rK-Hist: An R-Tree Based Histogram for Multi-Dimensional Selectivity Estimation

John Alexander Lopez

A Thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fullfilment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

September 2007

Your file Votre référence
ISBN: 978-0-494-34636-5
Our file Notre référence
ISBN: 978-0-494-34636-5

ABSTRACT

rK-Hist: An R-Tree Based Histogram for Multi-Dimensional Selectivity Estimation

John Alexander Lopez

Database query engines typically rely upon query size estimators in order to evaluate the potential cost of alternate query plans. In multi-dimensional database systems, such as those typically found in large data warehousing environments, these selectivity estimators often take the form of multi-dimensional histograms. But while single dimensional histograms have proven to be quite accurate, even in the presence of data skew, the multi-dimensional variations have generally been far less reliable.

In this thesis, we present a new histogram model that is based upon an r-tree space partitioning. The localization of the r-tree boxes is in turn controlled by a Hilbert space filling curve, while a series of efficient area equalization heuristics restructures the initial boxes to provide improved bucket representation. The proposed histogram works on an existing multi-processor platform that targets the relational database model (ROLAP).

Experimental results on real and synthetic data demonstrate significantly improved estimation accuracy relative to state of the art alternatives, as well as superior consistency across a variety of record distributions.

ACKNOWLEDGEMENTS

It has been a long trek. As every trek, it was demanding and sometimes exhausting but at the same time enjoyable, exciting and satisfying. On the walk, I was fortunate to meet a variety of wonderful people. Some of them have become close at a personal level as well. This is an attempt to show my appreciation for these people who directly or indirectly affected the course of this trek.

I am and always will be very grateful to Dr. Todd Eavis for giving me the opportunity to join the Sidera group and to teach me everything I know about research as a way of life. Besides explaining to me numerous concepts, sharing his style of thinking and his scientific interests, being a patient listener, and an outstanding mentor, his words were a great support when domestic and professional problems came up. I truly feel blessed to have had him as my supervisor.

I wish to express my sincere gratitude to my ex-manager and close friend Anna Mallikarjunan, not only for being flexible and understanding regarding my working hours when she was the lead of our "dream team", but also for our great discussions, for introducing me to great literature and writers, and for her words of encouragement.

Many thanks also to all my friends for all the cheer and fun they brought in life, for supporting me in numerous ways, and never stopped asking: "When are you going to finish?".

My close friends Khaled AbdelHay, Ahmad Dhaini, Gerardo Berbeglia, David Sevilla, David Cueva and Dania El-Khechen contributed towards this effort in ways I find difficult to describe. All of them with their own inimitable and unique personalities, helped eliminate drudgery from work and life in general. With some of them, late night strolls, tea/coffee/beer sessions (and quite a few shawarma plates) followed by

"profound" discussions, preparation of the famous Friday/Saturday parties, reading club meetings, soccer and hockey game sessions were the activities which helped me keep going and made life enjoyable.

Finally, I would like to give special thanks to my wife, Angely Jamis, whose patient love and support enabled me to complete this work. She not only helped me making some graphs and figures but also stayed at my side and put up with me when the trek became more strenuous. Gracias Amor.

To Lolita, Luz Alba and Angelyto for being the blessing they are in my life

Table of Contents

List of Figures

Chapter 1

Introduction

As database systems have grown in size, so too has the need to efficiently produce accurate approximations of the underlying data sets. In particular, the estimates of large data distributions have been utilized in the database context to support both approximate query answering and selectivity estimation. Very early DBMS platforms provided such functionality through crude statistical estimates that essentially treated the tuple space as a single uniformly distributed partition. However, with the introduction of histogram partitioning techniques [44], new tools for effective estimation emerged.

The majority of the early research focused on partitioning in single dimensions. For inherently multi-dimensional spaces, the initial approach involved the simple integration of a series of one-dimensional histograms. However, the underlying *attribute value independence assumption* [12] tends to produce extremely poor estimates of joint data distributions. Consequently, several noteworthy attempts have been made to produce true multi-dimensional histograms that more accurately reflect the natural patterns of clustering and skew that occur in such spaces [48, 53, 27]. We note that the problem of creating histograms for multi-dimensional spaces is quite challenging

since it was shown in [49] that optimal splitting even in two dimensions is NP-hard.

Among the plethora of histograms that have been produced during the last ten years, three methods in particular have proven over time to be the most efficient and most widely utilized. *hTree* was described in [48] and is considered the first attempt to support multi-dimensional selectivity estimation. Here, the model divides the d-dimensional space recursively into half spaces by using a boundary value and a different dimension each time. The dimension and the boundary value change at the beginning of each iteration. The β buckets built by *hTree* are non-overlapping d-dimensional rectangular regions. The *mHist* histogram attempts to improve upon the quality of partitioning by recursively dividing the space on the dimensions that are judged to most benefit from a split (in terms of density) [53]. While both mHist and the hTree produce "regular" grid patterns and non-overlapping rectangular regions, the *genHist* histogram produces irregular bucket patterns by iteratively identifying high density regions and subsequently permitting bucket overlap [27]. Unfortunately, it has also been shown to be quite expensive to generate.

Even though the previous methods were shown to produce very accurate approximations, further research, presented in this thesis, demonstrates that there are multiple datasets where their estimates are quite poor, generating normalized errors that range from 20% to 40%, something that makes them inadequate for use in query optimization.

1.1 Primary Research Contributions

In general, the focus of multi-dimensional methods is the identification of regions of close to equivalent point density. These hyper-rectangular areas are then divided

into a small number of memory resident *buckets*. Unlike the single dimensional case, there is no absolute, locality–preserving order on a multi-dimensional point space, so each histogram model employs heuristic techniques to identify regions of approximate uniformity. Regardless of the technique, however, the uniformity measure is effectively a simple density calculation in which the points within a bucket are assumed to be evenly distributed. We refer to this as the *uniform spread assumption* [54].

In fact, density can be a relatively poor measure of point distribution, a problem that is exacerbated as dimensionality increases. In this thesis, we present instead a new metric called k-uniformity (kU) that more accurately measures the quality of tuple distribution. The kU metric is integrated with a partitioning model often employed in multi-dimensional indexing environments. Specifically, we build upon the notion of multi-dimensional r-tree partitioning, in which the distribution of blocks is governed by the Hilbert space filling curve. By "piggy-backing" on top of an existing multi-dimensional ROLAP indexing model (i.e., RCUBE)[21], we are able to provide effective selectivity estimation in conjunction with powerful indexing functionality in multi-dimensional settings.

The kU metric is integrated with the indexing framework using a mechanism known as the *Sliding Window Algorithm* (SWA). In summary, SWA builds an initial set of buckets by using a data structure we call the "naive" r-tree histogram. The construction method generates buckets by exploiting a Hilbert sort of the data, working upward from the leaf level of the r-tree index. The leaf level of the Hilbert packed r-tree provides us with a set of β rectangular bounding boxes that each enclose regions of spatially related points. We integrate these β boxes into α *hyper-buckets* that constitute the initial histogram on which the final solution is based.

Because the default ordering of the Hilbert curve may produce less than optimal bucket partitions, we extend the basic technique with an optimization component. We use a quality measure (the kU metric) to identify a percentage of the total number of buckets with the worst distributions. Then, the algorithm scans each "poorly distributed" hyper-bucket by using a heuristic method that seeks to re-partition it to produce a more uniform point distribution. The final result of SWA is a multi-dimensional histogram containing buckets with high uniformity and greatly reduced "dead space".

Finally, we note that the SWA was built on top of an existing multi-processor platform that targets the relational database model (ROLAP). The platform was presented, implemented and tested in [21] and consitutes the architectural foundation for the present thesis.

1.2 Review Of Experimental Results

Experimental results demonstrate that our new r-tree/kU histogram, rK-Hist, produces estimation errors that significantly improve upon the current state of the art. In fact, in skewed spaces, we see estimation errors as low as 2%-7% in two to four dimensions. Moreover, results are consistently impressive as dimension count increases, suggesting that our new methods have broader applicability than existing techniques.

Results on the effect of the storage space on histogram accuracy demonstrate the excellent scalability of our approach. As the number of buckets increases, the average absolute error decreases almost in the same proportion.

Finally, experimental results on real data sets demonstrate the applicability of our new r-tree/kU histogram, rK-Hist, to real-life problems.

1.3 A Look Ahead

The thesis is organized as follows. Chapter 2 provides an overview of Data warehousing and Online Analytical Processing, including such things as fundamental OLAP operations and server architectures. The chapter also reviews the general field of database histograms and describes a number of the algorithms that have been designed for their efficient construction.

The succeeding chapters present the core contributions of this thesis, including the proposed algorithms, the implementation issues, and the experimental results. Chapter 3 focuses on the construction of the different multi-dimensional histograms that have given rise to rK-Hist and describes the algorithms and data structures used for their implementation. The experimental results and their analysis are discussed in Chapter 4. Finally, in Chapter 5, we present the conclusions and references to possible future work.

Chapter 2

Background

2.1 Introduction

Data warehousing and on-line analytical processing (OLAP) systems are progressively playing a more vital role in the growth and success of corporate organizations [25]. Today, a large number of commercial products and services are available, not to mention the multiple developments that are underway in both academia and the open source community. In this chapter, we review the fundamental principles and terminology relevant to the general OLAP domain, as well as providing a more detailed examination of the field of database histograms.

The chapter is organized as follows. In Section 2.2, we provide an overview of Data Warehousing and *on-line analytical processing* or OLAP, with an emphasis on their new requirements. In section 2.3, we present an introduction to *Parallel OLAP* and discuss recent developments in this field. Section 2.4 discusses query optimization and presents some approaches that have proven beneficial in different commercial DBMS. In section 2.5, we formally define the histogram, discuss one-dimensional and multi-dimensional implementations and provide a brief overview of a taxonomy that captures all previously proposed types of histograms and defines a

mechanism to derive new classes by combining elements in effective ways.

2.2 Data Warehousing and OLAP

Both data warehousing and OLAP are components of a larger model known as Decision Support Systems (DSS). In short, DSS are a collection of interactive computer-based applications that support decision-making activities. Ultimately, they empower decision makers with the ability to manipulate and analyze efficiently data stored in large repositories in order to solve problems and make effective decisions. These kinds of applications have become so essential and common among the companies striving for competitive edge that even the *Transaction Processing Performance Council* (TPC) [15], a non-profit corporation founded to define transaction processing and database benchmarks, has created a specific benchmark, known as TPC-H, intended to measure performance by simulating a decision support application.

Various software companies have implemented DSS systems as extensions of the functionality of a number of existing Database Management Systems (DBMS). This is one of the reasons why a large number of dissimilar DSS implementations can be found on the market. As a way to differentiate these implementations according to their functionality, level of detail, and depth of the analysis, we describe the three primary DSS models, including OLAP, which is the focus of our current research [21].

- **Information Processing**. This is considered the most superficial of all of the three models. Here, the main goal is to retrieve basic information with little analysis. Raw data, tabular formats, sorting operations, use of simple aggregate functions and basic analysis are the features that characterize the queries and reports at this level.

- **OLAP**. The basic reporting capabilities offered by Information Processing systems are extended by OLAP. Multidimensional analysis, new operations such as *drill-down*, *roll-up* and *pivot*, enhanced forms of visualization, aggregated data with more complex operations such as average, standard deviations and ranking, are common features that make these systems a more appealing and useful tool to decision makers.

- **Data Mining**. Data Mining is often considered the final stage in the search for knowledge on databases. As the term implies, the idea is to mine or excavate large repositories of data with the hope of finding precious "gems" (i.e., knowledge). This knowledge comes in the form of unknown patterns that are not easily extracted by using OLAP techniques alone. Some of the widely used techniques include *classification* (categorization of novel items), *association* (identification of patterns and trends), and *cluster analysis* (identification of data groupings via multi-attribute similarity). It is worth mentioning that Data Mining is part of a broader process called Knowledge Discovery in Databases or KDD. Figure 2.1 presents the components of a KDD system.

