger and larger number of buckets as the partition of data domain contains buckets having larger and larger maximal diagonals. Thus, the estimation is likely to be less accurate, as it is affected by the errors due to a larger number of approximate contributions.

In order to prevent large differences in values between the edges of buckets, each edge of the *MBR* of the layer to be summarized is partitioned into a number of portions which is proportional to the length of the edge itself. Let w_j be the length of the edge along the j-th dimension. Choosing $t_j = w_j \cdot \sqrt[d]{\frac{nb}{vol(L_i)}}$ guarantees both that $\prod_{j=1}^{d} t_j = nb$ and that the grid degree along each dimension is chosen by weighting the corresponding edge size. Indeed, this formula can result in non-integer value coefficients $t_1, \ldots, t_d$.

Therefore $t_1, \ldots, t_d$ must be rounded, with the constraint that their product cannot be larger than $nb(L)$. Rounding each t_j to the nearest integer $\lfloor t_j + 0.5 \rfloor$ does not guarantee that $\prod_{j=1}^{d} t_j = nb$. On the other hand, truncating each t_j to $\lfloor t_j \rfloor$ guarantees $\prod_{j=1}^{d} t_j = nb$, but this may result in a grid with much fewer cells than n_b. This is due to the fact that independent approximations of the values t_j does not enable to control the final approximate value of their product.

In order to prevent this problem, the degrees of the grid along each dimension are computed progressively, starting from t_1 to t_d, according to the following scheme:

$$t'_1 = max\{\lfloor t_1 \rfloor, 1\};\ t'_2 = max\left\{\left\lfloor \frac{t_1 \cdot t_2}{t'_1} \right\rfloor, 1\right\};\ \ldots\ t'_d = max\left\{\left\lfloor \frac{\prod_{j=1}^{d} t_j}{\prod_{j=1}^{d-1} t'_j} \right\rfloor, 1\right\}.$$

That is, the value of each t_j is approximated to t'_j by taking into account the approximations already performed at the $j-1$ previous steps[4].
For instance, consider a 4-dimensional layer L_i whose *MBR* has size $30 \times 60 \times 120 \times 240$. Let $Vol = 30 \cdot 60 \cdot 120 \cdot 240$ be the volume of the *MBR* of L_i, $nb = 100$ be the number of buckets which can be constructed on L_i, and

[3] In the following, $vol(L_i)$ will denote the volume of the *MBR* of the layer L_i

[4] A better approximation could be obtained by rounding t'_j to the nearest integer, but this would not guarantee to obtain a number of cells not larger than nb

$K = \sqrt[d]{\frac{nb}{Vol}}$. It results $t_1 = 30 \cdot K \cong 1.1$, $t_2 = 60 \cdot K \cong 2.2$, $t_3 = 120 \cdot K \cong 4.5$, and $t_4 = 240 \cdot K \cong 8.9$. Truncating t_j to $\lfloor t_j \rfloor$ would results in a number of bucket equal to $1 \cdot 2 \cdot 4 \cdot 8 = 64$. Instead, by progressively approximating t_j, it results $t'_1 = \lfloor t_1 \rfloor = 1$, $t'_2 = \left\lfloor \frac{t_1 \cdot t_2}{1} \right\rfloor = 2$, $t'_3 = \left\lfloor \frac{t_1 \cdot t_2 \cdot t_3}{1 \cdot 2} \right\rfloor = 5$, and $t'_4 = \left\lfloor \frac{t_1 \cdot t_2 \cdot t_3 \cdot t_4}{1 \cdot 2 \cdot 5} \right\rfloor = 10$.

Although in the case shown above $t'_1 \cdot t'_2 \cdot t'_3 \cdot t'_4 = nb$, it can happen that constructing a grid using this strategy results in nb' buckets, with nb' strictly less than nb. This can be due either to numerical approximation (the value of $\prod t'_j$ can be less than nb), or to the fact that some cells of the grid can correspond to null regions of the data domain, so that they are not stored explicitly. Therefore, after a layer L_i is summarized, the residual amount of storage space which will be available at step $i+1$ is given by $B_{i+1} = B_i - nb' \cdot W$ (that is, if some space which was assigned to the summarization of L_i has not been consumed, it is reinvested at the following steps).

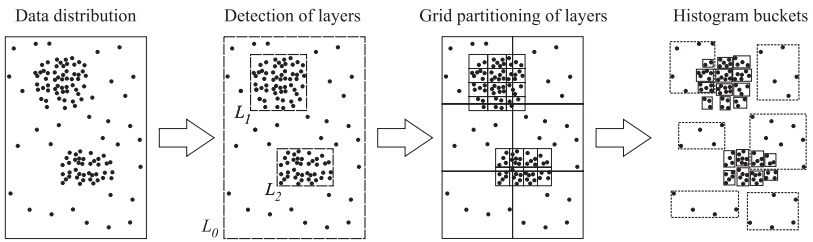


Fig. 4.3. Detection of layers, data partitioning, and bucket definition

Fig. 4.3 shows the execution of Step I and Step II on a two-dimensional data distribution.

Remark. Observe that, as the amount of storage space invested to partition a layer L_i depends on its homogeneity (estimated by means of its *SSE*), deciding on partitions is not merely based on the proportions between the length of the dimensions, but strictly depends on data uniformity. Thus, the less homogeneous the data inside L, the finer the grid and the larger the number of buckets associated to L. Moreover adopting the grid-based scheme allows us to partition L in linear time (each data point inside L is accessed once and summarized in the cell of the grid where it lies into): this feature will be particularly well-suited for the incremental approach where an efficient partitioning strategy is needed to propagate data changes to the histogram (see Section 4.2).

4.1.3 Step III: representation of the histogram

The strategy adopted to partition layers can yield overlapping buckets. In particular, as buckets aggregating points of L_0 (the layer consisting of outliers)

are likely to be larger than buckets describing clusters (outliers are generally spread over the whole data domain), several c-buckets $b_1, \ldots, b_k$ can lie onto the range of an o-bucket b. In this scenario $b_1, \ldots, b_k$ can be viewed as "holes" of b, as the aggregate information associated to b does not refer to points contained inside $b_1, \ldots, b_k$. This observation can be exploited to make query estimation more accurate. In the following, given an o-bucket b, the set of c-buckets completely contained into b will be denoted as $Holes(b)$.

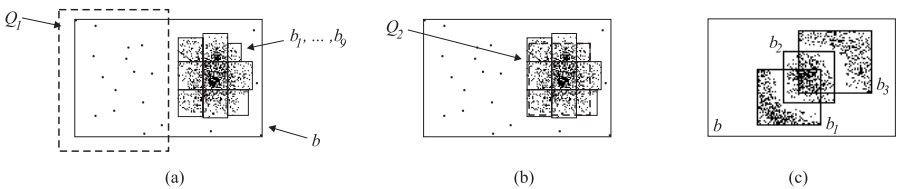


Fig. 4.4. O-buckets with holes

Consider the scenario depicted in Fig. 4.4(a), where the query Q_1 intersects one half of the range associated to the bucket b. Adopting linear interpolation to estimate Q_1 returns, it results $\widetilde{Q}_1 = \frac{vol(Q_1 \cap b)}{vol(b)} \cdot sum(b)$, where $Q_1 \cap b$ refers to the intersection between the query range and the range of b. Indeed, points belonging to the ranges of $b_1, \ldots, b_9$ give no contribution to the value of $sum(b)$. Therefore, a more precise estimate for Q_1 is: $\widetilde{Q}_1 = \frac{vol(Q_1 \cap b)}{vol(b) - vol(b_1, \ldots, b_9)} \cdot sum(b)$, where $vol(b_1, \ldots, b_9)$ denotes the volume of the range underlying the buckets $b_1, \ldots, b_9$. Likewise, the bucket b should give no contribution to the estimate of the query Q_2 in Fig. 4.4(b), which lies completely on the range underlying the buckets $b_1, \ldots, b_9$.

In the following the number of cells of an o-bucket b which are not contained in any hole of b will be said to be the *actual volume* of b. In the case depicted in Fig. 4.4(a) evaluating the actual volume of b can be accomplished efficiently, as $b_1, \ldots, b_9$ do not overlap. Indeed also c-buckets inside an o-bucket b can intersect one another [5]. For instance, in Fig. 4.4(c) the three buckets b_1, b_2, b_3 inside b overlap. In this case computing the actual volume of b requires $vol(b_1)$, $vol(b_2)$, $vol(b_3)$, $vol(b_1 \cap b_2)$, $vol(b_2 \cap b_3)$ and $vol(b_1 \cap b_2 \cap b_3)$ to be computed. This computation becomes more and more complex when more buckets intersect in the same region: it is necessary to compute the volumes of all the intersections between pairs of holes, triplets holes, and so on. Obviously, this slows down query estimations. Due to this reason, it is preferable to estimate the actual volume of an o-bucket b involved in a query instead of evaluating its exact value: To this end, only a maximal subset of $Holes(b)$ (denoted as $NOHoles(b)$) is considered, consisting of non-overlapping c-buckets

[5] Although no pair of clusters C_1, C_2 can overlap (otherwise C_1, C_2 would be a unique cluster), *MBR*s of clusters can overlap (see Fig. 4.4(c)). Thus partitioning overlapping *MBR*s can result in overlapping c-buckets.

completely contained inside the b, thus avoiding intersections between holes to be computed. For instance, in the case depicted in Fig. 4.4(c) it is possible to choose $NOHoles(b) = \{b_3\}$, thus the actual volume of b can be estimated as $vol(b) - vol(b_3)$. However, from several experiments on real-life data it turned out that intersections between c-buckets are unlikely to occur.

The adopted representation model partitions buckets into two levels. The buckets at the second level are those belonging to $NOHoles(b)$ for some b. The first level consists of all the other buckets.

The physical representation model can be exploited to evaluate query answers efficiently. The answer to a given query Q is computed as follows:
1) the first-level buckets whose range overlap the query range are located;
2) for each of these buckets b, all of its hole-buckets $b_1, \ldots, b_k$ are accessed and those involved in the query are located. Then, the contribution of both b and its holes to the query estimate are evaluated. In particular for each hole b_i the contribution to the estimate of Q is given by: $\frac{vol(b_i \cap Q)}{vol(b_i)} \cdot sum(b_i)$, whereas the contribution of b is $\frac{vol(b \cap Q) - \sum_{i=1}^{k} vol(b_i \cap Q)}{vol(b) - \sum_{i=1}^{k} vol(b_i)} \cdot sum(b)$.

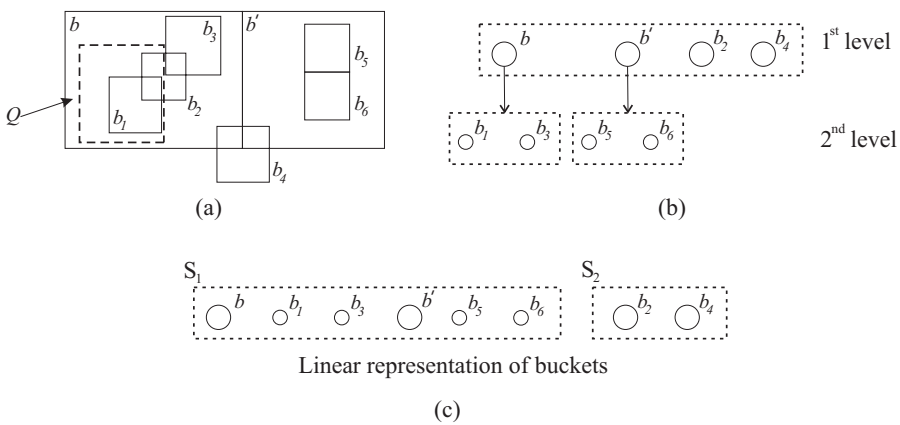


Fig. 4.5. Nested representation of buckets

For instance, the estimate of the query Q shown in Fig. 4.5(a), according to the representation shown in Fig. 4.5(b), is given by: $\widetilde{Q} = \frac{vol(b \cap Q) - vol(b_1 \cap Q)}{vol(b) - vol(b_1) - vol(b_3)} \cdot sum(b) + sum(b_1) + \frac{vol(b_2 \cap Q)}{vol(b_2)} \cdot sum(b_2)$.

