Histogram-Based Compression of Databases and Data Cubes

Alfredo Cuzzocrea
University of Calabria, Italy

INTRODUCTION

Histograms have been extensively studied and applied in the context of *Selectivity Estimation* (Kooi, 1980; Muralikrishna & DeWitt, 1998; Piatetsky-Shapiro et al., 1984; Poosala et al., 1996; Poosala, 1997), and are effectively implemented in commercial systems (e.g., Oracle Database, IBM DB2 Universal Database, Microsoft SQL Server) to **query optimization** purposes. In statistical databases (Malvestuto, 1993; Shoshani, 1997), histograms represent a method for approximating **probability distributions**. They have also been used in data mining activities, intrusion detection systems, scientific databases, that is, in all those applications which (*i*) operate on huge numbers of detailed records, (*ii*) extract useful knowledge only from condensed information consisting of summary data, (*iii*) but are not usually concerned with detailed information. Indeed, histograms can reach a surprising efficiency and effectiveness in approximating the actual distributions of data starting from **summarized information**. This has led the research community to investigate the use of histograms in the fields of database management systems (Acharya et al., 1999; Bruno et al., 2001; Gunopulos et al., 2000; Ioannidis & Poosala, 1999; Kooi, 1980; Muralikrishna & DeWitt, 1998; Piatetsky-Shapiro et al., 1984; Poosala, 1997; Poosala & Ioannidis, 1997), *online analytical processing* (OLAP) systems (Buccafurri et al., 2003; Cuzzocrea, 2005a; Cuzzocrea & Wang, 2007; Furfaro et al., 2005; Poosala & Ganti, 1999), and data stream management systems (Guha et al., 2001; Guha et al., 2002; Thaper et al., 2002), where, specifically, compressing data is mandatory in order to obtain fast answers and manage the endless arrival of new information, as no bound can be given to the amount of information which can be received.

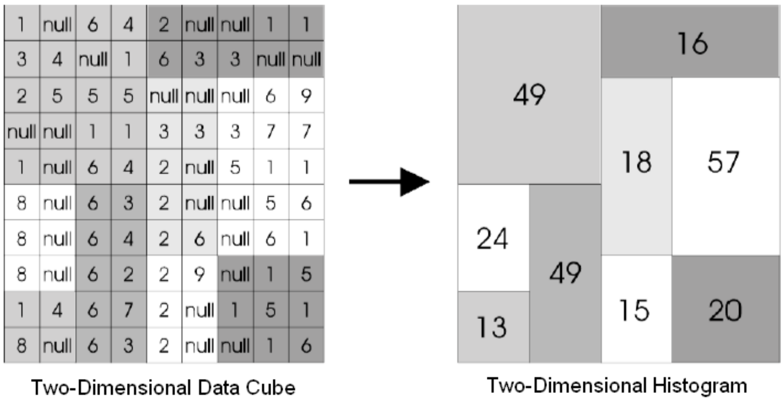
Histograms are data structures obtained by partitioning a **data distribution** (or, equally, a data domain) into a number of mutually disjoint blocks, called ***buckets***, and then storing, for each bucket, some **aggregate information** of the corresponding range of values, like the sum of values in that range (i.e., applying the SQL aggregate operator SUM), or the number of occurrences (i.e., applying the SQL aggregate operator COUNT), such that this information retains a certain "summarizing content."

Figure 1 shows an instance of a histogram built on a two-dimensional **data cube** (left-side of the figure), represented as a matrix. The corresponding (two-dimensional) histogram (right-side of the figure) is obtained by (*i*) partitioning the matrix into some rectangular buckets which do not overlap, and (*ii*) storing for each so-obtained bucket the sum of the measure attributes it contains.

Histograms are widely used to support two kinds of applications: (*i*) selectivity estimation inside *Query Optimizers* of DBMS, as highlighted before, and (*ii*) *approximate query answering* against databases and data cubes. In the former case, the data distribution to be compressed consists of the frequencies of values of attributes in a relation (it should be noted that, in this vest, histograms are mainly used within the core layer of DBMS, thus dealing with databases properly). In the latter case, the data distribution to be compressed consists of the data items of the target domain (i.e., a database or a data cube) directly, and the goal is to provide fast and approximate answers to resource-intensive queries instead of waiting-for time-consuming exact evaluations of queries. To this end, a widely-accepted idea is that of evaluating (with some approximation) queries against ***synopsis data structures*** (i.e., succinct, compressed representations of original data) computed over input data structures (i.e., a database or a data cube) instead of the same input data structures. Histograms are a very-popular class of synopsis data structures, so that they have been extensively used in the context of approximate query answering techniques. Some relevant experiences concerning this utilization of histograms are represented by the work of Ioannidis and Poosala (Ioannidis & Poosala, 1999), that propose using histograms to provide approximate answers to set-valued queries, and the work of Poosala and Ganti (Poosala & Ganti, 1999), that propose using histograms to provide approximate answers to *range-queries* (Ho et al., 1997) in OLAP.