2.2.1 Defining OLAP

Most large companies have accumulated huge repositories of data that can be used to facilitate the extraction of information and knowledge in the form of patterns and trends. This extraction and analysis is referred to as OLAP (On-Line Analytical Processing). In a sense, OLAP can be seen as a technique of structuring data in order to allow multi-dimensional analysis.

It is important to note that OLAP places some different requirements on database

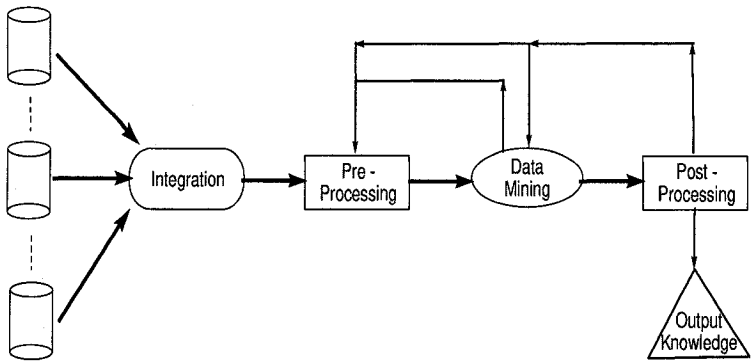


Figure 2.1: Architecture of a KDD System (Knowledge Discovery in Databases).

technology compared to traditional *on-line transaction processing* applications or OLTP. In contrast to OLTP systems, in which transactions frequently access few records, OLAP systems require the examination of large portions of data. The reason for this is that the queries that are resolved against the database often require the use of aggregate functions (such as SUM, AVG, COUNT, COUNT (*), MAX, and MIN). One particularly important feature of OLAP systems is that queries can be extremely time consuming given the amount of data they have to retrieve and analyze.

We can also describe OLAP in terms of the following five core operations and the functionality they offer. We note that these operations are often associated with a data model known as the cube, the details of which are presented in section 2.2.3

- **Roll-up**. This operation aggregates data from a lower or detailed level to a higher level summary. For instance, using the "date" dimension we could aggregate "quarter total sales" from "monthly total sales".

- **Drill Down**. This is roll-up's inverse operation. It shows the finer grain of a given dimension. For example, using the "location" dimension we can go from

"country total sales" to "state total sales".

- **Slice**. Here, the idea is to extract a "plane" of the original cube corresponding to a single attribute of a specific dimension. No aggregation is performed.

- **Dice**. The slice is an operation quite similar to the *slice*. It extracts a subset of data from the original cube by performing a selection on more than one dimension.

- **Pivot**. This operation reorients the cube allowing different views of the data.

2.2.2 Defining the Data Warehouse

In [35], Bill Inmon, considered the father of the contemporary data warehouse, defined the data warehouse as "*a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process*". In other words, a data warehouse can be seen as a huge data repository that holds an organization's historical data in tailored schemas in order to support the decision-making process. The analysis of trends and the discovery of patterns are two of the major advantages achieved through the use of this technology.

Physical Architectures

The data warehouse architecture is a multi-tier system consisting of several interconnected layers. In Figure 2.2, we can see the three main layers: Data warehouse, OLAP server, and client applications. The data warehouse itself is loaded using a process known as Extract, Transform, and Load. This ETL phase essentially pulls data from a series of operational sources, then connects and harmonizes the independent input streams before loading the final consistent data into the data warehouse.

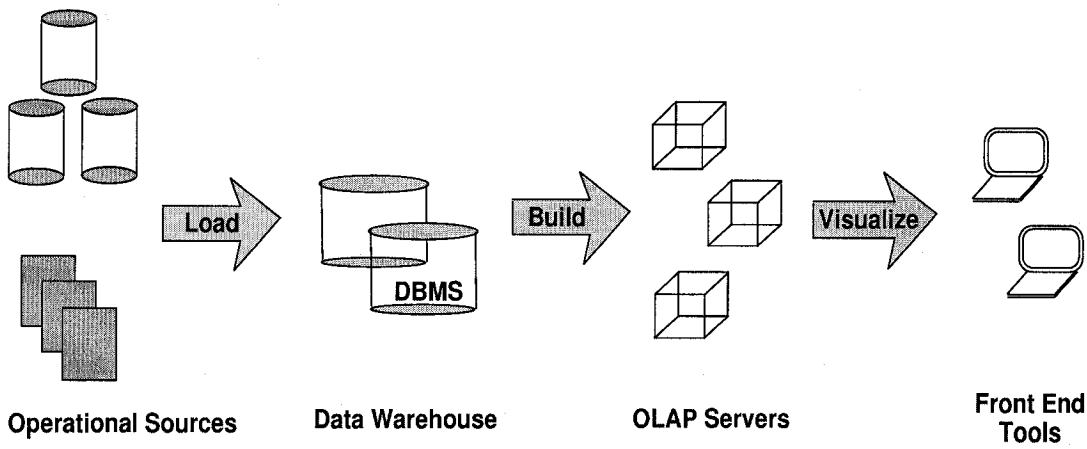


Figure 2.2: Physical Data Warehouse Architecture.

After the ETL process is applied to the different OLTP data sources, the input of the data warehouse (i.e., clean, correct and consistent data) is obtained. Then, the OLAP server provides the tools to collect, manage, process and present multidimensional data for analysis. In fact, depending upon the type of data structures used by the server and the storage mode chosen to hold data and pre-aggregations, there are actually three different processing models: ROLAP, MOLAP or HOLAP (discussed in more detail later). Finally, the last layer represents the various applications (i.e., querying, reporting and analysis tools) which interact with the OLAP server to provide the client with a graphical view of the data.

The data warehouse physical architecture can be modeled as *centralized*, *federated* or *tiered*. The use of any model depends on the organization's objectives and motivations. The *centralized model* is a single physical warehouse where all the information is stored. Users query the data warehouse directly. This model is by far the simplest architecture to design and implement.

The *federated model* is virtually represented as a central data warehouse, yet it is

physically distributed in multiple partitions - small data warehouses - depending on the type of operation or business (e.g., financing, manufacturing) or on the physical locations of the organization (e.g., by city or state). Users have the impression that they query only one "central" data warehouse; however, they are actually querying one of the many warehouses or some combination of them at the same time.

The *tiered model* is a combination of both centralized and federated models. It is composed of a centralized physical data warehouse containing all the detail and one or more layers of local data marts. The local data marts extract the data from the central data warehouse and store it as aggregated or sumarized information. This approach generally provides better performance — at the cost of greater complexity — due to the distribution and aggregation techniques it implements.

Conceptual Models

A conceptual data model describes the semantics of an organization by exposing the concepts or *entities* that are relevant to the system, and the associations or *relationships* between those concepts. In practice, the construction of this model consumes a considerable amount of time and relies heavily on user interaction. A conceptual model is vital for the success of the data warehouse implementation because it provides a general overview of the organization and delivers the insight needed to understand the data requirements.

Logical Models

At the core of OLAP and data warehouse applications are two specialized schemas that make multi-dimensional analysis possible by using relational database models that relax the rules of data normalization presented by E. Codd in [13]. These logical

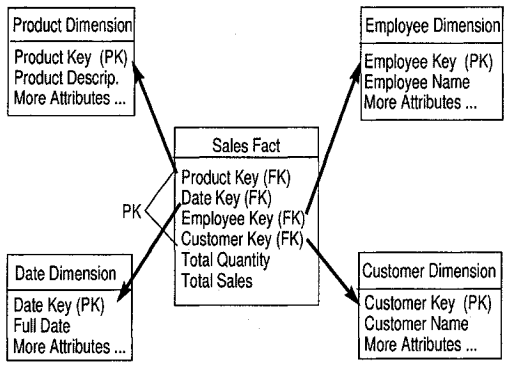


Figure 2.3: Star Schema.

representations are known as *Star* and *Snowflake* schemas.

- **Star Schema.** The star schema is the most common representation used to build up a data warehouse. The name is derived from its star shape. It consists of a central table called the *fact table* that is in turn connected to multiple *dimension tables.* The dimension tables are denormalized in the sense that they maximize query performance by accepting a certain degree of data redundancy. The fact table usually models a business process and is made up of a composite primary key — containing the keys of each of the associated dimension tables — and all the numeric measurements or *facts.* Figure 2.3 presents an example of this type of representation.

- **Snowflake Schema.** The snowflake schema can be seen as a variation of a star schema. This representation is composed of a central fact table that is connected to multiple *normalized* dimension tables. These dimension tables are in turn connected to additional tables that provide further information. For instance, a Product dimension table in a star schema might be normalized into a Products table, a Category table, and a Vendor table.

Physical Models: Storage Modes

In large-scale data warehouses, where the design of OLAP cubes is critical to provide maximum performance, it is crucial to make decisions about the storage mode and the level of aggregation. In the OLAP context there are three primary storage models: MOLAP, ROLAP and HOLAP [9].

MOLAP: Multidimensional OLAP

MOLAP is the most commercially successful data model for online analytical processing. It stores the data (i.e., base data and aggregations), in optimized multi-dimensional arrays. The arrays represent all possible combinations of data that can be created from the set of dimensions. Each possible value will be held in a cell that can be accessed directly (i.e., natural indexing). For this reason, MOLAP is considered the solution with the fastest response time on OLAP queries.

Advantages

- Fast query performance due to the natural indexing along each axis provided by the array model.
- Very compact for low dimension data sets.

Disadvantages

- The computing of the cuboids (i.e., aggregated data) can be time consuming especially when there is a huge amount of data.
- When dimensions with high cardinality exist, scalability becomes an issue. When using an array structure, we require a full materialization of the array.

Therefore, the cube has to represent all the cells in the space even if a value doesn't exist. This issue has been partially addressed by using sparse array compression techniques.

- Changes on the schema require rebuilding the complete array structure.

ROLAP: Relational OLAP

ROLAP is a form of online analytical processing that stores the data (i.e., base data and aggregations), in relational structures. Since the components of this architecture are conventional database tables, we can use basic SQL statements to retrieve the data and to perform any additional operation required by the end users. All the operations required to generate the sub-views can also be carried out by traditional RDBMS. When compared to MOLAP, this type of storage engine offers more scalable solutions since it only stores existing values (i.e., ignoring empty cells).

Advantages

- Since the data is stored in a standard relational database, it can be queried by any SQL reporting tool, not necessarily a special purpose OLAP tool.
- More scalable when handling huge amounts of data because it only stores existing values.

Disadvantages

- Explicit indexing is required on the tables to find the required records.
- Slow performance compared to MOLAP.
- It is limited by SQL capabilities.

HOLAP: Hybrid OLAP

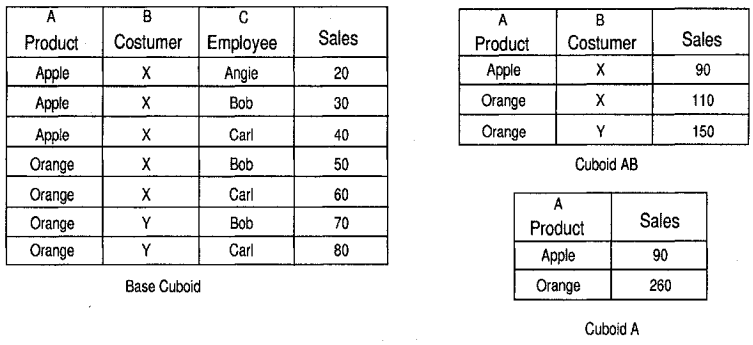
HOLAP is a combination of both ROLAP and MOLAP storage modes. It allows the distribution of data and aggregations on ROLAP and MOLAP structures (i.e., some tables are stored in MOLAP form while others are stored in ROLAP).

2.2.3 Defining The Data Cube

A data cube can be seen as a multi-dimensional construct that allows us to see the data contained in the data warehouse from different perspectives. In this section, we present the basic terminology that will be used throughout the thesis to describe the cube structure.