Therefore the adopted representation scheme enables range query answers to be estimated by accessing each bucket at most once.

Observe that representing some c-buckets as holes of o-buckets introduces no spatial overhead on the representation of o-buckets. That is, the two-levels organization of buckets can be linearized by representing buckets into two distinct sequences S_1, S_2. In particular, S_1 contains all o-buckets and their non-overlapping holes: each o-bucket b is followed by the representation of

c-buckets in $NOHoles(b)$ (see Fig. 4.5(c)). Thus, locating non-overlapping holes of an o-bucket b at position i in this sequence can be accomplished by scanning the positions of the sequence following i, till either the end of the sequence or an o-bucket having an empty intersection with b is reached (for instance, the holes b_1, b_3 of b occur in the sequence between b and b'). Sequence S_2 contains all c-buckets which do not belong to any $NOHoles(b)$ for any o-bucket b.

This is why c-buckets which partially overlap o-buckets are not considered as holes. For instance, in the case of Fig. 4.5(a), bucket b_4 is not taken into account to estimate the actual volumes of b and b'. Otherwise, a reference to b_4 should be inserted into both the representations of b and b' a reference to b_4 (which cannot be accomplished by a sequential physical representation of the histogram), and moreover bucket b_4 should be accessed more than once to estimate queries involving b and b'.

The idea of representing some buckets as holes of other buckets was introduced in [11]. In that work, holes of buckets are determined by query results feedback; if two holes overlap, one of them is shrunk; the aggregate information of the portion of the hole which has been cut off is estimated by linear interpolation and spread onto the overlying bucket. This approach is not suitable in this context as here holes are dense buckets, so that spreading their aggregate information onto an o-bucket (which is often much larger and sparser) may result in a severe loss of accuracy.

The algorithm implementing steps I, II, III is depicted in Fig. 4.6. Therein

- *size* takes as argument a set of buckets and returns their storage space consumption.

- *GridPartition* takes as argument the *MBR* of a layer and a storage space bound; it creates a grid on the *MBR* of layer such that the number of cells can be stored within the specified storage space bound; then it returns all the non-null buckets defined by the grid.

- *Distribute* takes as argument the layer of outliers $L[0]$ and the set of c-buckets. For each outlier o lying onto the range of some c-bucket, this function removes o from $L[0]$ and adds o to a c-bucket b whose range contains o (that is, the value of o is added to the aggregate $sum(b)$). If o lies onto the range of more than one c-bucket, then one of these c-buckets b is randomly chosen and o is added to it.

- *NOHoles* takes as argument an o-bucket b and returns a maximal subset of non-overlapping c-buckets whose range is completely contained inside that of b.

Remark. It is worth noting that the idea of possibly representing a c-bucket as a hole of an o-bucket cannot be extended to the case of pairs of c-buckets b', b'' such that b' is completely contained into b''. Fig. 4.7 shows an example

INPUT D: a multi-dimensional data distribution;
B: available amount of storage space for representing the histogram;

OUTPUT H: a clustering-based histogram on D within B consisting of two sequences S_1 and S_2 of buckets (see Fig. 4.5 c);

begin
L := DBSCAN(D);
$SSE_{tot} = \sum_{i=0}^{L.size\text{-}1} SSE(L[i])$;
C-Buckets= $\emptyset$;
for ($i \in \{1..L.size - 1\}$) **do begin**
$B_{L_i} = \frac{SSE(L[i])}{SSE_{tot}} \cdot (B - size(\textit{C-Buckets}))$;
C-Buckets = *C-Buckets*$\cup$ *GridPartition*$(L[i], B_{L_i})$;
$SSE_{tot} = SSE_{tot} - SSE(L[i])$;
endfor;
Distribute$(L[0], \textit{C-Buckets})$;
O-Buckets = *GridPartition*$(L[0], B - size(\textit{C-Buckets}))$;
for ($b \in$ *O-Buckets*) **do begin**
NOH = *NOHoles*(b);
C-Buckets= *C-Buckets* - *NOH*;
$H.S_1 = H.S_1 \cup b \cup NOH$;
endfor;
$H.S_2$ = *C-Buckets*;
return H;
end;

Fig. 4.6. *CHIST* Algorithm

Fig. 4.7. Two clusters whose *MBR*s overlap

of two clusters C_1, C_2 whose *MBR*s overlap. Observe that after partitioning C_1 into b_1, b_2 and C_2 into b_3, b_4, the range of the bucket b_4 is completely contained into that of b_2, but b_4 cannot be considered as a hole of b_2, as there are points of both C_1 and C_2 laid into the range of b_4.

4.2 Incremental maintenance of *CHIST*

The computational complexity of the proposed clustering-based histogram construction is dominated by the cost of executing DBSCAN. DBSCAN runs in $\mathcal{O}(N \cdot \log N)$ if a multi-dimensional indexing technique is adopted to support the efficient location of neighbors. Indeed, its complexity degrades to $\mathcal{O}(N^2)$ on high-dimensional data sets, where no indexing technique is known to be efficient in searching the neighbors of data points.

	500	1 000	2 000	4 000
CHIST	361 + 2.1	361 + 4.2	361 + 8.5	361 + 19.2
GENHIST	18	30	71	149
GHBH	1.5	1.6	2.0	2.2

Table 4.1. Histogram construction times (seconds)

Table 4.1 shows the construction times, of *CHIST*, GENHIST and *GHBH* on a 8-dimensional data distribution containing 200 000 tuples for different storage space bounds (500, 1000, 2000, and 4000 words).

Construction times for *CHIST* are expressed as a sum of two contributions: the first one is the time needed to execute DBSCAN, the second one refers to Step II and Step III of the algorithm. It is clear that the clustering step overwhelms the rest of the algorithm execution, and it make the *CHIST* algorithm significantly slower than the other ones. This could not be a crucial drawback: the summarization of data is an off-line task which is usually performed on historical data, so that in practice it is executed only once.

However, this is likely to limit the applicability of *CHIST* to static data sets, such as non-evolving historical data, where the construction of the histogram is performed only once.

Otherwise, in the case of evolving data sets, any change of the data would require the re-execution of the algorithm from scratch. In order to reduce the overhead due to this task, the re-computation of the histogram could be scheduled to be run periodically (e.g. every night) or when the system managing data is unloaded. But this could make the histogram out-of-date, thus compromising the estimation accuracy, especially in the case that data change much more frequently w.r.t. histogram re-computation. Observe that the adoption of a clustering technique more efficient than DBSCAN does not suffice to solve this problem, as no technique is known to accomplish the (from-scratch) clustering fast enough: any technique would require the overall data to be re-scanned at least once, which makes frequent histogram re-computations unfeasible in practice.

A possible solution to this problem is to adopt an incremental clustering technique to propagate efficiently data changes to the clusterization. An

incremental clustering algorithm computes the clusterization of the updated data starting from the preexisting clusterization and modifying it according to the data updates, aiming at reducing as much as possible the amount of data to be accessed. However, replacing the non-incremental clustering step with an incremental one at Step I may not suffice to make the whole technique well-suited for reacting to frequent updates. In fact, Step II requires a linear scanning of data to compute the bucketization of all layers. In order to exploit the advantage of incremental clustering, Step II needs to be changed too, so that layers which are not affected by the data updates are not repartitioned, thus exploiting as much as possible the preexisting bucketization.

Motivated by these observations, in this section an incremental algorithm for maintaining the histogram up-to-date w.r.t. data changes is proposed. In more detail, the proposed strategy works in three steps, which will be described in the following sections:

I Incremental clustering;
II Storage space distribution among layers and partitioning;
III Rearrangement of buckets.

Throughout the following sections each point p of the data distribution is assumed to be marked with two labels[6] $Flag(p)$ and $Layer(p)$. The former has a boolean value, specifying whether p is an outlier or belongs to a cluster. $Layer(p)$ is the identifier of the layer where p is summarized: thus, if p is an outlier summarized in a o-bucket then $Layer(p) = 0$, else if p is a point summarized in a c-bucket obtained by partitioning the layer L_i then $Layer(p)$ is the identifier of L_i. Basically, the values of $Flag(p)$ and $Layer(p)$ describe the current composition of layers before executing a bulk of updates, and are changed accordingly to the data updates during steps I,II. In particular, during the execution of these steps, $Layer(p)$ can be also assigned -1, meaning that p has not been assigned to any layer yet.

4.2.1 Step I: incremental clustering

The task performed at this step consists in propagating data updates to the clusterization. There are several techniques in literature which accomplish this task in an incremental fashion, that is they compute the clusterization of updated data without re-executing the clustering algorithm from scratch on all the data. The incremental clustering technique adopted to extend *CHIST* to evolving data sets is *Incremental DBSCAN* [39]. According to this technique, data updates may have different effects on the clusterization, and thus on the corresponding layers. When a new point p is added to the data distribution, one of the following cases may occur:

[6] Apart from further labels possibly associated to the points by the adopted clustering algorithm

I1- *no new cluster is created, and no old cluster is affected*: this happens if p is an outlier; in this case, the layer of outliers must be augmented, whereas the other layers need no change; $Flag(p)$ is assigned 0 (meaning that p is classified as an outlier) and $Layer(p)$ is assigned -1 (meaning that p is an outlier which has not been summarized in any bucket yet);

I2- *a new cluster including p is created, and no old cluster is affected*: in this case, a new layer is created (corresponding to the new cluster), and the layer of outliers may need to be reduced (in the case that some preexisting outliers are absorbed into the new cluster). Layers corresponding to pre-existing clusters need no change. In this case, for each point p' included in the new cluster, $Flag(p')$ is assigned 1 and $Layer(p')$ is assigned the id of the new cluster.

I3- *no new cluster is created, and some old clusters are affected*: this can arise from one of the following cases:

- *p is adsorbed by exactly one of the preexisting clusters*: in this case, the layer of the involved cluster must be augmented; $Flag(p)$ is assigned 1 and $Layer(p)$ is assigned the id of the cluster adsorbing p;
- *p is adsorbed by two or more clusters, and these clusters are merged in a single one*: in this case, the layers of the merged clusters must be deleted, and a new layer corresponding to the new cluster must be created. For each point p' adsorbed by the new cluster $Flag(p')$ is assigned 1 and $Layer(p')$ is assigned the id of the new cluster.

Moreover, in both cases the layer of outliers must be reduced if some preexisting outliers are absorbed into a cluster together with p. For each of these points p', $Flag(p')$ changes from 0 to 1 and $Layer(p')$ is assigned the id of the adsorbing cluster.

Analogously, when a point p is deleted from the data distribution it can be one of the following cases:

D1-*no old cluster is affected*: this happens if p was an outlier; in this case, the layer of outliers must be reduced and no other layer need updates;

D2-*exactly one old cluster is affected*: this happens if p belonged to a cluster C. In particular, one of the following cases can occur:

a. *C is reduced*: this happens when after the removal of p some points of C become outliers; in this case the layer of C must be reduced and the outlier layer must be augmented. In particular, for each p' which is no more a cluster point, $Flag(p')$ is assigned 0 and $Layer(p')$ is assigned -1;

b. *C is deleted*: this happens when the removal of p results in making no point of C have a dense neighborhood, thus all points of C become outliers; in this case the layer corresponding to C is deleted and the layer of outliers must be augmented. For each point p' which belonged to C $Flag(p')$ is assigned 0 and $Layer(p')$ are assigned -1;

c. *C is split into two or more clusters*: this happens when, after the removal of p, two or more core points are no more density-reachable from

one another; thus they define distinct clusters. In this case the layer corresponding to C is split into two layers (i.e. the layer of C is deleted and two new layers are created). Moreover the layer of outliers may need to be augmented (in the case that some points belonging to C become outliers). For each point p' involved in the split, the values of $Flag(p')$ and $Layer(p')$ are changed consistently.

The above-reported list summarizes the operations performed on layers for a single update. Indeed an incremental clustering step consists of processing a bulk of updates, which is processed as a sequence of single updates. See [39] for further details and graphical examples on how inserting/deleting points can change clusterization.