In both utilizations, a relevant problem is how to reconstruct the original data distribution form the compressed one. In turn, this derives from the fact that the original data distribution summarized within a bucket cannot be reconstructed exactly, but can be approximated using some estimation strategies, like *continuous value assumption* (CVA) (Colliat, 1996) or *uniform spread assumption* (USA) (Poosala et al., 1996). For a given storage space reduction, the problem of determining the "best" histogram (i.e., the histogram which minimizes the approximation of reconstructing the original content of ranges corresponding to buckets) is crucial. Indeed,

Figure 1. A two-dimensional data cube and its corresponding two-dimensional histogram



1	null	6	4	2	null	null	1	1
3	4	null	1	6	3	3	null	null
2	5	5	5	null	null	null	6	9
null	null	1	1	3	3	3	7	7
1	null	6	4	2	null	5	1	1
8	null	6	3	2	null	null	5	6
8	null	6	4	2	6	null	6	1
8	null	6	2	2	9	null	1	5
1	4	6	7	2	null	1	5	1
8	null	6	3	2	null	null	1	6

different partitions lead to dramatically different errors in reconstructing the original data distribution, especially for *skewed* (i.e., asymmetric) data. This issue has been investigated for some decades, and a large number of techniques for arranging histograms have been proposed (Buccafurri et al., 2003; Christodoulakis, 1984; Donjerkovic et al., 1999; Ioannidis & Poosala, 1995; Jagadish et al., 2001). The aim of every partition technique is to build a histogram whose buckets contain values with "small" differences, so that one can estimate a range query inside a bucket assuming that data distribution is uniform, thus successfully exploiting linear interpolation. Indeed, finding the optimal solution to this problem in multiple dimensions is NP-hard (Muthukrishnan et al., 1999). Several techniques and heuristics have been proposed to find sub-optimal solutions with provable quality guarantees (Jagadish et al., 1998; Gilbert et al., 2001). These guarantees regard the "distance" of the provided solution from the optimal one, but do not provide any measure of the approximation of each estimated answer to a range-query.

Another important, more recent utilization of histograms concerns with the *data visualization problem*, where the data compression paradigm is intended as a solution to aid the visualization of complex and multidimensional domains. This is a quite-unexplored line of research: pioneeristic works can be found in the DIVE-ON (Ammoura et al., 2001) and *Polaris* (Stolte et al., 2002) projects, whereas recent works can be found in (Cuzzocrea et al., 2007).

In this chapter, we survey several state-of-the-art histogram-based techniques for compressing databases and data cubes, ranging from one-dimensional to multidimensional data domains. Specifically, we highlight similarity and differences existing among the investigated techniques, and put-in-evidence how proposals have evolved over time towards more and more sophisticated and very-efficient solutions, beyond early experiences focused on selectivity estimation issues within the core layer of DBMS.

BACKGROUND

A *database* D is a tuple D = ⟨W,I,F⟩ such that (*i*) W is the *schema* of D; (*ii*) I is the *instance* of D, that is, its realization in terms of collections of tuples adhering to W; (*iii*) F is the collection of *functional dependencies* defined over W. In turn, W is a collection of *relation schemas* W = {T_0, T_1,…, T_P}, with $P = |W| – 1$, such that T_i, with $0 \leq i \leq P$, is defined as a tuple $T_i = \langle K, A_{i,0}, A_{i,1},…, A_{i,G}\rangle$, with $G = |T_i| – 1$, such that K is the *key* of T_i, and $A_{i,j}$ is the *j*th *attribute* of T_i. A functional dependence is expressed as a *logical rule* over attributes of T_i. The instance of a relation scheme T_i is named as *relation*, and denoted by R_i.

A *data cube* L is a tuple L = ⟨C,J,H,M⟩, such that: (*i*) C is the data domain of L containing (OLAP) *data cells*, which are the basic SQL aggregations of L computed against the relational data source S alimenting L; (*ii*) J is the set of *dimensions* of L, that is, the *functional attributes* (of S) with respect to which the underlying OLAP analysis is defined (in other words, J is the set of attributes with respect to which relational tuples in S are aggregated); (*iii*) H is the set of *hierarchies* related to the dimensions of L, that is, hierarchical representations of the functional attributes shaped-in-the-form-of generic trees; (*iv*) M is the set of *measures* of L, that is, the *attributes of interest* (of S) for the underlying OLAP analysis (in other words, M is the set of attributes with respect to which SQL aggregations stored in data cells of L are computed).