We begin by noting that an OLAP environment consists of a group of dimensions. A dimension is the categorization of some element in an organization with which we can associate performance. For instance, you can track your sales data against employee, product or customer in a period of time. Dimensions can also be described as collections of attributes, which are bound to one or more columns in a table or view. There are two types of attributes. *Feature or regular* attributes represent the concepts by which you want to classify/group your data. Some examples would be customer id, product id and store code. *Measures attributes*, on the other hand, are the calculated values associated with the elements represented by the regular dimensions. Some examples include total dollar sales, quantity sold, sku's on-hand and cost.

A d-dimensional data warehouse is associated with 2^d views (sometimes referred to as the view *Power Set*). In OLAP terminology, views are also known as *cuboids*



A Product	B Costumer	C Employee	Sales
Apple	X	Angie	20
Apple	X	Bob	30
Apple	X	Carl	40
Orange	X	Bob	50
Orange	X	Carl	60
Orange	Y	Bob	70
Orange	Y	Carl	80

Base Cuboid

A Product	B Costumer	Sales
Apple	X	90
Orange	X	110
Orange	Y	150

Cuboid AB

A Product	Sales
Apple	90
Orange	260

Cuboid A

Figure 2.4: An example of a "base cuboid" and two of its aggregations (i.e., subviews or cuboids).

or *group-bys*. Each view or cuboid is the combination (group-by) of a subset of dimensions and represents the aggregation of all tuples on those dimensions. Figure 2.4 presents a base cuboid and two sub-cubes at different levels of *granularity*. We say that the view "A" is of coarser granularity than the view "AB" because the former represents a higher level of aggregation.

Given the previous definitions, we will continue to describe a data cube as a multi-dimensional construct composed of 2^d views that includes the base cuboid — the finest granularity view containing the full complement of d dimensions — and all the possible aggregations along one or more of the d dimensions. Figure 2.5 illustrates a *lattice*, a modeling structure frequently used to represent the relationship between the 2^d cuboids [63, 21]. Although the term "cube" comes from the three-dimensioned geometric object, OLAP cubes can have 10 or more dimensions.

Until now, we have considered the data cube as a conceptual model. However, the data cube is also a physical construct that is stored either in MOLAP or ROLAP structures according to the selected design. Regardless the type of storage mode chosen, the cube structure is drawn from a Star Schema similar to the one presented

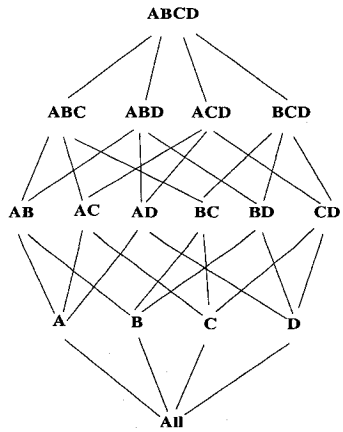


Figure 2.5: The data cube lattice depicts the relationships between all the 2^d views in a given d-dimensional space. The boundaries of the lattice correspond to the base cuboid and the "all" node representing the aggregation of all records — one tuple.

in figure 2.3 and described in Section 2.2.2.

2.3 The Sidera Platform

This thesis was conceived as part of a larger data warehousing project called Sidera, a project that explores the design and development of parallel OLAP server technology. Sidera is closely associated with cgmLab [7], a joint effort between Carleton University (Ottawa), Concordia University (Montreal), Dalhousie University (Halifax), and Griffith University (Brisbane). In short, the cgmLab is a distributed research lab which focuses on the development of parallel applications. They pursue research projects that explore the application of parallel computing to data and computationally intensive problems.

One of the most recent research projects undertaken by the group has been the

cgmCUBE project, a pioneering multi-year effort to design and implement a multi-processor platform for data cube generation that targets ROLAP environments. Professor Todd Eavis is co-investigator in this effort and his work has grown to be the framework on which a set of proposed algorithms and techniques, including ours, have been built.

2.3.1 Important Contributions

Eavis' work [21, 17, 11, 10, 19, 18] has been focused on the construction, maintenance, and indexing of data cubes. Since this work represents the foundation of the design and implementation of our algorithms, we will briefly present some of the research that is relevant to the present thesis.

- Parallel Data Cube Computation. Multiple algorithms have been proposed and implemented for the parallelization of the data cube. The most important achievement in this area has been the introduction of a general framework by which a number of existing sequential algorithms were implemented in a fully distributed environment. The novel approach employs a sophisticated model to partition the workload in advance allowing view construction to be fully localized.

  In addition to the algorithmic contributions, extensive evaluation and profiling has been performed. Sorting optimizations (pointer comparisons, exploitation of linear time Counting Sorts, etc.), elimination of redundant data copying, and a rigorously developed Parallel Cost Model, led to an order of magnitude performance improvement in the core system.

- Parallel Partial Cubes. The data cube operator supports the construction of 2^d

individual views (where "d" is the number of dimensions or attributes). The exponential growth in the number of sub-views makes this operator impractical in high-dimensional spaces. As a result, and in order to cope with this problem, a suite of algorithms for partial cube construction has been proposed and implemented. Based upon a greedy model, the algorithms consider various computational trade-offs in order to identify an efficient model for the generation of the selected set.

- Parallel Multi-dimensional Indexing. A parallel indexing model based on R-trees was proposed and implemented. The new technique consists of a combination of Hilbert-curve ordering and round robin disk striping for data partitioning. Furthermore, a distributed query engine was presented which includes the implementation of distributed query resolution mechanisms for complete and partial cubes.

Given that our focus is on the design and construction of a multi-dimensional histogram for selectivity estimation, we regard the *RCUBE* indexing model as the major contribution, indispensable to the success of our proposed technique.

2.3.2 Multi-Dimensional ROLAP Indexing: RCUBE

RCUBE is a distributed multi-dimensional ROLAP indexing scheme which has proven to be efficient for spatial searches in high dimensions and scalable in terms of data sizes, dimensions and number of processors.

The RCUBE indexing scheme makes use of local partial R-tree indexes built on each processor to resolve a portion of the submitted query. A similar approach was used in Master-Client R-trees [57]. However, unlike the Master-Client technique, the

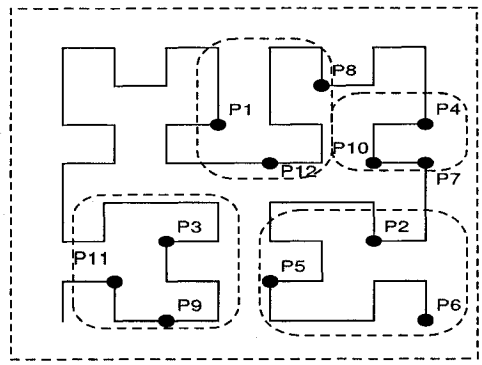


Figure 2.6: Hilbert Curve Packing.

solution proposed in [18] by Eavis et al. does not include a global R-tree on the front-end. Instead, RCUBE uses short messages to pass the queries directly to each processor in the cluster, ensuring that intermediate results will remain distributed and available for further processing.

A very important difference between RCUBE and previous methods [28, 55] is that RCUBE applies a combination of Hilbert-curve sort ordering and round robin disk striping for data partitioning. Hilbert-curve orderings have been shown to be an effective tool for ordering data such that items that are close to each other in the original space (i.e., multi-dimensional space), are likely to be placed close to each other in the sorted order (i.e., linear space [22, 55]). The use of Hilbert-based ordering makes this technique appropriate for answering arbitrary range queries. Figure 2.6 illustrates a typical case of Hilbert packing. Note that each rectangle represents a single, physical disk block.

In addition to the application of this sorting approach (i.e., Hilbert sorting), for the first time introduced as part of a ROLAP indexing method, RCUBE implements a striping solution in order to balance the retrieval times across all available processors.

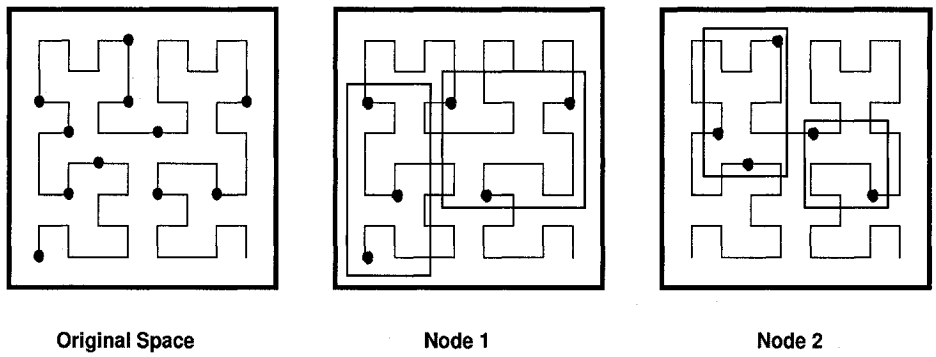


Figure 2.7: Striping data across two nodes. Note that every node contains two blocks with at most 3 records each.

Here, the idea is to stripe the Hilbert-curve ordered data in a round robin fashion such that successive records are sent to successive processors and then create local packed R-trees from the striped data. Figure 2.7 shows the effect of striping the original space across two processors.

In summary, there are a number of properties that make this method an ideal candidate for high-performance ROLAP environments:

- low communication volume.
- fully adapted to external memory (i.e., disks).
- incrementally maintainable.
- no shared disk required.
- scalable in terms of data sizes, dimensions, and number of processors.

2.3.3 The Current and Future Status for Sidera

In [21], Eavis established the foundation for the construction of a Parallel ROLAP Server, defining the starting point and creating the framework on which future work would be based.

Several components have been and are being developed in order to enlarge the original functionality. Some of the ongoing and conducted work includes:

- A Hilbert Space Compression Framework[16].
- A Framework for the Manipulation of OLAP Hierarchies[61].
- OLAP Caching.
- Partial Cube Generation.

2.4 Query Optimization

Every time a query is submitted to the database, the database management system (DBMS) has to determine the most efficient way to execute it. This task is performed by a component known as the *query optimizer*. Query optimization, despite its importance, has received little attention and other than the RCUBE, discussed in Section 2.3.2, no additional research had been conducted aiming to the improvement of the original query engine. This thesis investigates such an opportunity, with a particular emphasis upon selectivity estimation.

2.4.1 The Query Optimizer

In short, the job of the query optimizer is to produce a set of alternative query plans and then to choose what it considers the best option. A query plan is represented

by a set of coordinated execution steps. All plans are equivalent in terms of their final output but vary in their cost (i.e., the consumed resources such as CPU time, I/O cost, memory or a combination of these). Keep in mind that the alternative chosen by the optimizer might not be the optimal solution. This is because, given the complexity of the actual databases in terms of the number of logical/physical operators used to generate query plans and the numerous combinations that can be produced for a given query, the selection of "the best solution" could be more expensive (time consuming) than the time the query takes to execute. As Kalen Delaney states in [20]: "*The lowest estimated cost is not simply the lowest resource cost; the query optimizer chooses the plan that most quickly returns results to the user with a reasonable cost in resources.*"

There are two different approaches that have typically been used for the creation of query plans and the selection of the best alternative: rule-based (RBO) and cost-based (CBO) optimization[14]. The former chooses an execution plan based on the access paths available and a ranking of those access paths. It is an out-dated optimizer mode and most of the commercial vendors that used to work with this type of optimization have stopped supporting it. On the other hand, the latter, CBO, determines which execution plan is most efficient by considering available access paths and analyzing information based on statistics collected for the objects accessed by the issued query.

Finally, query optimization can also be defined as a search problem that in order to be solved requires a space of query plans, a cost estimation technique and an enumeration algorithm. In [8], Chaudhuri presents a description of these components, as follows.

- A space of plans (search space). The search space consists of a set of low-cost

plans whose construction depends on the set of available algebraic transformations and physical operators.

- A cost estimation technique. Here, the objective is to assign an estimated cost to each plan in the search space. This cost represents an estimation of the resources needed for the execution of the plan.

- An enumeration algorithm. This algorithm is responsible for choosing an inexpensive query execution plan by exploring the search space.