The histogram maintenance is supported by an auxiliary (main-memory resident) data structure consisting of two sets $\mathcal{L}_{new}$ and $\mathcal{L}_{old}$, whose items are of the form
$\langle L, MBR(L), sum(L), sum^2(L), count(L), B(L)\rangle$, where L is a layer identifier and $B(L)$ denotes the amount of storage space which was invested to partition L during the construction of the old histogram (obviously $B(L) = 0$ if L is a newly detected layer). The aggregate data $sum(L)$, $sum^2(L)$ and $count(L)$, as well as $MBR(L)$, will be used at Step 2 to evaluate the *SSE* of L, whereas $B(L)$ will be used to decide whether old layers need to be repartitioned or not. Basically, at the end of the incremental clustering step, $\mathcal{L}_{new}$ and $\mathcal{L}_{old}$ contain the up-to-date clusterization (w.r.t. the processed bulk of insertions and deletions). In particular, $\mathcal{L}_{old}$ contains the list of the layers which existed before the bulk of updates and which have not been affected by the updates; on the contrary, $\mathcal{L}_{new}$ consists of the layers which were not in the preexisting clusterization. Neither $\mathcal{L}_{old}$ nor $\mathcal{L}_{new}$ contain any tuple corresponding to the layer of outliers: aggregate data of L_0 are stored separately from these lists.

At the beginning of the incremental clustering step, $\mathcal{L}_{new}$ is empty while $\mathcal{L}_{old}$ contains the list of the preexisting layer identifiers and their aggregate data (except from L_0). During the execution of the incremental clustering step, both $\mathcal{L}_{new}$ and $\mathcal{L}_{old}$ are maintained up-to-date as follows. Consider an update operation u (i.e. insertion or deletion) in the processed bulk of updates. Let $Affected(u)$ be the set of layers affected by u and $Created(u)$ the set of layers created after performing u. Basically, $Affected(u)$ contains layers in $\mathcal{L}_{old} \cup \mathcal{L}_{new}$ which need either augmentation or reduction or deletion, whereas $Created(u)$ contains layers which need to be created (i.e. layers in $Created(u)$ can result from either splitting clusters, merging clusters, or creating new clusters). For each layer L in $Created(u)$ the tuple $\langle L, MBR(L), sum(L), sum^2(L), count(L), 0\rangle$ is inserted into $\mathcal{L}_{new}$. For each layer L in $Affected(u)$ the following operations are performed. If L has to be deleted, then the corresponding tuple is removed from the list it belongs to (either $\mathcal{L}_{new}$ or $\mathcal{L}_{old}$). Otherwise, if L needs either augmentation or reduction, the attributes $MBR(L)$, $sum(L)$, $sum^2(L)$, $count(L)$ in the corresponding tuple are updated. Moreover, if L was in $\mathcal{L}_{old}$ the corresponding tuple is moved

to $\mathcal{L}_{new}$ (after assigning 0 to $B(L)$). Finally, for each outlier p which had been summarized into the buckets of some layer in $\mathit{Affected}(u)$ the value of $\mathit{Layer}(p)$ is changed to -1.

Therefore, at the end of the incremental clustering, every point p classified as a cluster point is assigned $\mathit{Flag}(p) = 1$ and $\mathit{Layer}(p) = id(L)$, where L is the layer corresponding to the cluster containing p. For each outlier p, $\mathit{Flag}(p)$ is assigned 0; as regards $\mathit{Layer}(p)$ one of the following cases can occur:

- $\mathit{Layer}(p) = i \geq 0$: this means that p is currently summarized into a bucket associated to the layer L_i;
- $\mathit{Layer}(p) = -1$: this means that p is not currently summarized into any bucket (this can be due to two reasons: either p is a newly created outlier, or p was an outlier summarized into a layer affected by the data update).

As will be shown in the following section, every outlier whose *Layer* value is -1 will be assigned to exactly one layer and summarized into one of its buckets. That is, if the outlier p happens to be summarized into an o-bucket, then $\mathit{Layer}(p)$ will be assigned 0, else if p happens to be adsorbed by a c-bucket b, then $\mathit{Layer}(p)$ will be assigned the id of the layer which b refers to.

Lists $\mathcal{L}_{new}$ and $\mathcal{L}_{old}$ will be used at Step II to detect layers which need to be partitioned.

4.2.2 Step II: storage space distribution among layers and partitioning

The incremental clustering step results in a new clusterization, where new layers may be added and some preexisting layers may be either deleted or modified w.r.t. the previous clusterization. The overall amount of storage space B must be now redistributed among the layers in $\mathcal{L} = \mathcal{L}_{new} \cup \mathcal{L}_{old} \cup \{L_0\}$. Adopting the same criterion as the non-incremental approach (see Section 4.1.2) is likely to result in changing the amount of storage space assigned to layers in $\mathcal{L}_{old}$, thus requiring also all layers non-affected by data updates to be repartitioned. This should be avoided, as it would imply to re-scan all data points. Thus, in the incremental approach, a different strategy to distribute the available storage space B among layers is adopted. This strategy aims at being fair and restricting as much as possible the set of preexisting layers to be repartitioned.

Layers in $\mathcal{L}_{old}$ and $\mathcal{L}_{new}$ and the layer of outliers L_0 will be considered into three distinct phases, to be executed in the following order.

Partitioning layers in $\mathcal{L}_{old}$. Let $\widehat{B}(\mathcal{L}_{old})$ denote the portion of B which has to be assigned to all the layers in $\mathcal{L}_{old}$. According to a fair distribution of the available storage space B between $\mathcal{L}_{old}$ and $\mathcal{L}_{new}$, it should result:

$$\widehat{B}(\mathcal{L}_{old}) = \frac{\mathit{SSE}(\mathcal{L}_{old})}{\mathit{SSE}(\mathcal{L})} \cdot B,$$

where: $SSE(\mathcal{L}_{old}) = \sum_{L \in \mathcal{L}_{old}} SSE(L)$ is an estimate of the overall inhomogeneity of layers in $\mathcal{L}_{old}$, and: $SSE(\mathcal{L}) = \sum_{L \in \mathcal{L}} SSE(L)$ measures the overall inhomogeneity of all the layers resulting from the clusterization. Notice that the *SSE* of each layer L is computed by accessing the aggregate data $sum(L)$, $sum^2(L)$, $count(L)$, $MBR(L)$, stored in the tuple in $\mathcal{L}$ corresponding to L: $SSE(L) = sum^2(L) - \frac{(sum(L))^2}{Vol(L)}$.

Let $B(\mathcal{L}_{old}) = \sum_{L \in \mathcal{L}_{old}} B(L)$ be the amount of storage space consumed by the summarization of all the layers in $\mathcal{L}_{old}$. First $\widehat{B}(\mathcal{L}_{old})$ is compared to $B(\mathcal{L}_{old})$. The idea is that if $B(\mathcal{L}_{old})$ is pretty "close" to $\widehat{B}(\mathcal{L}_{old})$ there not need to repartition layers in $\mathcal{L}_{old}$. In particular, in order to decide whether $B(\mathcal{L}_{old})$ is close to $\widehat{B}(\mathcal{L}_{old})$, a threshold parameter t is introduced. Thus if $|\widehat{B}(\mathcal{L}_{old}) - B(\mathcal{L}_{old})| < t \cdot \widehat{B}(\mathcal{L}_{old})$, the preexisting partition of layers in $\mathcal{L}_{old}$ will not be changed.

Otherwise layers in $\mathcal{L}_{old}$ are repartitioned depending on which of the following cases occurs:

- $B(\mathcal{L}_{old}) > (1+t) \cdot \widehat{B}(\mathcal{L}_{old})$: this means that the amount of storage space currently invested to summarize layers in $\mathcal{L}_{old}$ is on the whole too large (according to the adopted fair-distribution criterion); thus some layers in $\mathcal{L}_{old}$ are repartitioned by means of a coarser-grain grid in order to release some storage space;
- $B(\mathcal{L}_{old}) < (1-t) \cdot \widehat{B}(\mathcal{L}_{old})$: in this case the storage space currently invested to summarize $\mathcal{L}_{old}$ is increased, by repartitioning by means of a finer-grain grid the layers in $\mathcal{L}_{old}$ which are the most in need of a finer partition.

In order to choose the layers to be repartitioned, for each layer L in $\mathcal{L}_{old}$, $\widehat{B}(L) = \frac{SSE(L)}{SSE(\mathcal{L})} \cdot B$ is evaluated. The value of $\widehat{B}(L)$ is a fair portion of the available storage space to be assigned to L.

A layer L in $\mathcal{L}_{old}$ such that $B(L) > \widehat{B}(L)$ is said to be *indebted*, in the sense that it is assigned an amount of storage space larger than the amount it would be assigned in a fair space distribution based on its relative inhomogeneity. So it is "in debt" of some storage space to other layers. On the contrary, a layer L such that $B(L) < \widehat{B}(L)$ is said to be *creditor*, in the sense that it is assigned an amount of storage space smaller than the one it would need according to its *SSE*. That is, it is creditor of some storage space.

Consider the case that $B(\mathcal{L}_{old}) > (1+t) \cdot \widehat{B}(\mathcal{L}_{old})$ holds. Then, it is straightforward to see that there is at least one indebted layer in $\mathcal{L}_{old}$ such that $B(L) > (1+t) \cdot \widehat{B}(L)$. Let L^* be the most indebted layer in $\mathcal{L}_{old}$. The idea is to deprive L^* of some storage space in order to make the overall space consumed by layers in $\mathcal{L}_{old}$ closer to $\widehat{B}(\mathcal{L}_{old})$. In particular, a portion of storage space is stolen from $B(L^*)$, which makes L^* creditor of $\frac{t}{2} \cdot \widehat{B}(L^*)$. Therefore, the amount of storage space stolen from L^* is:

$$B^-(L^*) = B(L^*) - \left(1 - \frac{t}{2}\right) \cdot \widehat{B}(L^*).$$

Then L^* is repartitioned by investing the amount of storage space $B(L^*) - B^-(L^*)$. If at the end of this step $B(\mathcal{L}_{old}) > \widehat{B}(\mathcal{L}_{old})$ still holds, then the layer in $\mathcal{L}_{old}$ which is the most in debt is chosen and deprived of some storage space, using the same strategy as above. This process goes on until $B(\mathcal{L}_{old}) \leq \widehat{B}(\mathcal{L}_{old})$. That is, layers which are "very much indebted" are selected and made "pretty" in credit (a threshold value $t/2$ is used to estimate that a layer is creditor in a small extent): this strategy aims at reaching rapidly the condition $B(\mathcal{L}_{old}) \leq \widehat{B}(\mathcal{L}_{old})$, by reducing the number of layers to be repartitioned, which is mandatory for the efficiency requirements of the incremental approach.

In the case that $B(\mathcal{L}_{old}) < (1-t) \cdot \widehat{B}(\mathcal{L}_{old})$ an analogous approach is adopted: the layer L^* which is creditor of the largest amount of storage space is chosen its storage space is increased by adding to it:

$$B^+(L^*) = \left(1 + \frac{t}{2}\right) \cdot \widehat{B}(L^*) - B(L^*),$$

which means making L^* indebted of at most $\frac{t}{2} \cdot B(L^*)$. Then L^* is repartitioned, and this procedure is reiterated on the other creditors in $\mathcal{L}_{old}$ until $\widehat{B}(\mathcal{L}_{old}) \geq B(\mathcal{L}_{old})$.

By means of experiments, the threshold value $t = 20\%$ has resulted in preserving the accuracy of the updated histogram and effectively limiting the number of layers to be repartitioned.

If a layer $L \in \mathcal{L}_{old}$ is chosen to be repartitioned (as it is creditor or indebted in too large extent), the *Layer* value of the outliers which were summarized in the buckets of L at some previous step is assigned the value -1. These outliers will be considered for summarization into some bucket at the following step.

In the following, outliers whose *Layer* value is -1 will be said to be *new outliers*, whereas outliers whose *Layer* value is greater than or equal to 0 will be said to be *old outliers*.

Partitioning layers in $\mathcal{L}_{new}$. Layers $L_1, \ldots, L_\alpha$ in $\mathcal{L}_{new}$ are partitioned sequentially according to the same scheme adopted in the non-incremental approach. The amount of storage space invested to partition layers in $\mathcal{L}_{new}$ is $B(\mathcal{L}_{new}) = \frac{SSE(\mathcal{L}_{new})}{SSE(\mathcal{L})} \cdot B$. Then for each $i \in [1..\alpha]$, the layer L_i is summarized according to the grid-partitioning scheme described in Section 4.1.2 by investing the amount of storage space:
$B(L_i) = B_i \cdot \frac{SSE(L_i)}{SSE(L_0) + \sum_{j=i}^{\alpha} SSE(L_j)}$,
where $B_1 = B(\mathcal{L}_{new})$ and B_i is the portion of $B(\mathcal{L}_{new})$ which is left from the summarization of $L_1, \ldots, L_{i-1}$.