2.4.2 Selectivity Estimation

As we mentioned before, a query plan consists of a set of coordinated execution steps. Each execution step represents an operation that is carried out on a set of rows. This set of rows or data stream is known as a *relation.* The size of a relation represents perhaps the most important estimate calculated by the optimizer since it directly matches the number of disk blocks that have to be retrieved in order to answer a query. The more blocks the system retrieves the higher the cost of a query plan. Note that even though the sizes of the initial relations are known, the sizes of the intermediate or final relations are unknown.

The problem of estimating the size of a relation is known as *selectivity estimation.* Selectivity estimates are of great importance in query optimization because the optimizer uses the estimates to select appropriate access paths and to determine efficient join orders. This is a complex problem that requires the use of statistical information. Statistics refers to the collection and organization of numerical data that need to be kept in order to make generalizations. Some statistical data kept by different systems includes the number of rows per object (table or index), the number of physical pages

used per object and the data distribution on certain columns. Since many DBMS systems use **histograms** to hold the information that is required to estimate the selectivity of a relation, this will be the subject of our next section.

2.5 Histograms

2.5.1 Introduction

Merriam Webster on-line's dictionary defines a histogram as "*a representation of a frequency distribution by means of rectangles whose widths represent class intervals and whose areas are proportional to the corresponding frequencies*". Despite this definition, histograms have been proven useful even when disassociated from their usual visual representation and treated as purely mathematical objects that capture the distribution of a data set [37]. In this chapter, we present an introduction to histograms and discuss some of the topics that have been of interest throughout much of the research work done so far. We also show techniques to partition data distributions, indicate what data to store for each bucket, and present how to estimate an answer using the histograms.

2.5.2 Histogram Definitions

In this section, we present some definitions and terminology that will be useful throughout the reading of this thesis. This material is based upon the work first presented in [54, 37].

Motivation

In Section 2.4.1, we talked about the importance of statistical data in the decision-making process of a query optimizer. Data distributions are perhaps the most useful

statistical data a query optimizer can use in order to generate an optimal execution plan. Even though keeping these distributions is a great advantage for any database, the size of modern OLAP databases makes this large amount of data impractical to use for estimates. Histograms emerged as an approximation mechanism aimed at addressing this limitation.

Histograms have a long history of use for selectivity estimation within query optimization and more recently for query answering approximation [36, 44, 52, 38, 40, 54]. As a selectivity estimator mechanism, histograms are most frequently used to determine the utility of specific physical operators (e.g., access paths (indexes)) and to determine join orders. Although there are other techniques that can be used to approximate the selectivity of a query such as wavelets [64], table sampling [30], and discrete cosine transform[46], none of them has proven to be as efficient as histograms in terms of accuracy and resource consumption [2, 31].

Histograms have also been proposed as a mechanism for answering queries approximately. Here, they come into play as a way to reduce query response times when the precise answer is not necessary or early feedback is helpful. Our focus throughout this thesis will be on range query selectivity estimation.

Data Distributions

Consider a relation R with n attributes X_i ($i = 1...n$). The *domain* D_i of X_i is the set of all possible values of X_i. The *value set* V_i is the set of values of X_i that are actually present in R. Let $V_i = v_i(k) : 1 \leq k \leq D_i$, where $v_i(k) < v_i(j)$ when k < j. The *spread* $s_i(k)$ of $v_i(k)$ is defined as $s_i(k) = v_i(k+1)$ - $v_i(k)$, for $1 \leq k < D_i$. (The spread of the last value in D_i is 1, i.e., $s_i(D_i) = 1$.) The *frequency* $f_i(k)$ of v_i(k) is the number of tuples t $\in$ R with t.$X_i = v_i(k)$. The *area* $a_i(k)$ of $v_i(k)$ is defined as

$a_i(k) = f_i(k) \times s_i(k)$ .The *data distribution* of X_i is the set of pairs $T = (v_i(1), f_i(1))$, $(v_i(1), f_i(1)), \ldots, (v_i(D_i), f_i(D_i))$.

Histograms for Database Applications

A classical representation of a histogram in terms of database environments was provided by Viswanath Poosala et al. when the authors pointed out: "A histogram on attribute X is constructed by partitioning the data distribution τ into β (≥ 1) mutually disjoint subsets called *buckets* and approximating the frequencies and values in each bucket in some common fashion" [54]. In the same paper, the authors provided a taxonomy that captures all previously proposed types of histograms and defines a mechanism to derive new classes by combining elements in effective ways. According to this taxonomy, histograms are characterized by the following four orthogonal properties: *partition class*, *partition constraint*, *sort parameter* and *source parameter*.

2.5.3 Histogram Taxonomy

In this section, we discuss the available choices to instantiate the histogram properties mentioned in 2.5.2.

Partition class

This property specifies any constraint on the number of elements that can be assigned to a bucket. There are two classes of great importance: *serial* — which places no restrictions and *end-biased* — which requires that all but one of its buckets are singleton (i.e., they contain a single element of the data distribution).

Source Parameter and Sort Parameter

The sort parameter is calculated for each element in the data distribution and its value is derived from each attribute value and frequency. Some of the sort parameters that have been discussed in detail in the literature are *attribute value (V)*, *frequency (F)* and *area (A)*. Histogram buckets might be seen as a group of elements of the data distribution that are contiguous in the order of the *sorted parameter*.

On the other hand, the source parameter identifies the property of the data distribution that is the most critical and is used along with the *partition class*, in order to determine unique partitions. The most useful source parameters are *spread (S)*, *frequency (F)* and *area (A)*.

Partition Constraint

This property can be defined as a mathematical constraint on the source parameter and is used as a mechanism to distinguish a histogram within its partition class. *Equi-sum*, *v-optimal*, *maxdiff* and *compressed* are the most commonly used partition constraints and will be further defined below.

- **Equi-sum**. In this kind of histograms the sum of the source values in each bucket is approximately the same. If we have β buckets the sum of the source values in each bucket will be approximately equal to $1/\beta$ times the sum of all the source values in the histogram. The two first histograms proposed in the literature, equi-width[44] and equi-depth[52], are part of this partition constraint.

- **V-optimal Histograms**. This constraint partitions the data distribution so that the variance of source-parameter values is minimized within each bucket. In other words, the error in the approximation which is equal to $\sum_{i=1}^{\beta} p_i V_i$ is

minimized, where p_i is the number of entries in bucket b_i and V_i is the variance of the source values in the same bucket [38, 54].

V-optimal has been shown to be the most effective constraint for several selectivity estimation problems. However, its most significant downfall is the construction cost which is exponential in the number of source-parameter values. In [40], Jagadish et al. made a key contribution in this direction. They present a dynamic-programming based algorithm that computes optimal bucket boundaries in time that is quadratic in the number of source-parameter values and linear in the number of buckets.

- **Maxdiff Histograms**. As with v-optimal, the goal of this kind of histograms is to avoid grouping together in the same bucket different source-parameter values. Maxdiff tries to achieve this goal by sorting the data in source parameter order and placing boundaries between adjacent source-parameter values whose difference is among the largest (one of the β-1 largest diferences).

- **Compressed Histograms**. In a compressed histogram, singleton buckets are created to hold the largest source values. The rest of the values are partitioned as in an equi-sum histogram (e.g., equi-depth). This partition constraint can be considered an improvement over equi-depth since it gets exact information on largest values. Nevertheless, it has been proven less accurate than v-optimal and maxdiff.

Value Approximation and Frequency Approximation Within a Bucket

When it comes to (a) how to use the information available within each bucket in order to estimate answers for user queries and (b) what data to store for each bucket, a

key question concerns the kind of assumptions histograms make regarding value and frequency. These assumptions entail important design decisions since they implicitly define a balance between the number of buckets, the amount of information kept in each bucket, and any constraint regarding the total available space for a histogram.

- **Value Approximation**. Different approaches have been proposed in the literature for approximating the set of attribute values within a bucket. The most simple is the *continuous value assumption*, where the least amount of information is maintained (just the lowest and highest value in each bucket). In this assumption, all possible values in the domain that are in the range of the bucket (i.e., between the lowest value and highest value), are assumed to be present. If the domain is an uncountably infinite set, interpolation has to be used in order to estimate the contribution of a bucket to a query result.

  In [54], Poosala et al. present a more effective approach called *uniform spread assumption*. Under this assumption, attribute values are assumed to be placed at equal intervals between the boundaries of each bucket (i.e., between the lowest and highest values) — average spread. Unlike the continuous value assumption, the number of attribute values assumed is not given by the domain, but by the number of distinct attribute values inside the bucket. Therefore, this assumption requires the storage of the lowest and highest values along with the number of distinct attribute values in the bucket.

- **Frequency Approximation**. To date, almost all effort at approximating the frequency within a bucket has been concentrated on the *uniform distribution assumption*. Simply put, this assumption suggests that the frequencies of all elements held by each bucket are the same and equal to the average of the actual

frequencies. Among the few exceptions are a *linear spline-based* approximation [43] and a *4-level tree index* proposal that uses 32-bit information for each bucket for storing approximated cumulative frequencies [6].

2.5.4 One-Dimensional Histograms

Several different histograms have been proposed in the literature. In this section we present some well-known histograms that have been proven to be effective. Each of these will be identified by its partition class, partition constraint, sort parameter and source parameter by using the following notation presented in [54]: *p(s,u)*, where p denotes the partition constraint, s sort parameter and u source parameter. The most common way to differentiate one partition class from another is to add an identifier beside the name of the partition constraint. For instance, while v-optimal alludes to a serial class, v-optimal-end-biased denotes the end-biased class.

Equi-width: Equi-sum(V,S)

Equi-width histograms divide the value set into ranges of equal length. That means, the sum of the source-parameter (i.e., spreads) in each bucket are equal to $1/\beta$ times the maximum minus the minimum value that appears in the set value V. This kind of histograms represent an improvement over the *trivial histogram* — a single bucket histogram that is essentially equivalent to the uniform distribution assumption.

Equi-depth: Equi-sum(V,F)

The main goal of this kind of histogram, also known as *equi-height*, is to select buckets with (roughly) an equal number of tuples. In terms of the taxonomy, these are presented as *equi-sum(V,F)*.

Equi-depth histograms are constructed by first sorting the data on V and taking β-1 equally-spaced splits. These β-1 bucket boundaries or delimiters are known as *quantiles*. To carry out the construction of these quantiles some techniques have been proposed either to compute the exact values or to compute approximations (e.g., the P^2 algorithm [41] and random sampling algorithms [48, 52]).

V-Optimal Histograms

As mentioned in section 2.5.3, v-optimal histograms partition the data distribution such that the variance of source-parameter values within each bucket is minimized. Numerous source and sort parameters have been proposed in the literature. Among them the most used are attribute value (V), frequency (F) and area (A).

Among all possible combinations, V-Optimal(V,F) and V-Optimal(V,A) have proven to be the most effective implementations. They group contiguous attribute values while minimizing the variances in frequencies and areas respectively. The attribute value as sort parameter has turned out to be a good approximation of the value domain, while the frequency and the area as source parameters ensure that skew in both frequency and value are considered when defining the boundaries of each bucket.

The main drawback of this kind of histogram is the exponential construction cost. An alternative to the exponential solution is to implement end-biased histograms instead of serial class histograms. V-Optimal-end-biased histograms have demonstrated themselves to be less expensive than the serial version. Nevertheless, they are also less accurate.

Maxdiff Histograms

In section 2.5.3, we presented the partition class that forms the foundation of Maxdiff histograms. The decision of using different sort and source parameters is made in terms of the same analysis carried out for the v-optimal histograms (e.g., Maxdiff(V,F) and Maxdiff(V,A)). The experiments in [54] showed Maxdiff(V,A) as the histogram of choice over all data distributions. That is because Maxdiff(V,A) outperformed the alternatives in both issues, construction time and generated error.

Compressed Histograms

As mentioned in section 2.5.3, these histograms place the attribute values with the largest source parameter values (frequency or area) in singleton buckets. The rest of the values are partitioned as in an equi-sum histogram. When data follows highly skewed distributions, these histograms appear to be a good alternative since they achieve great accuracy, keeping the highest values of the source-parameter (e.g., frequencies or areas) in independent buckets.

2.5.5 Multi-Dimensional Histograms

When multidimensional spaces are our object of analysis, different kinds of issues come into play. Partition constraints, as well as sort and source parameters, might be used again, while the value and frequency assumptions have to be subtly adjusted to the new requirements. In this section we review the three most significant methods previously presented in the literature.

hTree Histograms

Muralikrishna and DeWitt proposed in [48] what is considered today the first multidimensional histogram. In their original work, an algorithm for generating multi-dimensional equi-depth histograms was presented. Like the one-dimensional equi-depth histogram, the n-dimensional version called for buckets with roughly the same number of tuples. In terms of its method for dividing the space this algorithm is quite similar to a Grid-file. The β non-overlapping d-dimensional rectangular regions are built by dividing the entire data space recursively into half spaces, using a boundary value and a different dimension each time. The dimension and the boundary value change at the beginning of each iteration.

In terms of the taxonomy presented in Section 2.5.3, this class of algorithms correspond to a *serial* partition class (given the non-overlapping quality of the buckets) that uses the multi-dimensional values as the sort parameter, frequency as the source parameter, and the boundaries chosen with equi-sum as the partition constraint.

In [48] we also see the first time an R-tree mechanism was proposed to retrieve data efficiently from a set of buckets. In their proposal, the leaves of the R-tree correspond to the multi-dimensional equi-depth buckets. This novel variant was called *hTree* in order to differentiate it from the R-tree itself. *hTree* became the name used in the literature to describe the complete process used to generate multi-dimensional histograms and not only to describe the index. It is worth mentioning here that our own research also uses R-tree structures as a framework for the creation of buckets.

mHist Histograms

Almost ten years after *hTree* was presented, a new multi-dimensional histogram came into play, namely *mHist-p* [53]. This new technique first places the entire joint distribution in a single bucket. Then, at each step of the processing algorithm, the space of one of the available buckets is partitioned along one of the dimensions into p subspaces, until it reaches the maximum number of buckets. The split is made based on the 1-dimensional partition constraint and source parameter chosen, and along the dimension that is considered the most "critical" (i.e., *whose marginal distribution is the most in need of partitioning*). According to [53], among the *mHist-p* algorithms, p=2 (i.e., *mHist*-2) proved to have the best performance. Different partition constraints and source parameters were tested, including v-optimal, maxdiff, frequency and area respectively. Overall, *mHist* represented an improvement to the then existing techniques.

genHist Histograms

In [27], Gunopulos et al. introduced *genHist*, which allows unrestricted overlap among buckets. *genHist* was presented as the most robust and accurate multi-dimensional histogram among a plethora of tested approaches (hTree, mHist, kernels, sampling, and independence assumption). The main idea is to build progressively coarser grids over the data set, convert the densest cells into buckets of the histogram, and then remove a certain percentage of tuples in those cells to make the resulting distribution more uniform. As mentioned above, unlike all previous techniques, *genHist* allows buckets to overlap in the space of multi-dimensional values, thus creating more distinct areas than there are buckets per se. The data distribution approximation for a given

query box is therefore a combination of what all the overlapping buckets that form the area indicate.

It is worth mentioning that, even though *genHist* was originally proposed in the context of multi-dimensional real-valued data, it has also been applied to problems with different valued datasets, as demonstrated in [27, 50].

Conclusions

In this chapter, we have examined some of the primary concepts and functionality relevant to data warehousing, on-line analytical processing, query processing and histograms. We began with a discussion of the general area of Decision Support Systems, presenting it as an important component in the improvement of today's decision-making process. Data warehouse architectures and conceptual/logical models (i.e., star and snowflake schema) were also illustrated, with a particular emphasis on the multi-dimensional nature of the data. The three most important storage models (MOLAP, ROLAP and HOLAP) were also examined, presenting advantages and disadvantages of each implementation.

We also explained the importance of selectivity estimation in query optimization, introducing the subject of our own research, namely histograms. We formally defined the histogram, providing a brief overview of a taxonomy that captures all previously proposed types of histograms. Finally, we discussed different one-dimensional and multi-dimensional implementations.

Chapter 3

rK-Hist: A New Multi-Dimensional Histogram

3.1 Introduction

On-line Analytical Processing (OLAP) has become an essential tool in the corporate decision making process. In that respect, we note that decision support places some rather different requirements on database technology compared to traditional on-line transaction processing applications. One of the most important changes posed by this novel class of systems is in the nature of their queries, namely queries that involve multiple predicates. OLAP systems are frequently required to retrieve data from several perspectives - *dimensions* - which affects directly the way query optimizers produce accurate size estimates. These estimates are crucial in determining satisfactory query execution plans. As a result, OLAP systems must provide adequate means for dealing with analytical queries that are multidimensional in nature. We also note that, in addition to query optimizers, we can use the histograms for approximate query resolution, a function that is extremely valuable for terabyte scale data warehouses.

In this chapter, we propose a multidimensional histogram that approximates the

density of multidimensional datasets. Our histogram extends a previously proposed multi-processor platform that targets the relational database model (Sidera).

This chapter is organized as follows. Section 3.2 reviews previous research in the area of Multidimensional Histograms, in terms of its primary weaknesses and/or limitations. In Section 3.3 we present the motivation for our own work. Section 3.4 provides the features and terminology relevant to the techniques presented in the remainder of the chapter. In Section 3.5, we discuss different hyper-bucket integration techniques that produce what we called r-tree histograms. We present a complete description of our new approach in Section 3.6 and discuss the quality measure used by it in Section 3.7. Finally, we review our research objectives in Section 3.8 and conclude the chapter with a brief summary in Section 3.9.

3.2 Related Work

The first non-trivial attempt to approximate data distributions for the purposes of selectivity estimation was described in [44]. Here, a single dimensional point space is divided into a set of buckets, in which the *spread* of values in each bucket is equivalent. The result is what is now referred to as the *equi-width histogram.* In [52], the *equi-depth* (or equi-height) histogram, in which buckets represent equivalent frequency summations, was shown to provide significantly better selectivity estimation than the original equi-width method.

Subsequently, a series of alternative partitioning parameters and techniques was presented in the literature, each attempting to more accurately reflect the point distributions of arbitrary spaces. In fact, the plethora of methods led to the design of a histogram taxonomy [54], which described histograms in terms of elements such as the

source parameter (e.g., spread, frequency, and area) and the partitioning constraint (i.e., how to partition based upon the source parameter). In addition to the *equi-sum* constraint employed by equi-width and equi-depth histograms, alternative constraints have included the *v-optimal* approach that tries to minimize the variance of source parameter values [38], and the *maxdiff* technique that searches for the largest "gaps" between source values [54].

In multi-dimensional spaces the problem is somewhat more challenging since it was shown in [49] that optimal splitting even in two dimensions is NP-hard. In order to provide a solution to the multidimensionality expressed by OLAP systems, most DBMS vendors adopted the *attribute value independence* assumption. AVI is a well-known heuristic that estimates the selectivity of a conjunction of predicates on multiple attributes by taking the product of the marginal selectivities of the individual predicates, e.g. $Predicate(A = a \wedge B = b)$ corresponds to $Predicate(A = a) \cdot Predicate(B = b)$ [2]

Because the application of the AVI assumption proved to be a source of significant errors [12], numerous techniques for modeling correlated multidimensional distributions have been proposed in the research literature. Among those techniques, multi-dimensional histograms have been the subject of much experimental work in the last few years. Instead of adopting the AVI assumption, multi-dimensional histograms approximate the densities by splitting the datasets into d-dimensional buckets. Since no assumptions are made about the dependence or independence of the values, query optimizers are capable of more accurate approximations by using multi-dimensional histograms.

In this section we review the three most significant methods previously presented

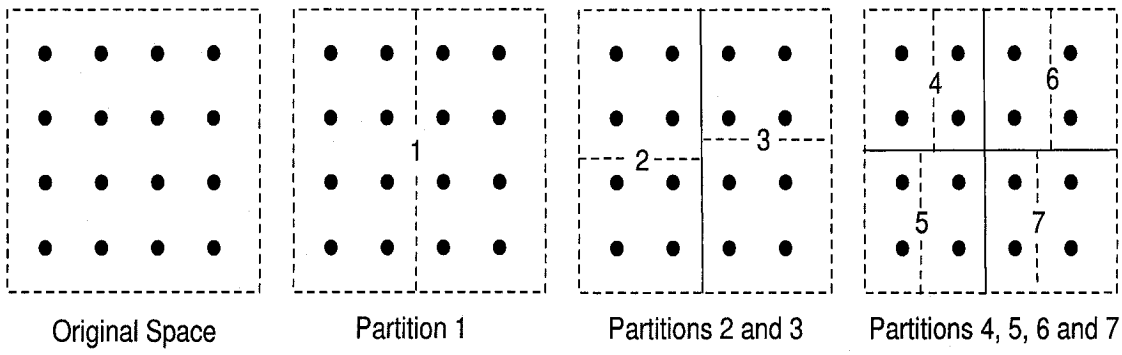


Figure 3.1: hTree recursively partitions the data space into half-spaces by using a different dimension each time.

in the literature – introduced in Section 2.5.5 – and identify the benefits provided by these approaches, as well as those features that in fact make them impractical to use in realistic environments.

The first work on muti-dimensional histograms that was reported in the literature was undertaken by Muralikrishna and DeWitt [48]. Here, the authors specifically described a 2-dimensional equi-depth histogram referred to as *hTree*. In terms of their approach to generate an explicit number of buckets, they opted for a *Grid-file*-oriented strategy. Fundamentally, they recursively divide the data space into half-spaces by using the value of one of the dimensions as a boundary each time. As Figure 3.1 illustrates, the problem of generating equi-depth histograms consists of covering all the points in the data space with β rectangles such that each rectangle encloses approximately $\frac{R}{\beta}$ points within it, where R represents the total number of points in the data space. The foremost advantage of hTree is its low sorting cost $C \times P \times \log\left(\frac{P^d}{b^{\frac{d(d-1)}{2}}}\right)$ where C is some constant, P represents the number of data pages in the relation R and b represents the same number of partitions at each stage of the algorithm.

While this technique has a significantly reduced construction cost, the accuracy of its approximation is not as good as more recent proposals due in most part to the arbitrariness when deciding the order in which the dimensions are to be split. For spaces with more than five dimensions, as we will see in Chapter 4, the error is so large that it makes any estimation inappropriate. Finally, this technique obtained its most accurate results while working on uniform dataset. Such behavior was expected given the uniform partitioning made by hTree. However, it is important to mention that real data sets are far from being uniform. In fact, during the experiments, we noticed that when the data distribution was slightly changed from uniform to skewed (0.1%) the normalized absolute error increased considerably.

In [53], Poosala and Ioannidis presented *mHist*, a new multidimensional histogram that attempts to improve upon the quality of partitioning by recursively dividing the space on the dimensions that are judged to most benefit from a split (in terms of density). mHist works directly on the arbitrariness presented in hTree regarding the selection of the dimension to be split in every iteration. This technique decides the partitioning point (dimension and attribute) according to the *marginal distributions* (i.e., the individual data distributions of each of the attributes) of the dataset. First, the algorithm chooses the distribution γ that contains the attribute χ whose marginal distribution is the most in need of partitioning and then it partitions γ along χ into ρ buckets. The new number of buckets, in turn, becomes the input for a new iteration (i.e., a new estimation of marginal distributions, the selection of a new partitioning point and the respective splitting). Note that ρ represents an important variable since different values may result in different histograms. The authors referred to the mHist technique that uses χ splits as mHist-χ. Overall, mHist-2 was considered the

approach that most often results in desirable histograms[53]. An illustration of mHist algorithm's splitting process can be seen in Figure 3.2.

mHist histograms tried to solve some of the problems with hTree histograms for cases when the data distribution is highly skewed. However, because of the way mHist recursively partitions the data set, it assigns too many buckets to the densest tuple clusters, and almost none to the rest of the data domain, degrading the overall histogram accuracy.