Partitioning L_0. Let $B' = B - B(\mathcal{L}_{new}) - B(\mathcal{L}_{old})$ be the amount of storage space which can be invested to summarize L_0, i.e. the portion of B which is left from summarizing layers in $\mathcal{L}_{new}$ and $\mathcal{L}_{old}$. L_0 is repartitioned if one of the following cases occurs:

1. $B(L_0) \geq B'$: this means that the current bucketization of L_0 makes the overall storage space consumption of the histogram exceed B, thus L_0 must be repartitioned using a coarser-grain grid;
2. $B(L_0) \leq (1-t) \cdot B'$: this means that the space currently invested to partition L_0 is too small (according to the threshold t), thus L_0 must be repartitioned using a finer-grain grid.

If either case 1 or case 2 occurs, L_0 must be repartitioned, thus a new grid is defined on L_0 (by investing the amount of storage space $B - B(\mathcal{L}_{\text{new}}) - B(\mathcal{L}_{old})$). In this case both new and old outliers are scanned, and each outlier is summarized either into a c-bucket or into an o-bucket, depending on whether it lies into the range of some c-bucket or not.

Otherwise, if neither case 1 nor case 2 occurs, the existing grid-partitioning of L_0 is kept and the current summarization is updated as follows. First, the new outliers are scanned and summarized into either a c-bucket or an o-bucket, as for the previous case. Then, the buckets of L_0 are deprived of the outliers which lie into the range of some newly created c-bucket.

Details on how these tasks are accomplished in the implementation are given in Section 4.3.

4.2.3 Step III: rearrangement of buckets

The task accomplished at this step consists in applying the same physical representation scheme as the non-incremental approach to the set of buckets resulting from Step II. The up-to-date histogram consists of buckets of four types: 1) c-buckets resulting from partitioning layers in $\mathcal{L}_{new}$, 2) c-buckets resulting from repartitioning selected layers in $\mathcal{L}_{old}$, 3) c-buckets inherited from the previous histogram which refer to layers in $\mathcal{L}_{old}$ which have not been repartitioned, 4) o-buckets partitioning L_0 (these buckets can result either from updating the o-buckets of the previous histogram or from repartitioning L_0). The preexisting arrangement of buckets is not exploited to rearrange new buckets, as this does not result in a relevant overhead. In fact all the operations needed to accomplish this task are performed in main memory (where the new bucketization is stored), without accessing disk-resident data.

The algorithm implementing steps I to III is shown in Fig. 4.8.

4.3 Costs of the non-incremental and incremental approaches

The difference in the number of disk accesses between the two approaches is due to two reasons. First, the adoption of the incremental clustering, which is likely to result in much fewer disk accesses w.r.t. the non-incremental one. Secondly, the strategy adopted at Step II, which aims at limiting the number of layers to be repartitioned, avoiding rescanning the whole data. As regards the

INPUT D: a multi-dimensional data distribution;
B: available amount of storage space for representing the histogram;
H: the histogram currently built on D;
u: the bulk of updates to be propagated to H;

OUTPUT H':an up-to-date histogram on D within B;

begin
$\langle \mathcal{L}_{old}, \mathcal{L}_{new}, L_0 \rangle$:= IncrementalDBSCAN(H, D, u);
$Partitioned$=$\emptyset$;
$NewBuckets$=$\emptyset$;
$\widehat{B}(\mathcal{L}_{old}) = \frac{SSE(\mathcal{L}_{old})}{SSE(\mathcal{L})} \cdot B$;
$B(\mathcal{L}_{old}) = \sum_{L \in \mathcal{L}_{old}} B(L)$;
if ($B(\mathcal{L}_{old}) \geq (1+t) \cdot \widehat{B}(\mathcal{L}_{old})$) **then begin**
  while ($B(\mathcal{L}_{old}) \geq (1+t) \cdot \widehat{B}(\mathcal{L}_{old})$) **do begin**
    $L = SelectMostIndebted(\mathcal{L}_{old})$;
    $Partitioned$=$Partitioned \cup L$;
    $B(L) = (1-\frac{t}{2}) \cdot \frac{SSE(L)}{SSE(\mathcal{L})} \cdot B$;
    $NewBuckets$= $NewBuckets \cup GridPartition(L,\ B(L))$;
  endwhile
elsif ($B(\mathcal{L}_{old}) \leq (1-t) \cdot \widehat{B}(\mathcal{L}_{old})$) **then begin**
  while ($B(\mathcal{L}_{old}) \leq (1-t) \cdot \widehat{B}(\mathcal{L}_{old})$) **do begin**
    $L = SelectMostCreditor(\mathcal{L}_{old})$;
    $Partitioned$=$Partitioned \cup L$;
    $B(L) = (1+\frac{t}{2}) \cdot \frac{SSE(L)}{SSE(\mathcal{L})} \cdot B$;
    $NewBuckets$= $NewBuckets \cup GridPartition(L,\ B(L))$;
  end_while
endif;
$\mathcal{L}_{old} = \mathcal{L}_{old} - Partitioned$;
for ($L \in \mathcal{L}_{new}$) **do begin**
  $B(L) = \frac{L.SSE}{L_0.SSE + SSE(\mathcal{L}_{new})} \cdot (B - size(Partitioned) - size(\mathcal{L}_{old}))$;
  $NewBuckets$= $NewBuckets \cup\ GridPartition(L, B(L))$;
  $Partitioned$=$Partitioned \cup L$;
endfor;
$B' = B - size(Partitioned) - size(\mathcal{L}_{old})$;
if ($B(L_0) \leq B'$ **AND** $B(L_0) \geq (1-t) \cdot B'$) **then begin**
  $O\text{-}Buckets$= $H.O\text{-}Buckets$;
  $DistributeNewOutliers(NewBuckets,\ O\text{-}Buckets)$;
  $MoveOutliers(NewBuckets,\ O\text{-}Buckets)$;
else begin
  $newSize = B - size(Partitioned) - size(\mathcal{L}_{old})$;
  $O\text{-}Buckets$= $PartitionAndDistribute(L_0, newSize, NewBuckets)$;
endif;
H' =$Assemble(NewBuckets, UnchangedBuckets(H, \mathcal{L}_{old}), O\text{-}Buckets)$;
return H';
end

Fig. 4.8. Incremental *CHIST* algorithm

former aspect, the extent of the benefit introduced by the use of an incremental clustering approach strictly depends on the particular clustering algorithm invoked. In the case of DBSCAN, no simple formula is known to provide the speedup factor corresponding to the use of its incremental version, thus the speedup must be determined experimentally.

As regards the second aspect, the number of disk accesses can be compared as follows. In the non incremental approach, after accomplishing the clusterization, a region query must be posed corresponding to the *MBR* of each detected dense cluster to partition it according to the grid; then, the list of outliers must be scanned to distribute them among c-buckets and o-buckets. Thus, denoting the number of dense clusters as c, the number of data points as N and the number of pages containing outliers as *Out*, the number of disk accesses is $\mathcal{O}(c \cdot \log N + Out)$ (it is assumed that a multi-dimensional index enabling region queries to be answered with $\log N$ accesses is maintained, as well as an inverted index of the pages containing outliers). As regards the incremental approach, let c' be the number of clusters which need to be partitioned, *OldOut* the number of pages containing the old outliers and *NewOut* the number of pages containing the new outliers. It is necessary to pose c' region queries to partition the dense clusters and *NewOut* pages pages have to be scanned in order to distribute the new outliers among c-buckets and o-buckets. Moreover, if it is the case that L_0 must be repartitioned, *OldOut* pages have to be scanned in order to repartition it and possibly adsorb old outliers into new c-buckets. Otherwise, if L_0 does not need repartitioning, only b_{new} region queries must be posed on the set of old outliers to possibly adsorb some of them into new c-buckets (b_{new} denotes the number of the new c-buckets). Therefore, the overall number of disk accesses is $\mathcal{O}(c' \cdot \log N + NewOut + X)$, where X is either $b_{new} \cdot \log OldOut$ (if L_0 is not repartitioned) or *OldOut* (if L_0 must be repartitioned). Observe that in order to support the incremental approach an inverted index is also maintained on the newly detected outliers (which allows us to scan all the new outliers by means of *NewOut* accesses) as well as a multi-dimensional index to answer region queries on old outliers with $\log Out$ accesses. Notice that in the worst case $NewOut + X = Out$ (in the non-incremental approach there is no distinction between new and old outliers, thus $Out = NewOut + OldOut$), but in the case that the outlier layer is not repartitioned $NewOut + X$ can be reasonably assumed much smaller than *Out*. Moreover c' can be assumed much smaller than c. Therefore if the number of outliers is "small" w.r.t. the whole data size, then the adoption of the incremental strategy always results in a relevant benefit, otherwise the extent of this benefit depends on the probability that L_0 must be repartitioned.

The latter issue cannot be investigated but experimentally, as well as the speedup due due adoption of the incremental clustering strategy. Therefore in the following section an experimental analysis of the overall benefit of the incremental approach will be provided.

4.4 Experimental analysis

This section analyzes the performances of the proposed clustering-based histogram construction. The accuracy of query estimates is measured and compared to the accuracy yielded by the *GHBH* technique, proposed in the previous chapter, that has been shown to outperform the state-of-the-art techniques. Then, experiments testing the effectiveness of the incremental approach, comparing it with the from-scratch execution of *CHIST* are presented.

4.4.1 Comparing *CHIST* with *GHBH*

In this section some experimental results are presented, comparing the accuracy of estimating query selectivity by means of *CHIST* with *GHBH*. The experiment were conducted on both synthetic and real-life data. Synthetic data were obtained by adopting the same generator described in Section 3.9.1. However, in Section 3.9.6 only a $8 \times 16 \times 256 \times 1024$ data distribution has been considered, on which *GHBH* yields very accurate estimates (20% average relative error in the most difficult setting). In that context "easy" data distribution were studied because, otherwise, the other techniques, especially wavelet-based ones, would have yielded too high errors, and the comparison would have not been meaningful. In this case, as only *CHIST* and *GHBH* will be compared, the accuracy of the techniques can be compared on synthetic data more difficult to be summarized (i.e. data consisting of a larger number of tuples defined on a larger data domain and with a higher dimensionality).

As regards experiments on real-life data, *Forest Cover* (see Section 3.9.2) has been considered. Specifically, in the following FC_{10} will denote the 10-dimensional data distribution already described in Section 3.9.2, and FC_5 will denote the projection of these data on the five attributes with the largest domain.

Diagrams (a, b) in Fig. 4.9 refer to a 4-dimensional synthetic data ($n = 1000$; $T = 100\,000$; $r = 100$; $z_{min} = 0.5$; $z_{max} = 2.5$; $l_{min} = 30$; $l_{max} = 200$), whereas Diagrams (c, d) refer to an 8-dimensional synthetic data ($d = 8$; $n = 1000$; $T = 200\,000$; $r = 200$; $z_{min} = 0.5$; $z_{max} = 2.5$; $l_{min} = 30$; $l_{max} = 200$). In both cases the DBSCAN parameters have been set as $MinPts = 4$ and $Eps = 4$. Observe that the differences in accuracy gets smaller as either the available storage space or the query selectivity increases. This is mainly due to the fact that both techniques perform very well for both large available storage space and large query selectivity. The difference between the two techniques is more relevant when the available storage space is very small. In this case, *GHBH*, which adopt a top-down partitioning strategy, find difficult reaching all the dense regions of data, thus some of them are summarized together with sparse ones. This explain also why the differences are more relevant also for low selectivities, where the homogeneity of each single bucket is more relevant than in larger queries.