Even though the experiments presented in [53] showed mHist as a very accurate histogram outperforming other techniques like AVI, hTree and a "basic" implementation of the Hilbert curve, it was shown in [27] that for a large number of datasets with different data distributions mHist's estimates are unusually poor and in some cases they are even worse than hTree. It seems that the several synthetic joint data distributions that were generated in [53] were not sufficient; thus the differences in terms of error estimation between the results presented in [53] and [27] might be attributed to a lack of tests (in the former one) conducted on more diverse data sets.

Since mHist, there have been a large number of interesting techniques that have been proposed. One of the most referenced proposals was presented in [27] by Gunopulos et al. Here, the authors introduced *genHist*, which addresses some of the problems outlined above and, in contrast to the previous techniques, allows buckets to overlap in the multi-dimensional space, just as r-tree buckets do. The algorithm works as follows. First, it partitions the data space in a similar way to the normal grid-file algorithm. After that, from the partitions created, the ones with the highest density are selected in order to apply a heuristic technique that creates the first set of final buckets. Subsequently, successively bigger grids are built over the resulting data set

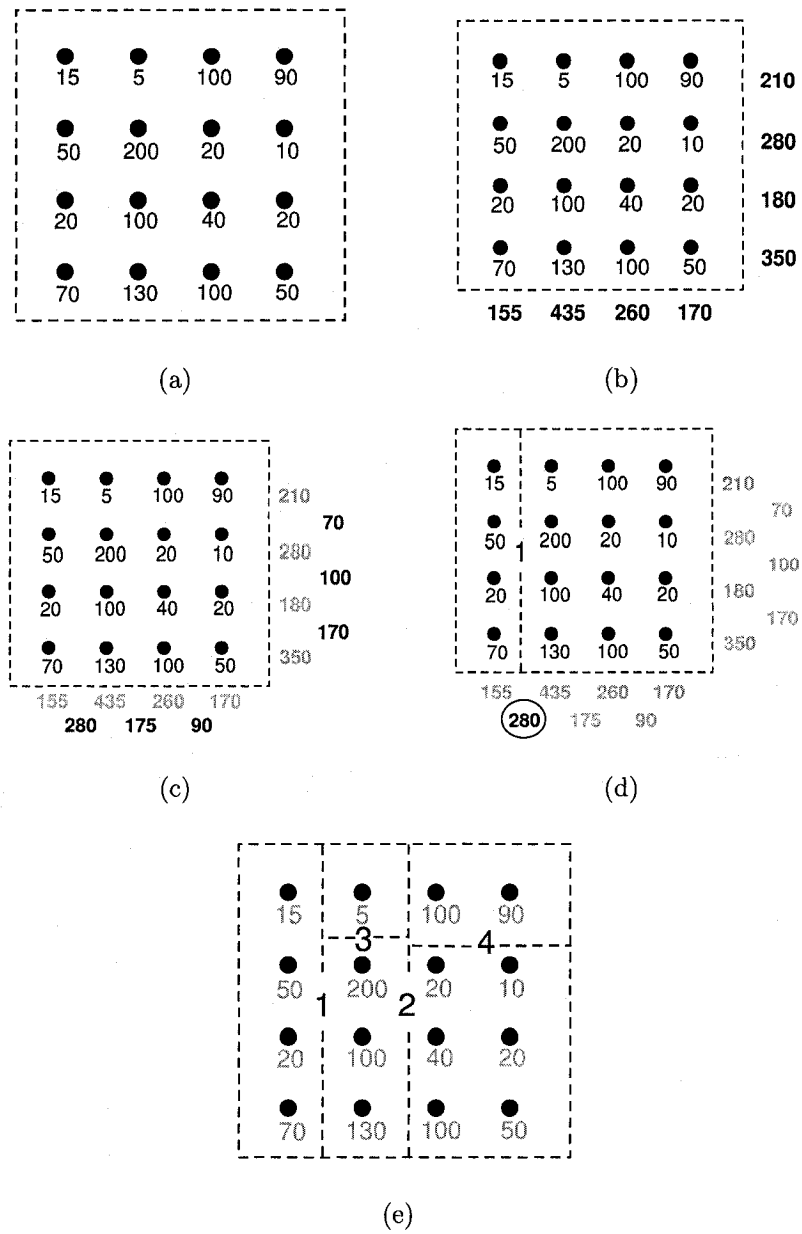


Figure 3.2: (a) The original dataset. (b) Step 1. Estimating marginal distibutions. (c) Step 2. Calculating differences between adjacent marginal distibutions. (d) Step 3. Identifying the maximum difference and creating the first bucket. (e) Step 4. Partitioning of the data space after creating four mHist buckets.

and the densest cells are converted to buckets until it reaches the total number of buckets required.

A major disadvantage of this approach is the difficulty of choosing the right values for a crucial number of parameters. Specifically, the initial grid size, the number of buckets created per iteration, and the value that controls the rate by which the number of partitions (grid size) decreases. These are values that are dependent of the data set and at the same time are required to guarantee high-quality approximations. Even though this technique generally results in better accuracy than the techniques discussed above, as illustrated in Chapter 4, it forces the parameter setting to be redefined every time there is a change in the data distribution causing degraded performance in some cases. Another drawback of this technique is that, according to the authors, it requires many passes (at least 5 to 10) over the whole data set [27].

In addition to the "static" histograms described above, we note the existence of a special class of multi-dimensional histograms based on dynamically generated models. Here, rather that constructing the histogram statically from an existing data set, the histogram is generated and updated in real time. *stHoles*, for example, uses incoming queries to produce nested buckets [50], while ISOMER extends the basic stHoles model by adding the notion of entropy minimization [60]. Dynamic generation using an intermediate summary structure has also been pursued in the context of continuous data streams [62]. In general, the dynamic techniques have been shown to work fairly well in low dimensions, though they cannot be expected to outperform the static methods across a broad range of data distributions.

A small number of papers have also presented theoretical support for histogram construction models that define upper bounds on estimation errors [40, 26]. However,

these limits are available only for the much simpler case of one dimensional histograms.

Finally, we add that alternatives to histograms have also been explored in the literature. Examples include wavelets [64], table sampling [30], and discrete cosine transform[46]. In general, however, histograms have remained the most popular target for selectivity estimation and approximate query answering due to their effectiveness and robustness across a wide variety of application domains. A relatively full record of their history can be found in [37].

With respect to multi-dimensional indexing, we note that this has historically been an active area of research [24], though few of the experimental methods have found their way into production systems. The r-tree itself was proposed by Guttman [29], while the concept of pre-packing the r-tree for improved storage and performance was first discussed by Roussopoulos and Leifker [56]. Packing strategies were reviewed by Leutenegger et al. [47] and Kamel and Faloutsos [42]. The former promoted a technique they called Sort Tile Recursion (STR), while the latter identified Hilbert ordering as the superior approach.

The concept of representing a multi-dimensional space as a single dimensional, non-differentiable curve began with Peano [51]. Jagadish provides an analysis of four such curves — snake scan, gray codes, z-order, and Hilbert — and identifies the Hilbert curve as providing the best overall clustering properties [39], a result duplicated in [23].

3.3 Motivation

Though the research efforts described in the previous section have attempted to exploit the power of multi-dimensional histograms to more efficiently approximate the

selectivity of a given query, they have had only partial success. Each approach provided estimates and construction costs that were either unimpressive or suggested that performance in general would in fact be quite poor. For these two reasons — accuracy and construction cost — there is clearly an opportunity and an impetus for further research in this area.

In approaching the design of a new multi-dimensional histogram, we identified the following four primary objectives:

1. Minimize the "dead space" resident in each bucket, thereby improving the uniformity of point distribution.

2. Maximize the accuracy in approximating the underlying data distribution of diverse datasets while being efficient in terms of running time.

3. Support straightforward integration with standard relational systems.

4. Build upon proven, optimized algorithms, while exploiting the well studied r-tree based distributed multi-dimensional ROLAP indexing scheme, RCUBE.

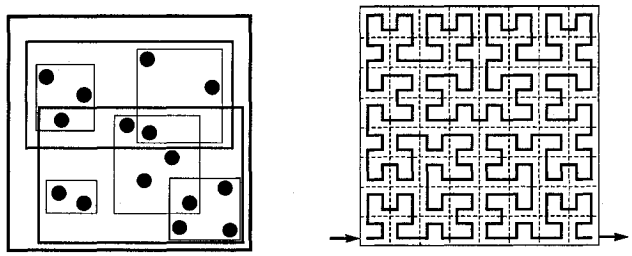
In the remainder of this chapter, we present the details of rK-Hist, an r-tree based histogram for multi-dimensional selectivity estimation. Upon the conclusion of the chapter, we will review the extent to which we have accomplished each of the four basic objectives.

3.4 A New Approach: Tree Indexing/Histogram Partitioning

Our new methods build directly upon fundamental r-tree construction techniques, as well as the methods for implementing and manipulating space filling curves. Therefore, in this section, we briefly review the salient features and terminology relevant to the techniques presented in the remainder of the thesis.

Previous multi-dimensional histogram methods have taken what might be described as *ad hoc* approaches to space partitioning. However, given that partitioning is a fundamental problem in a number of research domains, there is much to be gained by both an examination and exploitation of previous partitioning work. This point was also made in [37], where the similarity between tree indexing and histogram partitioning was highlighted. Specifically, it was suggested that construction of a B+tree effectively produced single dimension buckets that could easily form the basis of a simple histogram. Furthermore, the tree-based design suggested the possibility of a hierarchical histogram that could efficiently return approximation results. Having said that, the basic technique is hampered by the fact that index partitioning is not necessarily conducive to producing uniform bucket distribution and, hence, low error rates [54].

Our new methods in fact build directly upon the notion of index-histogram integration. Of course, given that our focus is multi-dimensional rather than single dimensional spaces, the B-tree and its variations are clearly not appropriate as a starting point. Instead, we draw upon complementary research carried out in multi-dimensional database environments, specifically that associated with enterprise-scale data warehousing (DW). While a number of indexing methods have been utilized in



(a) (b)

Figure 3.3: (a) r-tree partitioning for a maximum node count of four. (b) A $\mathcal{H}_3^2$ Hilbert curve.

DW domains (e.g., bitmaps, join indexes), the r-tree has proven to be one of the more popular and successful. It is particularly appealing in the current context as it represents a true multi-dimensional decomposition of the point space.

Briefly, the r-tree is a hierarchical, d-dimensional tree-based index that organizes the query space as a collection of nested, possibly over-lapping hyper-rectangles [29]. The tree is balanced and has a height $H = \lceil \log_M n \rceil$, where M is the maximum *branching factor* and n is equivalent to the number of points in the data set. A user query Φ may be defined as $\{r_1, r_2, \ldots, r_d\}$ for ranges $(r_{min(i)}, r_{max(i)}), 1 \leq i \leq d$, and is answered by comparing the values on $\{r_1, r_2, \ldots, r_d\}$ with the coordinates of the rectangle $\aleph$ that surrounds the points in each data page. Figure 3.3(a) illustrates a simple r-tree decomposition.

A number of extensions to the basic r-tree were proposed in order to improve the clustering properties of the original tree. The r^+-tree [58] alters the basic model by prohibiting box overlap at the same level of the tree, while the r^*-tree [3] uses an object re-insertion technique to reduce overlap during node splitting. However, in more static environments such as that found in data warehousing, more effective clustering and storage can be obtained by *pre-packing* the r-tree index. In this regard,

the Hilbert order employed in [42] is particularly interesting as it can be used to provide a linear ordering of points in a d-dimensional space, one that can then be mapped to the single dimension of the storage medium.

First proposed in [33], the Hilbert curve is a non-differentiable curve of length s^d that traverses all s^d points of the d-dimensional grid having side length s, making unit steps and turning only at right angles. The curve can be visualized as an interval I existing within the unit square S. If we decompose S into equivalent sub-squares S_i, for $1 \leq i \leq 4$, then with a series of reflections and rotations, we may concatenate the sub-intervals I_i to satisfy the requirements of the linear mapping. We say that a d-dimensional cubic space has order k if it has a side length $s = 2^k$ and use the notation $\mathcal{H}_k^d$ to denote the k-th order approximation of a d-dimensional curve, for $k \geq 1$ and $d \geq 2$. Figure 3.3(b) shows a small 2-d curve.