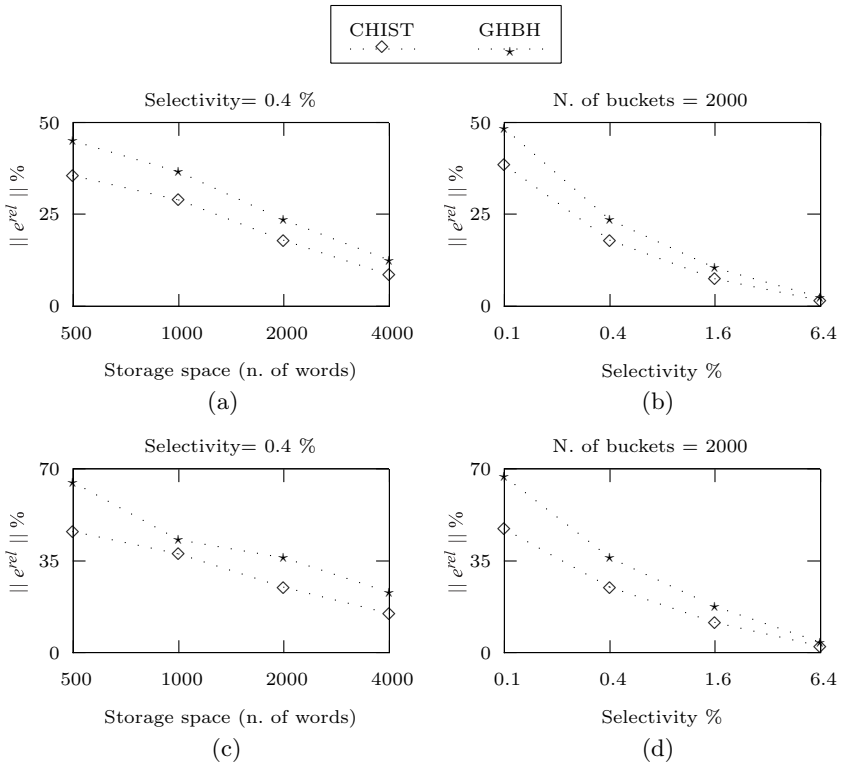


Fig. 4.9. Accuracy of techniques on 4D (a, b) and 8D (c, d) synthetic data

Diagrams (a, b) in Fig. 4.10 were obtained on FC_5, whereas diagrams (c, d) refer to FC_{10} (in these cases, the adopted parameters for DBSCAN were $MinPts$= 4 and Eps= 25). The same differences between the two techniques, already observed for synthetic data, has been noticed in these experiments.

4.4.2 Efficiency of the incremental approach

In this section the effectiveness of the incremental approach is compared with the from scratch execution of *CHIST*. To this end, given a multi-dimensional data distribution D, bulks of updates are simulated as follows. First, a distribution D' is generated on the same domain and using the same data generator as D. Then a bulk of insertions on D is created by randomly extracting some points from D'. The idea of extracting points from D' to be inserted into D is that this allows us to simulate both the creation of new dense clusters and new outliers in D. Deletions on D consist of randomly selected points of D. Thus a bulk of updates is a set of insertions and deletions, generated as explained above. For a bulk of updates u, the percentage of insertions in u will be denoted as p_u.

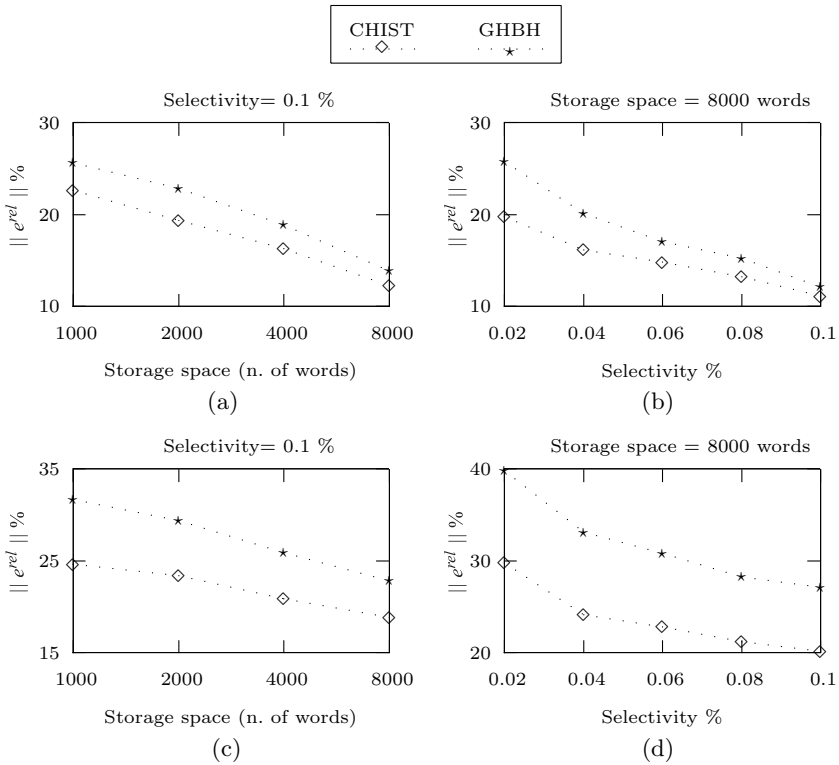


Fig. 4.10. Accuracy of techniques on FC_5 (a, b) and FC_{10} (c, d) real-life data

Diagrams in Fig. 4.11(a,b) refer to a 4D data distribution of size $1\,000^4$, while diagrams in Fig. 4.11 (d,e) refer to an 8D data distribution of size 1000^8.

Fig. 4.11 (a,c) shows the speedup due to the use of Incremental *CHIST* versus the size of updates (expressed as percentage of the data size) on a 4D and a 8D synthetic data distribution, respectively. For a bulk of updates u, the speedup is the ratio $\frac{N.\ of\ pages\ accessed\ by\ CHIST}{N.\ of\ pages\ accessed\ by\ Incremental\ CHIST}$. Fig. 4.11 (a,c) show that the benefit of using the incremental approach is very relevant (in both cases a speedup value of about 200 for update size of 1% has been achieved). As expected, the speedup decreases as the size of the updates gets larger. Observe that the speedup depends also on the type of updates: the larger the percentage of insertions, the higher the speedup. This is in accordance with [39], where it was observed that deletions on the average result in more complex changes of the clusterization, as they involve a larger number of preexisting clusters than insertions.

Diagrams in Fig. 4.11 (b,d) study the accuracy of the incremental approach, compared with that of the non-incremental one. They depict the ratio $e = \frac{e_{ni}}{e_i}$ between the relative errors provided by the non-incremental approach

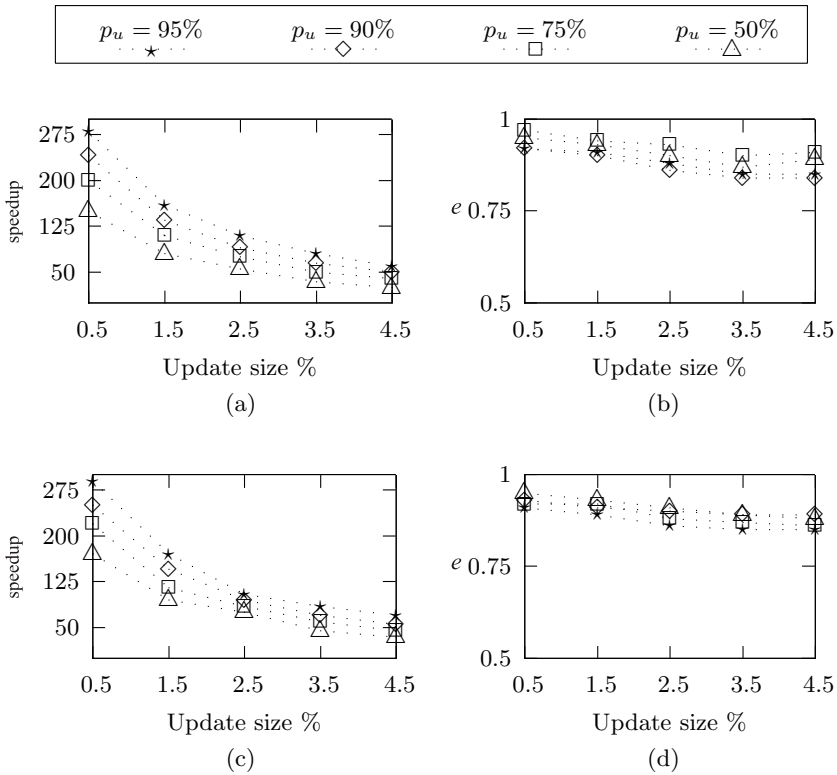


Fig. 4.11. Speedup and accuracy variation on 4D (a, b) and 8D (c,d) synthetic data

(i.e. e_{ni}) and and the incremental one (i.e. e_i) versus the size of updates. Experiments were conducted using 2000 buckets and on two workloads of 50000 queries of selectivity between 0.4% and 0.6%. Fig. 4.11 (b,e) show that the ratio $\frac{e_{ni}}{e_i}$ is close to 1 and is almost unaffected by the size of updates. This means that the adoption of the incremental approach does not result in degrading accuracy w.r.t. the non-incremental one.

Conclusions

In this thesis the problem of effectively summarizing multi-dimensional data as support for several application contexts has been investigated. Some of these contexts are selectivity estimation of range predicates for supporting query optimization, efficient exploratory data analysis in OLAP applications, window queries in spatial databases, and load balancing for parallel query execution. All these application scenarios require to efficiently estimate range queries over some data distribution. Data summarization has been widely accepted as the main solution to efficiently estimate range queries, especially on large amounts of data. A wide overview of several existing techniques for data summarization has been presented. All those techniques, more or less dramatically, incur in the curse of dimensionality, thus yielding poor accuracy in estimates as the dimensionality of summarization data becomes larger, unless the estimates are made less efficient.

An effort towards an improvement of existing histogram-based summarization techniques, which is particularly significant in the high-dimensionality settings, has been proposed in this thesis.

The greatest problem in approximating multi-dimensional data by means of histograms is the construction of buckets summarizing together dense and sparse regions of the data domain, thus obtaining buckets containing very skewed data distributions, which can not be approximated by a summary information with acceptable accuracy. There are two reasons why this could happen:

1. the space bound is too small for enabling to isolate the dense regions by means of a top-down iterative splitting strategy;
2. the splitting strategy itself could be ineffective in directing the splitting towards the isolation of dense regions;

The former is an intrinsic problem of all the techniques which base the data domain partition on a hierarchical strategy. The problem can be partially overcome by adopting an efficient representation model enabling a larger number of buckets to be constructed within the storage space bound. To this aim, the

hierarchical partition paradigm has been exploited to make the representation of buckets more efficient w.r.t. the traditional "flat" representation scheme adopted by classical histograms. Two new classes of histograms based on hierarchical binary partitions have been proposed, namely *HBH* and *GHBH*. Both represent the partition by means of a very compact bit-string representing a binary tree corresponding to the binary space partition. This representation can be viewed as a form of lossless compression applied on top of the common lossy compression on which histograms are based. *GHBH* differs from *HBH* as it adopts a grid-constrained hierarchical partition, where splits partitioning any block of data must be laid onto a grid dividing the block into a fixed number of sub-blocks with equal size. The space-efficient physical representation model enables *HBH* and *GHBH* to store a larger number of buckets within the same storage space bound which enables more accurate estimations than the traditional *MBR*-based representation model. In fact, even though the *MBR* (minimum bounding rectangle) can describe more accurately the actual data distribution inside each bucket, it requires a storage space to be represented which could be invested to construct a larger number of less accurate buckets. From experiments, it resulted that accuracy provided by a larger number of less accurate buckets overwhelm that provided by a smaller number of more accurate buckets.

However, the advantage of a larger number of buckets is actually relevant only if the strategy adopted to construct them is effective. In fact, if the splitting strategy is not effective in directing the splitting towards the isolation of dense regions, more buckets could not yield significant benefits. Since constructing the optimal histogram (i.e., the histogram minimizing a non-trivial measure of the error within a storage space bound) is practically infeasible, several heuristics for guiding a greedy algorithm have been investigated. A new heuristic performing definitely better than other ones, comprehending some well known heuristics previously proposed in literature, has been found to effectively drive the data domain partitioning to isolate dense from sparse regions, without requiring a more expensive computational cost.

By means of experiments, *GHBH* has been shown to outperform other state-of-the-art techniques, such as *MHIST*, *Min-Skew*, *GENHIST*, and other wavelet-based ones. The differences in accuracy result more and more evident as the dimensionality increases, as the number of buckets required to accurately summarize data increases (thus the benefits of the space-efficient representation model becomes more evident) as well as the difficulty in isolating dense regions in more and more skewed data (as it usually happens when the number of dimensions increases, thus enlarging the differences between a good and a poor heuristics guiding the partitioning). Moreover, it has been shown that the *GHBH* construction times does not depends on the heuristic, and that the larger number of buckets does not slow down query estimates, as their hierarchical organization enables to efficiently access them.

The proposal of *GHBH* might issue new research challenges in the direction of improving histograms construction techniques by exploiting the value

of lossless-compression-based representation models for storing histograms. Indeed, the proposed specific compression paradigm, as it exploits the hierarchical scheme adopted to construct the histogram, cannot be used to enhance the physical representation of different classes of histogram, such as wavelet-based ones, as well as techniques like *GENHIST*, which enables bucket overlapping. Therefore it would be interesting to design ad-hoc compression-based representation models which are suitable for other summarization techniques, and thus to investigate more generally the value of compression in the context of approximate data structures.

Despite *GHBH* represents a significant improvement within the class of histograms based on hierarchical binary partitions, as remarked before, there is an intrinsic problem affecting all the techniques which base the data domain partitioning on a hierarchical strategy and making the curse of dimensionality persist. In fact, even the space-efficient representation paradigm, jointly with the most effective partition strategy, could be not suitable to accurately approximate data defined over large multi-dimensional domains, as the top-down strategy could require an excessive number of splits to isolate all the dense regions. The opposite strategy, i.e. the bottom-up one, seems to be the only choice in those situations. The idea of partitioning data by means of a bottom-up approach, which starts from small regions and progressively aggregate data in larger and larger regions, have not found wide application in histogram-based compression techniques. *GENHIST*, generally considered the state-of-the-art in compression techniques, is the most known technique adopting this strategy.

The problem of isolating dense regions, however, has been widely studied in the data mining context, specifically in the *data clustering* problem. In this thesis, an approach which is based on the exploitation of a clustering algorithm for locating dense regions before partitioning them has been introduced. This solution, obviously avoid the problem of summarizing dense and sparse regions together, as dense and spare regions are summarized separately. The proposed technique, namely *CHIST*, exploits the well known *DBSCAN* algorithm. *DBSCAN* has been chosen instead of other techniques as it is considered the most effective in isolating dense regions. Clusters obtained by *DBSCAN* are considered as layers, and each layer is partitioned separately by means of a regular grid. More general partitioning schemes, such that proposed by *GHBH* have not been adopted since layers defined by *DBSCAN* are likely to be already quite homogeneous. In addition, a grid partitioning can be computed more efficiently, and this represent a significant advantage especially in the case of incremental maintenance of the histogram, which has proposed to make *CHIST* suitable also for approximating evolving data sets. In this case, the incremental maintenance of the histogram is based on the *Incremental DBSCAN* algorithm. The estimate accuracies yielded by *CHIST* have been compared to those yielded by *GHBH*. *CHIST* has been found to perform better than *GHBH*, when query selectivity are particularly small or the compression ratio is very high. This is due mainly to the difficulty of the top-

down approach in locating dense region which becomes particularly evident when the storage space bound is "quite" low or the query is defined of a range with size much smaller than the average size of the histogram buckets. However, in these "difficult" settings, *GHBH* still provides acceptable error rates, while other techniques provide estimates which are completely unacceptable. The effectiveness of the incremental approach has been tested, showing that it enables to achieve a relevant speedup factor w.r.t. the from-scratch histogram construction and that its adoption preserves accuracy as data changes.

Future work on *CHIST* will be devoted to investigating how other clustering techniques can be embedded into the scheme of general *CHIST* algorithm, and how their adoption affects both the accuracy and the construction time of histogram. Even though *DBSCAN* is very effective in locating dense regions, there are two main problems related to its adoption:

- *DBSCAN* is a parametric technique. Setting its parameters could result quite difficult. Even though *CHIST* performances have not resulted strongly affected by *DBSCAN* parameters as long as *DBSCAN* manage in isolating dense regions, completely wrong parameters could make *DBSCAN* find no dense regions or too many dense regions, thus annihilating the advantages of *CHIST* w.r.t. other techniques.
- the time complexity of *DBSCAN* is $\mathcal{O}(N_z \log N_z)$, where N_z is the number of tuples to be processed, when efficient indexing techniques are available. As at high dimensionality the curse of dimensionality makes indexing techniques inefficient (finding an efficient multi-dimensional indexing technique for high-dimensionality settings is still an open problem), the complexity becomes $\mathcal{O}({N_z}^2)$, which could become too heavy for large data sets.

Two approaches seem promising in order to overcome the first problem:

1. studying estimation techniques for effective *DBSCAN* parameters on the basis of the overall data distribution density and storage space bound;
2. adopting *OPTICS* instead of *DBSCAN*. *OPTICS* performs an ordering of tuples which subsequently enables to quickly find the clusters that *DBSCAN* would find on the basis of several different parameters. Interestingly, *OPTICS* can be used to define hierarchical clustering, which in turn can be adopted to define overlapping buckets with increasing size and decreasing density. In this case, kernel estimators [129] are likely to provide better performances w.r.t. the classical uniform distribution assumption inside buckets on which classical histograms are based.

Results in these two directions could enable overcoming only the problem related to the choice of the parameters. As regards the problem related to the complexity of the clustering operation, some ideas which could be developed are the following:

1. performing the summarization during the clustering operation and decreasing the clustering execution cost by accepting less accurate clusters.

Indeed, *DBSCAN* provides an accuracy which is unnecessary in the context on data summarization. In fact, the exact shape of clusters is not necessary, as they are later partitioned according to their *MBR*, which is a simple hyper-rectangle. The straightforward approach based on considering the cluster *MBR* during its construction, and adding all the points within the *MBR* to the cluster has been found to perform poorly, as it yields too large buckets with low density and high inhomogeneity. A possibility could be defining a technique similar to *DBSCAN*, which instead of clusters creates buckets starting from points with dense neighborhood and enlarging their *MBR*s until a measure of their homogeneity is maintained within some threshold.

2. exploiting data dimensionality reduction techniques, such as those based on statistical interaction models. This solution would avoid the problems due to high data dimensionality. However, this approach could be also used with other compression-techniques, and would not be specifically targeted to clustering-based histograms.

Of course, several other not mentioned approaches could be suitable, and may be that among those there is the most effective one. However, it is not uncommon in research that starting a study even towards a "not-so-correct" direction makes understand where actually the solution of the problem is hidden.

References

1. Aboulnaga A. and Chaudhuri S., Self-tuning histograms: building histograms without looking at data, *Proceedings of 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*: 181-192, June 1-3, 1999, Philadelphia (PA), USA.
2. Acharya S., Gibbons P. B., Poosala V., and Ramaswamy S., Join synopses for approximating query answering, *Proceedings of 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*: 275-286, June 1-3, 1999, Philadelphia (PA), USA.
3. Acharya S., Poosala V., and Ramaswamy S., Selectivity estimation in spatial databases, *Proceedings of 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*: 13-24, June 1-3, 1999, Philadelphia (PA), USA.
4. Ahrens J. H. and Dieter U., Sequential random sampling, *ACM Transactions on Mathematical Software (TOMS)*, 11(2): 157-169, June, 1985.
5. Alsabti K., Ranka S, and Singh V., A one-pass algorithm for accurately estimating quantiles for disk-resident data, *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB 1997)*: 346-355, August 25-29, 1997, Athens, Greece.
6. Ankerst M., Bruenig M. M., Kriegel H. P., and Sander J., OPTICS: ordering points to identify the clustering structure, *Proceedings of 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*: 49-60, June 1-3, 1999, Philadelphia (PA), USA.
7. Barbarà D., Du Mouchel W., Faloutsos C., Haas P. J., Hellerstein J. M., Ioannidis Y. E., Jagadish H. V., Johnson T., Ng R., Poosala V., Ross K. A., and Sevcik K. C., The New Jersey data reduction report, *IEEE Data Engineering Bulletin*, 20(4): 3-45, December 1997.
8. Beniger J. R. and Robyn D. L., Quantitative graphics in statistics: a brief history, *The American Statistician*, 32(1):1-11, February 1978.
9. Blohsfeld B., Korus D., and Seeger B., A comparison of selectivity estimators for range queries on metric attributes, *Proceedings of 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*: 239-250, June 1-3, 1999, Philadelphia (PA), USA.
10. Bradley P. S., Fayyad U. M., and Reina C., Scaling clustering algorithms to large databases, *Proceedings of 4th International Conference on Knowledge*

Discovery and Data Mining (KDD 1998): 9-15, August 27-31, 1998, New York City (NY), USA.

11. Bruno N., Chaudhuri S., and Gravano L., STHoles: a multi-dimensional workload aware histogram, *Proceedings of 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD 2001)*: 211-222, May 21-24, 2001, Santa Barbara (CA), USA.
12. Buccafurri F., Furfaro F., and Saccà D., Estimating range queries using aggregate data with integrity constraints: a probabilistic approach, *Proceedings of 8th International Conference on Database Theory (ICDT 2001)*: 390-404, January 4-6, 2001, London, UK.
13. Buccafurri F., Rosaci D., Pontieri L., and Saccà D., Improving Range Query Estimation on Histograms, *Proceedings of 18th International Conference on Data Engineering (ICDE 2002)*: 628-638, February 26-March 1, 2002, San Jose (CA), USA.
14. Buccafurri F., Furfaro F., Lax G., and Saccà D., Binary-Tree Histograms with Tree Indices, *Proceedings of 13th International Conference on Database and Expert Systems Applications (DEXA 2002)*: 861-870, September 2-6, 2002, Aix-en-Provence, France.
15. Buccafurri F., Furfaro F., Saccà D., and Sirangelo C., A Quad-Tree Based Multiresolution Approach for Two-dimensional Summary Data, *Proceedings of 15th International Conference on Scientific and Statistical Database Management (SSDBM 2003)*: 127-137, 9-11 July, 2003, Cambridge (MA), USA.
16. Cardenas A. F., Evaluation and selection of file organization - A model and system, *Communications of the ACM*, 16(9): 540-548, September 1973.
17. Cardenas A. F., Analysis and performance of inverted database structures. *Communications of the ACM*, 18(5): 253-263, May 1975.
18. Chamberlin D. D. and Boyce R. F., SEQUEL: a structured english query language, *Proceedings of 1974 ACM-SIGMOD Workshop on Data Description, Access and Control (SIGFIDET/SIGMOM 1974)*, Vol 1: 249-264, May 1-3, 1974, Ann Arbor (MI), USA.
19. Chakrabarti K., Garofalakis M., Rastogi R., and Shim K., Approximate query processing using wavelets, *Proceedings of 26th International Conference on Very Large Data Bases (VLDB 2000)*: 111-122, 10-14 Sept, 2000, Cairo, Egypt.
20. Chakrabarti K., Garofalakis M., Rastogi R., and Shim K., Approximate query processing using wavelets, *The VLDB Journal*, 10(2-3): 199-223, 2001.
21. Chaudhuri S. and Dayal U., An overview of data warehousing and OLAP technology, *SIGMOD RECORD*, 26(1): 65-74, March 1997.
22. Chaudhuri S., An overview of query optimization in relational systems, *Proceedings of 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1998)*: 34-43, June 1-3, 1998, Seattle (WA), USA.
23. Chaudhuri S., Motwani R., and Narasayya V., Random sampling for histogram construction: how much is enough?, *Proceedings of 1998 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1998)*: 436-447, June 2-4, 1998, Seattle (WA), USA.
24. Chaudhuri S. and Gravano L., Evaluating top-k selection queries, *Proc. of 25th International Conference on Very Large Data Bases (VLDB 1999)*: 397-410, September 7-10, 1999, Edinburgh, UK.
25. Chen C. M. and Roussopoulos N., Adaptive selectivity estimation using query feedback, *Proceedings of 1994 ACM SIGMOD International Conference on*