Given the point order defined by the Hilbert curve, the packing strategy proceeds as follows:

1. Sort the points in terms of their position along the Hilbert curve so that the n points are associated with m pages of size $\lceil \frac{n}{M} \rceil$. The page size/branching factor is chosen so as to be a multiple of the disk's physical block size.

2. Associate each of the m leaf node pages with an ID that will be used as a file offset by parent bounding boxes.

3. Construct the remainder of the index by recursively packing the bounding boxes of lower levels until the root node is reached.

The end result is a hierarchy of *bounding boxes* that partition the space at varying levels of granularity. Moreover, the boxes are constructed so as to most effectively

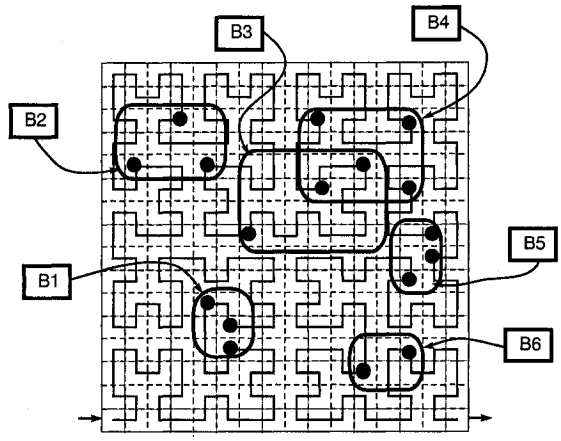


Figure 3.4: Packed r-tree leaf level partitioning for an $\mathcal{H}_4^2$ Hilbert curve. Note the order of the bucket sequence.

maintain point locality. In other words, points close to one another in the multi-dimensional space tend to be partitioned into the same bucket (or sibling buckets). Figure 3.4 illustrates how the Hilbert packed r-tree can be used to partition a small two dimensional space. Note the inherent clustering properties of the underlying Hilbert curve.

3.5 r-tree Histograms

The Hilbert packed r-tree provides us with an interesting starting point in terms of its ability to cluster multi-dimensional data buckets. Specifically, following the Hilbert sort of the data and the r-tree partitioning, we are left with a hierarchy of β rectangular bounding boxes that each enclose regions of spatially related points. That being said, the r-tree index itself is simply too large to function directly as a histogram, since histograms must remain fully memory resident in order to be effective. As such, the β boxes must be integrated in some manner into α *hyper-buckets*. Hyper-bucket integration can in fact be supported in two different ways.

3.5.1 Top down construction

Because tree-based indexes are hierarchical, there are in fact a series of space partitions at varying degrees of granularity. For example, given $n = 15$, $M = 3$, we have height $H = \lceil \log_M n \rceil = 3$. As such, there are three partitions of the d-space, at successively finer levels of detail. In the top down histogram, the idea is to construct α hyper-buckets working downwards from the root. At first glance, it would be tempting to simply "slice off the top of the tree". Doing so, however, would result in a ragged cut of the tree, which is of absolutely no value. Instead, the technique is as follows:

1. Start with a fully constructed r-tree index, consisting of $H = \lceil \log_M n \rceil$ levels.
2. Descend from the root, examining each of the i levels, $1 \leq i \leq H$.
3. Identify the first level L for which $\alpha \leq |L_i|$
4. Coalesce this level into α hyper-buckets, each containing $\lceil |L_i|/\alpha \rceil$ r-tree buckets.

As a concrete example, assume that we have enough memory to construct a 200-bucket histogram. Furthermore, assume that we also have an index structure with initial bucket counts of $L_1 = 20$, $L_2 = 120$, $L_3 = 800$, and $L_4 = 5000$. The top down method selects L_3 and coalesces the 800 tree buckets into 200 hyper-buckets of 4 nodes each. Figure 3.5 provides a simple illustration.

3.5.2 Bottom up construction

In contrast to the top down approach, we can also construct the α hyper-buckets working strictly from the leaf level of the r-tree. The technique for construction is

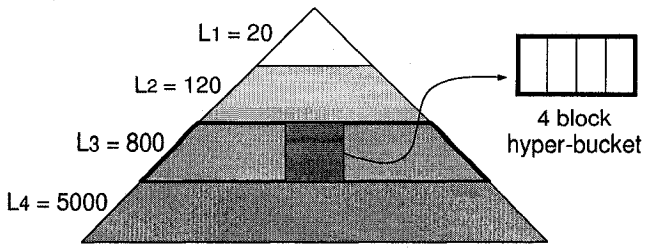


Figure 3.5: Top down histogram partitioning for $\alpha = 200$. The 800 r-tree buckets at Level 3 are coalesced into a sequence of hyper-buckets.

essentially the same as that of the top down method, in terms of the concatenation of r-tree buckets into histogram hyper-buckets. In fact, the boundaries of the hyper-buckets produced by a bottom up histogram are actually quite similar to those of the top down model. Nevertheless, extensive testing has shown that due to its less restrictive initial partitioning, the bottom up method does produce slightly more uniform hyper-bucket distributions. Moreover, it is simpler to integrate with the inherently bottom-up r-tree construction algorithms. As such, it forms the basis of the rK-Hist methods discussed throughout the remainder of the thesis.

3.5.3 Performance considerations

It is assumed that rK-Hist would be utilized in a comprehensive multi-dimensional database environment in which an associated r-tree(s) was also being constructed. This allows us to "piggy back" rK-Hist on top of the core r-tree generation algorithms, thereby obtaining most of the histogram functionality for little additional cost. While this is the ideal, we note that rK-Hist can also be constructed without an associated r-tree, if desired. In either case, the most costly element of the r-tree/rK-Hist process is the sorting of the underlying data set in Hilbert order. It is therefore important to ensure that Hilbert sorting is not prohibitively expensive.

We note the following. Given an $O(n \lg n)$ comparison based sort, we can assume an average of $\lg n$ Hilbert space comparisons for each of the n tuples in the data set R. Since tuple conversions are identical in each case, significant performance improvements can be realized by minimizing the number of redundant Hilbert transformations. Consequently, our framework uses a pre-processing step in which the n tuples of R are converted into Hilbert ordinal form (i.e., their numeric position along the curve) *prior* to sorting. A purely integer-based sort is then performed, followed by a linear mapping back into tuple form. This single optimization typically reduces costs by an order of magnitude or more, depending on dimension count. In fact, our modified Hilbert sort typically runs within a factor of two of the cost of a simple, multi-dimensional integer sort.

3.5.4 A naive r-tree histogram

Assuming that the initial r-tree blocks have been coalesced into α hyper-buckets that partition the full space, we are now ready to produce an accessible histogram. At this initial stage, we will utilize the standard frequency based equi-depth distribution model for our buckets. That is, within each hyper-bucket B we record its point count B_{count} and volume B_{volume}. Given a user-defined d-dimensional query $\Phi = \{r_1, r_2, \ldots, r_d\}$, a selectivity estimate may therefore be defined as the intersection of Φ with a subset S of the α hyper-rectangles.

Recall, however, that the r-tree blocks, and hence histogram hyper-rectangles, can overlap in the point space. One might expect selectivity estimation to be considerably more difficult as a result. In fact, this is not the case. When boxes overlap, each defines a distribution that is unique to its own boundaries and to the specific points that it contains. In other words, even though portions of the space may be shared,

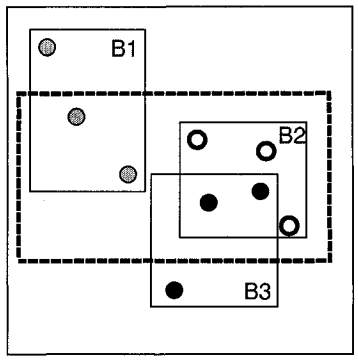


Figure 3.6: A user query (dashed line) intersecting a series of over-lapping partitions.

their points are not. Consequently, much like the genHist histogram, the distribution estimates may simply be summed together as though the boxes were non-overlapping. As a concrete example, Figure 3.6 depicts a pair of over-lapping histogram buckets, B2 and B3. In this case, their contributions to the estimate of Φ can be summed directly. Intuitively, the summation indicates a region of increased density.

Calculation of the final estimate therefore proceeds as follows. For a user query Φ, we evaluate each histogram bucket B to determine the degree of overlap. Assuming *uniform spread* inside the bucket, we calculate the estimate on B in terms of the ratio of the overlapped region to the full volume of B. More formally, we may say:

$$estimate(S, \Phi) = \sum_{B \epsilon S} \frac{B_{count}}{B_{volume}} Vol(\Phi \cap B)$$

where $Vol(\Phi \cap B)$ is the volume of the intersection between Φ and B.

Finally, we note that the r-tree histogram is itself constructed as a small r-tree. As such, selectivity estimation does not require serial access of the α buckets of the histogram. Instead, the leaf nodes/buckets can be accessed with the standard r-tree cost complexity ($O(\lceil \log_M n \rceil)$).

3.6 The Sliding Window Algorithm

We have described the algorithm presented thus far as the "naive" r-tree histogram. In short, we have exploited the standard r-tree construction algorithm so as to produce a set of α buckets approximating the underlying multi-dimensional point space. However, the default ordering of the Hilbert curve may produce less than optimal bucket partitions. This is primarily because outliers on the curve may occasionally distort the scale of the encapsulating bounding boxes. While we can accept a certain degree of "box stretching" — no space filling curve can perfectly maintain locality in a multi-dimensional space — the existence of extreme outliers may lead to unacceptably inaccurate estimation errors.

As a result, we extend the basic technique with an optimization component. Algorithm 1 describes the enhanced r-tree approach. We begin by creating the naive r-tree histogram. As noted, this will likely include a small number of hyper-buckets with relatively poor point distribution characteristics. To minimize the error generated by such buckets, we actually create an initial set I consisting of $\alpha - (\alpha * \theta)$ buckets, where θ refers to the *under-sampling ratio*. We then use a quality measure to identify the $(I * \frac{1}{2}\theta)$ buckets with the worst distributions (the quality measure will be discussed in Section 3.7). While θ is a tunable parameter, we note that in practice a θ ratio of 10% works consistently well. The selected buckets are then placed into a list Ψ.

At this point, the algorithm scans each hyper-bucket B to determine how its distribution quality might be improved. As noted previously, optimal partitioning is NP-Hard. Consequently, we will employ a heuristic method that seeks to re-partition B to produce more uniform point distribution. However, for a hyper-bucket B containing k constituent buckets, there are exactly 2^k possible partitions. It is infeasible

Algorithm 1 Sliding Window

Input: A Hilbert ordered data file R, the total bucket count α, and the under-sampling ratio θ.

Output: An enhanced multi-dimensional r-tree histogram.

1: Partition into blocks as per the naive r-tree histogram algorithm.
2: Using the bottom up method, create $I = \alpha - (\alpha * \theta)$ hyper-buckets from the leaf-nodes of the tree.
3: Create a list Ψ with the $(I * \frac{1}{2}\theta)$ hyper-buckets of lowest quality produced in the previous step.
4: Within each hyper-block in Ψ, record the constituent disk block IDs.
5: **repeat**
6: **for** each hyper-bucket B in the list Ψ **do**
7: Using a "sliding window" technique, examine the combinations of successive buckets housed in B.
8: Determine the two sub buckets, B_1 and B_2, with the most improved quality measurements. Use block IDs to obtain point distributions as necessary.
9: **if** B_1 and B_2 represent an improvement over B **then**
10: Replace B in Ψ with B_1 and B_2.
11: **end if**
12: **end for**
13: **until** $|\Psi| + |I| = \alpha$ OR no buckets in Ψ can be further improved.