Managment of Data (SIGMOD 1994): 161-172, May 24-27, 1994, Minneapolis (MN), USA.
26. Cheung T.-Y., Estimating block accesses and number of records in file managment. *Communication of the ACM*, 25(7): 484-487, July 1982.
27. Christodoulakis S., *Estimating selectivities in data bases*, PhD Thesis, CSRG-136, Computer System Research Group, University of Toronto, Canada.
28. Christodoulakis S., Estimating record selectivities, *Information Systems*, 8(2): 105-115, 1983.
29. Christodoulakis S., Estimating block transers and join sizes, *Proceedings of 1983 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1983)*: 40-54, May 23-26, 1983, San Jose (CA), USA.
30. Christodoulakis S., Estimating block selectivities, *Information systems*, 9(1): 69-79, 1984.
31. Christodoulakis S., Implications of certain assumptions in database performance evaluation, *ACM Transactions on Database Systems (TODS)*, 9(2): 163-186, June 1984.
32. Codd E. F., A relational model of data for large shared data banks, *Communications of the ACM*, 13(6): 377-387, June 1970.
33. Comer D., The ubiquitous B-Tree, *Computing Surveys*: 11(2): 121-137, June 1979.
34. Deligiannakis A., Garofalakis M. N., and Roussopoulos N., A fast approximation scheme for probabilistic wavelet synopses, *Proceedings of 17th International Conference on Scientific and Statistical Database Management (SSDBM 2005)*: 243-252, 27-29 June, 2005, Santa Barbara (CA), USA.
35. Deshpande A., Garofalakis M., and Rastogi R., Independence is good: dependency-based histogram synopses for high-dimensional data, *Proceedings of 2001 ACM SIGMOD International Conference on Managment of Data (SIGMOD 2001)*: 199-210, May 21-24, 2001, Santa Barbara (CA), USA.
36. Donjerkovic D. and Ramakrishnan R., Probabilistic optimization of top N queries, *Proc. of 25th International Conference on Very Large Data Bases (VLDB 1999)*: 411-422, September 7-10, 1999, Scotland, UK.
37. Eckart C. and Young G., The approximation of one matrix by another of lower rank, *Psychometrika*, 1(3): 211-218, September 1936.
38. Ester M., Kriegel H. P., Sander J., and Xu X., A density-based algorithm for discorvering clusters in large spatial databases with noise, *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD 1996)*: 226-231, August 2-4, 1996, Portland (OR), USA.
39. Ester M., Kriegel H. P., Wimmer M., and Xu X., Incremental clustering for mining in a data warehousing environment, *Proceedings of 24th International Conference on Very Large Data Bases (VLDB 1998)*: 323-333, August 24-27, 1998, New York City (NY), USA.
40. Fan C. T., Muller M. E., and Rezucha I., Development of sampling plans by using sequential (item by item) selection techniques and digital computers, *Journal of the American Statistical Association*, 57(298): 387-402, June 1962.
41. Fedorowicz J. E., A zipfian model of inverted file storage requirements, *Proceedings of 12th Annual Pittsbusgh Conference on Modeling and Simulation*: 1393-1399, Apr 30-May 2, 1981, Pittsburgh (PA), USA.
42. Fedorowicz J. E., A zipfian model of an automatic bibliographic system: an application to MEDLINE, *Journal of the American Society for Information Science*, 33(4): 223-232, July 1982.

43. Fedorowicz J. E., Database evaluation using multiple regression techniques, *Proceedings of 1984 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1984)*: 70-76, June 18-21, 1984, Boston (MA), USA.
44. Furtado P. and Madeira H., Summary GRIDS: building accurate multidimensional histograms, *Proceedings of 6th International Conference on Database Systems for Advanced Applications (DASFAA 1999)*: 187-194, April 19-21, 1999, Hsinchu, Taiwan.
45. Garofalakis M. and Gibbons P. B., Wavelet synopses with error guarantees, *Proceedings of 2002 ACM SIGMOD International Conference on Managment of Data (SIGMOD 2002)*: 476-487, June 3-6, 2002, Madison (WI), USA.
46. Garofalakis M. and Gibbons P. B., Probabilistic wavelet synopses, *ACM Transactions on Database Systems (TODS)*, 29(1): 43-90, March 2004.
47. Garofalakis M. and Kumar A., Deterministic wavelet thresholding for maximum-error metrics, *Proceedings of 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2004)*: 166-176, June 14-16, 2004, Paris, France.
48. Garofalakis M. and Kumar A., Wavelet synopses for general error metrics, *ACM Transactions on Database Systems (TODS)*, 30(4): 888-928, December 2005.
49. Gibbons P. B. and Matias Y., New sampling-based summary statistics for improving approximate query answers, *Proceedings of 1998 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1998)*: 331-342, June 2-4, 1998, Seattle (WA), USA.
50. Gilbert A. C., Kotidis Y., Muthukrishnan S., and Strauss M. J., Optimal and approximate computation of summary statistics for range aggregates, *Proceedings of 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2001)*, May 21-23, 2001, Santa Barbara (CA), USA.
51. Gould P. R., Letting the data speak for themselves, *Annals of the Association of American Geographers*, 71(2): 166-176, June 1981.
52. Gray J., Bosworth A., Layman A., and Pirahesh H., Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total, *Proceedings of 12th International Conference on Data Engineering (ICDE 1996)*: 152-159, February 26-March 1, 1996, New Orleans (LA), USA.
53. Gray J., Chaudhuri S., Bosworth A., Layman A., Reichart D., Venkatrao M., Pellow F., and Pirahesh H.: Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total, *Data Mining and Knowledge Discovery*, 1(1): 29-53, March 1997.
54. Guha S., Rastogi R., and Shim, K., CURE: An efficient clustering algorithm for Large Databases, *Proceedings of 1998 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1998)*: 73-84, June 2-4, 1998, Seattle (WA), USA.
55. Guha S., Shim K, and Woo J., REHIST: relative error histogram construction algorithms, *Proceedings of 30th International Conference on Very Large Data Bases (VLDB 2004)*: 300-311, August 31-September 3, 2004, Toronto, Canada.
56. Gunopulos D., Kollios G., Tsotras V. J., and Domeniconi C., Approximating multi-dimensional aggregate range queries over real attributes, *Proceedings of 2000 ACM SIGMOD International Conference on Managment of Data (SIGMOD 2000)*: 463-474, May 14-19, 2000, Dallas (TX), USA.

57. Gunopulos D., Kollios G., Tsotras V. J., and Domeniconi C., Selectivity estimators for multidimensional range queries over real attributes, *The VLDB Journal*, 14(2): 137-154, April 2005.
58. Haas P. J. and Swami A. N., Sequential sampling procedures for query size estimation, *Proceedings of 1992 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1992)*: 341-350, June 2-5, 1992, San Diego (CA), USA.
59. Haas P. J. and Hellerstein J. M., Ripple joins for online aggregation, *Proceedings of 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*: 287-298, June 1-3, 1999, Philadelphia (PA), USA.
60. Harinarayan V., Rajaraman A., and Ullman J. D., Implementing data cubes efficiently, *Proceedings of 1996 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1996)*: 205-216, June 4-6, 1996, Montreal, Canada.
61. Heising W. P., Note on random addressing techniques. *IBM System Journal*: 2(2): 112-116, June 1963.
62. Hellerstein J. M., Haas P. J., and Wang H. J., Online aggregation, *Proceedings of 1997 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1997)*: 171-182, May 13-15, 1997, Tucson (AZ), USA.
63. Hill E., Analysis of an inverted data base structure, *Proceedings of 1978 International Conference on Information Storage and Retrieval (SIGIR 1978)*: 37-64, May 10-12, 1978, Rochester (NY), USA.
64. Hou W.-C., Ozsoyoglu G., and Taneja B. K., Statistical estimators for relational algebra expessions, *Proceedings of 7th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1988)*: 276-287, March 21-23, 1988, Austin (TX), USA.
65. Hou W.-C., Özsoyoglu G., Dogdu E, Error-constraint COUNT query evaluation in relational databases, *Proceedings of 1991 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1991)*: 278-287, May 29-31, 1991, Denver(CO), USA.
66. Immon W. H., *Building the data warehouse*, John Wiley and Sons, New York, 1996.
67. Ioannidis Y. E. and Wong E., Query optimization by simulated annealing, *Proceedings of 1987 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1987)*: 9-22, May 27-29, 1987, San Francisco (CA), USA.
68. Ioannidis Y. E. and Kang Y. C., Randomized algorithms for optimizing large join queries, *Proceedings of 1990 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1990)*: 312-321, May 23-25, 1990, Atlantic City (NJ), USA.
69. Ioannidis Y. E. and Christodoulakis S., On the propagation of errors in the size of join results, *Proceedings of 1991 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1991)*: 268-277, May 29-31, 1991, Denver(CO), USA.
70. Ioannidis Y. E. and Christodoulakis S., Optimal histograms for limiting worst-case error propagation in size of join results, *ACM Transactions on Database Systems (TODS)*, 18(4): 709-748, June 1984.
71. Ioannidis Y. E. Universality of serial histograms, *Proceedings of 19th International Conference on Very Large Data Bases (VLDB 1993)*: 256-267, August 24-27, 1993, Dublin, Ireland.

72. Ioannidis Y. E. and Poosala V., Balancing histogram optimality and practicality for query result size estimation, *Proceedings of 1995 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1995)*: 233-244, May 22-25, 1995, San Josè (CA), USA.
73. Ioannidis Y. E. The history of histograms (abridged), *Proceedings of 29th International Conference on Very Large Data Bases (VLDB 2003)*: 19-30, September 9-12, 2003, Berlin, Germany.
74. Jagadish H. V., Koudas N., Muthukrishnan S., Poosala V., Sevcik K., and Suel T., Optimal histograms with quality guarantees, *Proceedings of 24th International Conference on Very Large Data Bases (VLDB 1998)*: 275-286, August 24-27, 1998, New York City (NY), USA.
75. Jagadish, H. V., Jin, H., Ooi, B. C., and Tan, K.-L., Global optimization of histograms, *Proceedings of 2001 ACM SIGMOD International Conference on Managment of Data (SIGMOD 2001)*: 223-234, May 21-24, 2001, Santa Barbara (CA), USA.
76. Jain R. and Chlamtac I., The P^2 algorithm for dynamic calculation of quantiles and histograms without storing observations, *Communications of the ACM*, 28(10): 1076-1085, October 1985.
77. Jarke M. and Kock J., Query optimization in database systems, *Computing Surveys*, 16(2): 111-152, June 1984.
78. Jawerth, B. and Sweldens, W., An overview of wavelet based multiresolution analyses, *SIAM Review*, 36(3): 377-412, September 1994.
79. Kamel N. and King R., A model of data distributions based on texture analysis, *Proceedings of 1985 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1985)*: 319-325, May 28-31, 1985, Austin (TX), USA.
80. Kaufmann L. and Rousseeuw P. J., Clustering by means of medoids, *Statistical Data Analysis based on the L1 Norm and Related Methods*: 405-416, Elsevier/North Holland, Amsterdam, 1987.
81. Kaufman L. and Rousseeuw P. J., *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley, 2005.
82. Kober V. and Cristobal G., Fast recursive algorithms for short-time discrete cosine transform, *Electronic Letters*, 35(15): 1236-1238, July 1999.
83. König A. and Weikum G., Combining histograms and parametric curve fitting for feedback-driven query result-size estimation, *Proceedings of 25th International Conference on Very Large Data Bases (VLDB 1999)*: 423-434, September 7-10, 1999, Edinburgh, UK.
84. Kooi R. (1980) *The optimization of queries in relational databases*, PhD Thesis, Case Western Reserve University, Cleveland, Ohio.
85. Kooi R. and Frankfurth D., Query optimization in INGRES, *IEEE Database engineering bulletin*: 5(3): 2-5, September 1982.
86. Korn F., Johnson T., and Jagadish H. V., Range selectivity estimation for continuous attributes, *Proceedings of 15th International Conference on Scientific and Statistical Database Management (SSDBM 2003)*: 244-253, 28-30 July, 1999, Cleveland (OH), USA.
87. Lee J.-H., Kim D.-H., and Chung C.-W., Multi-dimensional selectivity estimation using compressed histogram information, *Proceedings of 1999 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1999)*: 205-214, June 1-3, 1999, Philadelphia (PA), USA.

88. Lefons E., Silvestri A., and Tangorra F., An analytic approach to statistical satabases, *Proceedings of 9th International Conference on Very Large Data Bases (VLDB 1983)*: 260-274, October 31-November 2, 1983, Florence, Italy.
89. Ling Y. and Sun W., An evaluation of sampling-based size estimation methods for selection in database systems, *Proceedings of 11th International Conference on Data Engineering (ICDE 1995)*: 532-539, March 6-10, 1995, Taipei, Taiwan.
90. Lipton R. J. and Naughton J. F., Query size estimation by adaptive sampling, *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*: 40-46, April 2-4, 1990, Nashville (TN), USA.
91. Lipton R. J., Naughton J. F., and Schneider D. A., Practical selectivity estimation through adaptive sampling, *Proceedings of 1990 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1990)*: 1-11, May 23-25, 1990, Atlantic City (NJ), USA.
92. Luk W. S., On estimating block accesses in database organizations, *Communication of the ACM*, 26(11): 945-947, November 1983.
93. MacQueen J. B., Some methods for classification and analysis of multivariate observations, *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1: 281-297, Berkeley, University of California Press, 1967.
94. Mackert L. F. and Lohman G. M, R* optimizer validation and performance evaluation for local queries, *Proceedings of 1986 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1986)*: 84-95, May 28-30, 1986, Washington (DC), USA.
95. Mackert L. F. and Lohman G. M, R* optimizer validation and performance evaluation for distributed queries, *Proceedings of 12th International Conference on Very Large Data Bases (VLDB 1986)*: 149-159, August 25-28, 1986, Kyoto, Japan.
96. Mamoulis N. and Papadias D., Selectivity estimation of complex spatial queries, *Proceedings of 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD 2001)*: 155-174, July 12-15, 2001, Redondo Beach (CA), USA.
97. Manku G. S., Rajagopalan S., and Lindsay B. G., Approximate medians and other quantiles in one pass and with limited memory, *Proceedings of 1998 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1998)*: 426-435, June 2-4, 1998, Seattle (WA), USA.
98. Marshall A. and Olkin I., *Inequalities: theory of majorization and its applications*, Academic press, New York, 1979.
99. Matias Y., Vitter J. S. and Wang M., Wavelet-based histograms for selectivity estimation, *Proceedings of 1998 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1998)*, June 2-4, 1998, Seattle (WA), USA.
100. Merrett T. H., Database cost analysys: a top-down approach, *Proceedings of 1977 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1977)*: 135-143, August 3-5, 1977, Toronto, Canada.
101. Merrett T. H. and Otoo E. J., Distribution models of relations, *Proceedings of 5th International Conference on Very Large Data Bases (VLDB 1979)*: 418-425, October 3-5, 1979, Rio de Janeiro, Brasil.
102. Moore G. E., Cramming more components onto integrated circuits, *Electronics Magazine*, 38(8): 114-117, April 1965.
103. Muralikrishna M. and DeWitt D. J., Equi-depth histograms for estimating selectivity factors for multi-dimensional, *Proceedings of 1988 ACM SIGMOD*

International Conference on Managment of Data (SIGMOD 1988): 28-36, June 1-3, 1988, Chicago (IL), USA.

104. Muthukrishnan S., Poosala V., and Suel T., On rectangular partitioning in two dimensions: algorithms, complexity and applications, *Proceedings of 7th International Conference on Database Theory (ICDT 1999)*: 236-256, January 10-12, 1999, Jerusalem, Israel.
105. Ng R. T. and Han J., Efficient and effectevive clustering methods for spatial data mining, *Proceedings of 20th International Conference on Very Large Data Bases (VLDB 1994)*; 144-155, September 12-15, 1994, Santiago de Chile, Chile.
106. Piatetsky-Shapiro G., The optimal selection of secondary indices is NP-complete, *ACM SIGMOD Record*, 13(2): 72-75, January 1983.
107. Piatetsky-Shapiro G., *A self-organizing database system - A different approach to query optimization*, PhD thesis, Department of Computer Science, New York University, 1984.
108. Piatetsky-Shapiro G. and Connell C., Accurate estimation of the number of tuples satisfying a condition, *Proceedings of 1984 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1984)*: 256-276, June 18-21, 1984, Boston (MA), USA.
109. Poosala V., Ioannidis Y. E., Haas P. J., and Shekita E. J., Improved histograms for selectivity estimation of range predicates, *Proceedings of 1996 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1996)*: 294-305, June 4-6, 1996, Montreal, Canada.
110. Poosala V. and Ioannidis Y. E., Estimation of query-result distribution and its application in parallel-join load balancing, *Proc. of 22nd International Conference on Very Large Data Bases (VLDB 1996)*: 448-459, September 3-6, 1996, Mumbai (Bombai), India.
111. Poosala V. and Ioannidis Y. E., Selectivity estimation without the attribute value independence assumption, *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB 1997)*: 486-495, August 25-29, 1997, Athens, Greece.
112. Raatikainen K. E. E., Simultaneous Estimation of Several Persentiles, *Simulation*, 49(4): 159-164, October 1987.
113. Rothnie J. B. Jr. and Lonzano T., Attribute based file organization in a paged memory environment, *Communications of the ACM*, 17(2): 63-69, February 1974.
114. Schkolnick M., Secondary index optimization, *Proceedings of 1975 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1975)*: 186-192, May 14-16, 1975, San Jose (CA), USA.
115. Schuegraf E. J., Compression of large inverted files with hyperbolic term distribution, *Information Processing and Management*, 12(6): 377-384, 1976.
116. Selinger P. G., Astrahan M. M., Chamberlin D. D., Lorie R. A., and Price T. G., Access path selection in a relational database management system, *Proceedings of 1979 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1979)*: 23-34, May 30-June 1, 1979, Boston (MA), USA.
117. Shanmugasundaram J., Fayyad U., and Bradley P. S., Compressed data cubes for OLAP aggregate query approximation on continuous dimensions, *Proceedings of 5th International Conference on Knowledge Discovery and Data Mining (KDD 1999)*: 223-232, August 15-18, 1999, San Diego (CA), USA.

118. Shoshani A. and Wong H. K. T., Statistical and scientific database issues, *IEEE Transactions On Software Engineering (TSE)*, 11(10): 1040-1047, October 1985.
119. Siler K. F., A stochastic evaluation model for database organization in data retrival systems, *Communications of the ACM*, 19(2): 84-95, February 1976.
120. Stollnitz E. J., DeRose T. D., and Salesin D. H., *Wavelets for Computer Graphics - Theory and Applications*, Morgan Kaufmann Publishers, San Francisco (CA), USA, 1996.
121. Stonebraker M., *The INGRES Papers: Anatomy of a Relational Database System*, Addison-Wesley, 1986
122. Sun W., Ling Y., Rishe N., and Deng Y. An instant and accurate size estimation method for joins and selection in a retrieval-intensive environment. *Proceedings of 1993 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1993)*: 79-88, May 26-28, 1993, Washington (DC), USA.
123. Swami A. and Gupta A., Optimization of large join queries, *Proceedings of 1988 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1988)*: 8-17, June 1-3, 1988, Chicago (IL), USA.
124. Vander Zanden B. T., Taylor H. M. and Bitton D., Estimating block accesses when attributes are correlated, *Proceedings of 12th International Conference on Very Large Data Bases (VLDB 1986)*: 119-127, August 25-28, 1986, Kyoto, Japan.
125. Vitter J. S., Faster methods for random sampling, *Commununications of the ACM*, 27(7): 703-718, July 1984.
126. Vitter J. S., An efficient algorithm for sequential random sampling, *ACM Transactions on Mathematical Software*, 13(1): 58-67, March 1987.
127. Vitter J. S., Wang M., and Iyer, B., Data cube approximation and histograms via wavelets, *Proceedings of the 1998 ACM CIKM International Conference on Information and Knowledge Management (CIKM 1998)*: 96-104, November 3-7, 1998, Bethesda (MD) USA.
128. Vitter J. S. and Wang M., Approximate computation of multidimensional aggregates of sparse data using wavelets, *Proceedings of 1999 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1999)*: 193-204, June 1-3, 1999, Philadelphia (PA), USA.
129. Wand M. P. and Jones M. C., Kernel smoothing, *Monographs on Statistics and Applied Probability*, Chapman & Hall, 1995.
130. Wang M., Vitter J. S., Lim L., and Padmanabhan S., Wavelet-based cost estimation for spatial queries, *Proceedings of 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD 2001)*: 175-196, July 12-15, 2001, Redondo Beach (CA), USA.
131. Whang K.-Y. and Wiederhold G., Estimating block accesses in database organizations: A closed noniterative formula, *Communications of the ACM*, 26(11): 940-944, November 1983.
132. Yao S. B., An attribute based model for database access cost analysis, *ACM Transactions on Database Systems (TODS)*, 2(1): 45–67, March 1977.
133. Yao S. B., Approximating block accesses in databases organizations. *Communications of the ACM*, 20 (4): 260-261, April 1977.
134. Yu C. T. and Chang C. C., Distributed qury processing, *Computing Surveys*, 16(4): 399-433, December 1984.

135. Zhang D., Gunopulos D., Tsotras V. J., and Seeger B., Temporal aggregation over data streams using multiple granularities, *Proceedings of 8th International Conference on Extending Database Technology (EDBT 2002)*: 646-663, March 25-27, 2002, Prague, Czech Republic.
136. Zhang D., Gunopulos D., Tsotras V. J., and Seeger B., Temporal and spatio-temporal aggregations over data streams using multiple time granularities, *Information Systems*, 28(1-2): 61-84 , March-April 2003.
137. Zhang T., Ramakrishnan R., and Livny M., BIRCH: An Efficient Data Clustering Method for Very Large Databases, *Proceedings of 1996 ACM SIGMOD International Conference on Managment of Data (SIGMOD 1996)*: 103-114, June 4-6, 1996, Montreal, Canada.
138. Zipf G. K., *Human behaviour and the principle of the least effort*, Addison-Wesley, Reading, Massachusets, 1949.

Index

Aboulnaga A., 54
Acharya S., 52
Ahrens J. H., 60
Alsabti K., 49

Beninger J. R, 40
Blohsfeld B., 58
Bruno N., 56
Buccafurri F., 43, 57

Cardenas A. F., 2
Chakrabart K., 38
Chaudhuri S., 49, 54
Chen C. M., 27
Cheung T.-Y., 3
Chlamtac I., 49
Christodoulakis S., 4, 26, 48, 50
Codd E. F., 1
Connell C., 49

Deshpande A., 53
DeWitt D. J., 50
Dieter U., 60

Eckart C., 53

Fedorowicz J., 26
Fedorowicz J. E., 26
Furtado P., 52

Garofalakis M., 39
Gibbons P. B., 39
Gilbert A. C., 44
Graunt J., 41
Guerry A. M., 40
Guha S., 44
Gunopulos D., 55, 58
Gupta A., 12

Haar A., 28
Heising W. P., 4
Hellerstein J. M., 21
Hill E., 26

Immon W. H., 13
Ioannidis Y. E., 12, 50, 51, 53

Jagadish H. V., 44
Jain R., 49
Juran J. M., 4

König A., 57
Kamel N., 49
Kang Y. C., 12
King R., 49
Kooi R., 41
Korn F., 27
Kumar A., 39

Lee J.-H., 57
Lefons E., 26
Lim L., 38
Lonzano T., 2
Luk W. S., 3, 4

Madeira H., 52
Manku G. S., 49
Matias Y., 36

Merret T. H., 10
Merrett T. H., 41, 48
Moore G. E., 1
Morlet J., 28
Muralikrishna M., 50
Muthukrishnan S., 48

Nightingale F., 41

Otoo E. J., 10, 41, 48

Padmanabhan S., 38
Pareto V., 4
Pearson K., 40
Piatetsky-Shapiro G., 6, 49
Playfair W., 41
Poosala V., 45, 51, 53

Raatikained K. E. E., 49
Robyn D. L., 40
Rothnie J. B. Jr., 2
Roussopoulos N., 27

Schuegraf E. J., 26
Selinger P. G., 25
Shanmugasundaram J., 27
Siler K. F., 26
Sun W., 26
Swami A., 12

Vander Zanden B. T., 4, 51
Vitter J. S., 37, 60

Wang M., 37
Weikum G., 57
Whang K.-Y., 3, 7
Wiederhold G., 3, 7
Wong E., 12

Yao S. B., 3
Young G., 53

Zipf G. K., 25