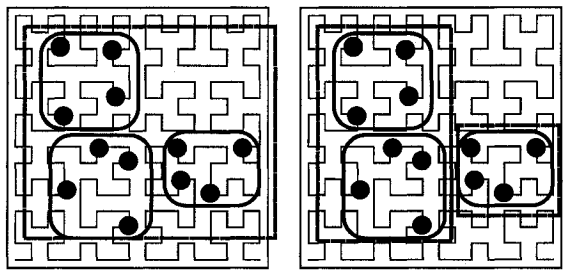


Figure 3.7: Use of the the "sliding window" method to decompose hyper-blocks along the curve.

to calculate each such partitioning for each bucket in Ψ. Instead, we use a "sliding window" technique to decompose B into two sub-buckets B_1 and B_2 as per the order of the Hilbert curve itself. For example, assume that B contains four constituent blocks B_a, B_b, B_c, B_d. We therefore consider the decompositions $\{(B_a), (B_b, B_c, B_d)\}$, $\{(B_a, B_b), (B_c, B_d)\}$, $\{(B_a, B_b, B_c), (B_d)\}$. There are, of course exactly $k-1$ such splits. Figure 3.7 illustrates the way in which the sliding window method might be used to eliminate "dead space" that has been accumulated during the traversal of the Hilbert curve.

During the recalculation of the quality measure, it is of course necessary to work with the raw distribution of points in the leaf level r-tree buckets. While this information could be retained in memory, the scale of many multi-dimensional data sets would make this infeasible. As such, the hyper-buckets of Ψ maintain the list of block IDs constituting the hyper-bucket. When re-calculations are required, these blocks, and only these blocks, need be retrieved. It is therefore unnecessary to re-scan the full data set in future rounds.

Once the primary hyper-buckets have been partitioned, they may be re-inserted into Ψ for consideration during the next partitioning round. We note that while

we partition each hyper-bucket into just two sub-buckets during the previous step, the algorithm is free to further partition a sub-bucket in subsequent iterations. The current list is then reprocessed in a greedy fashion until either we have produced α total buckets or no improvement can be produced in the hyper-buckets currently found in Ψ. Typically, this happens within just two to three iterations.

3.7 The k-Uniformity metric

In the previous section, we presented a sliding window algorithm that was meant to improve the uniformity of hyper-buckets. Within the blocks that comprise the hyper-buckets, however, the quality of the distribution must be defined more precisely. Traditionally, distribution quality is associated with a *density* metric. In effect, this provides us with an estimate of the number of points per unit square. For example, given a d-dimensional space with dimension cardinalities $\{C_1, C_2, \ldots C_d\}$, and a bucket B, we have $B_{volume} = \prod_{k=1}^{d} C_k$. For a point count of B_{count}, the density is therefore calculated as $\frac{\beta_{count}}{\beta_{volume}}$. However, while density does provide a coarse estimate of the quality of the point distribution, it can lead to relatively significant estimation errors in non-uniform spaces. For example, Figure 3.8 displays three common bucket distributions. Each 2-d box contains exactly 10 points and is constructed with axes of equivalent length. Consequently, each has an identical density value and would therefore be deemed to provide selectivity estimates of equivalent accuracy. However, this assumption is clearly not accurate. While Bucket A represents an essentially uniform distribution, Bucket B houses far more "dead space". Bucket C is even worse. It should be obvious that when the *uniform spread assumption* is applied to these three buckets, estimation error will vary widely.

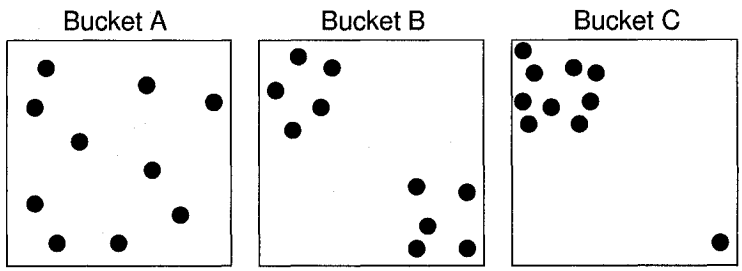


Figure 3.8: Examples of three typical point distributions, each resulting in the same density measurement.

Alternatively, one can define bucket quality in terms of the *uniformity* of its bucket distribution. In other words, the objective function should be defined it terms of a metric that seeks to minimize the dead space between bucket points. While a number of brute force approaches to such an optimization are certainly possible, the scale of the problem requires a technique that is computationally superior to a naive $O(n^2)$ solution.

To this end, we propose the k-uniformity (kU) metric. Rather than trying to estimate the relative "closeness" of bucket points, we take the possibly counter-intuitive approach of assessing the relative "emptiness" of the space. To do do, we adapt a technique used in the construction of multi-dimensional indexes. Specifically, we will be using a variation of the partitioning strategy for the **k-d-tree**. Recall that the k-d-tree [4] is a form of binary search tree in which the "test" attribute varies at successive levels. To be precise, for a d-dimensional space, we recursively partition on $A_1, A_2, \ldots A_d$, each time splitting the space on the A_i median. Figure 3.9 provides a simple illustration for a 2-d space.

We use this partitioning mechanism as the basis of the calculation of the k-uniformity metric described in Algorithm 2. Beginning with the points of the current

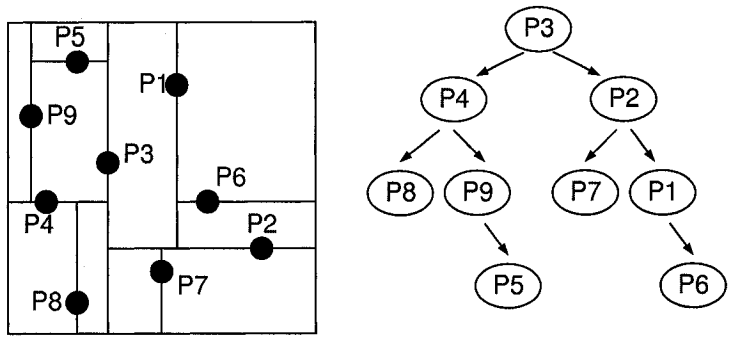


Figure 3.9: The k-d-tree.

bucket B, and an initially empty list of the volumes of the bucket's partitions, we recursively partition the bucket space as per the k-d-tree strategy described in Algorithm 3. Essentially, high/low splits eventually reduce the m points of B to a list of m partitions, each containing a single point. If we consider the hyper-rectangular boundary of such a partition, its volume may be used as a representation of the dead space encapsulating each point. In other words, the greater the volume, the more isolated the point. It is this partition volume that is added to the volume list L. When the recursion terminates, Algorithm 2 then calculates the standard deviation on the box volumes of L. This simple floating point value is the k-uniformity measure for B.

Algorithm 2 k-uniformity calculation

Input: A bucket B, a dimension set $A_1, A_2, \ldots A_d$, and an empty Volume List L.
Output: A k-uniformity measure.

1: Call the recursive k-partitioning function with arguments (B, A_1, L).
2: Calculate the average box volume V_{avg} of the boxes in L
3: Return the standard deviation of the boxes in L relative to V_{avg}

Algorithm 3 Recursive k-partitioning

Input: A set of input points $\alpha_{current}$, the current dimension A_i, and a volume list L.
Output: A k-uniformity measure.

1: **if** The current partition size = 1 **then**
2: Calculate the volume $Volume_p$ of the current partition
3: Append the volume to the *volume list* L
4: **end if**
5: **for** the current dimension A_i **do**
6: Find the median value M_{A_i}
7: Set $\beta_{low} =\leq M_{A_i}$
8: Recurse on $(\beta_{low}, A_{i+1}, L)$
9: Set $\beta_{high} => M_{A_i}$
10: Recurse on $\beta_{high}, A_{i+1}, L)$
11: **end for**

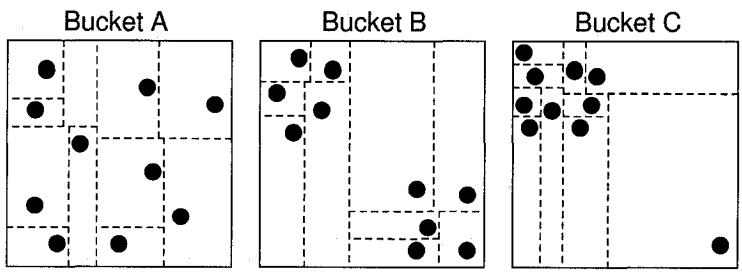


Figure 3.10: Bucket breakdown by area for three previous buckets.

Before describing the formal characteristics of Algorithm 2 and Algorithm 3, it is useful to provide a graphical picture of the k-partitioning technique. Figure 3.10 displays the space partitioning for the three buckets shown earlier. Bearing in mind that the purpose of a standard deviation calculation is to show the spread of values, the effect of the k-uniformity measure should be clear. Bucket A would produce a low kU measure, indicating desirable uniformity. The kU for Bucket B would be somewhat higher, while Bucket C would the most extreme of the three. As such, it would become an obvious candidate for restructuring.

While the kU metric clearly provides concrete advantages, such functionality cannot come at the price of unacceptable computational cost. Given that the kU measure is calculated on the points of each bucket, processing cost is proportional to the size of the full data set. If we examine Algorithm 3, we see that for an initial box with m points, we create $O(\lg m)$ partitioning levels. For each of these levels, we must identify medians on the m points. For this we employ the randomized median finding technique described in [34], which runs in $O(m)$ time in the average case. As such, the complexity of the partitioning phase is $O(m \lg m)$, identical to that of an optimized k-d-tree construction algorithm.

Once the partitions have been generated, the kU metric is calculated. Since this is a volume-based calculation, the time complexity for the m partitions is simply $O(d * m)$. The bound on the full computation is therefore the concatenation of the two steps, or $O(m \lg m + dm)$. The dominant component depends of course on the dimension count and the size of the data set, but in most cases, the processing time would be bounded simply as $O(m \lg m)$. Finally, we note that since the kU for each box is computed independently, the full cost of kU measurement on the data set is simply the summation of the cost for each bucket.

To demonstrate the practical effect of kU based bucket partitioning, we have graphically captured the bucket partitioning patterns of hTree, mHist, genHist, and rk-Hist (kU + sliding window) on a two dimensional data set containing one million points. The data was synthetically generated using a zipfian skew of 0.4 and cardinalities of 60000 on each dimension.

Figures 3.11, 3.12, 3.13, and 3.14 illustrate the final result. (Please note that because of the resolution, it might not be clear that a) there are one million points in

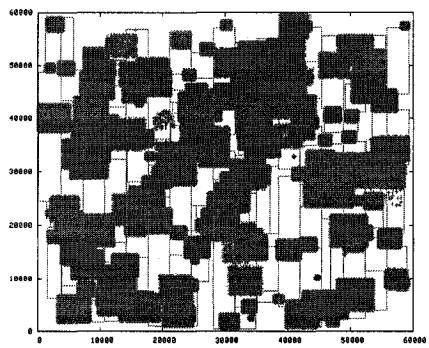


Figure 3.11: Bucket partitioning for the hTree histogram.

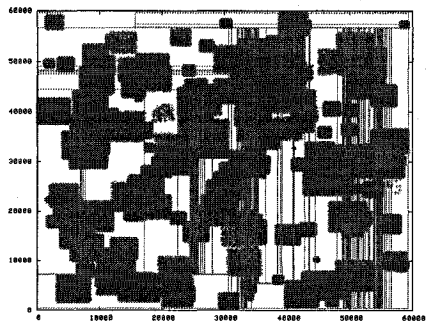


Figure 3.12: Bucket partitioning for the mHist histogram.

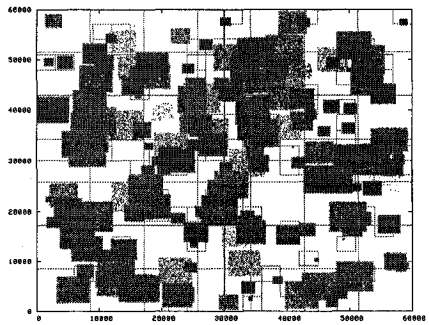


Figure 3.13: Bucket partitioning for the genHist histogram.

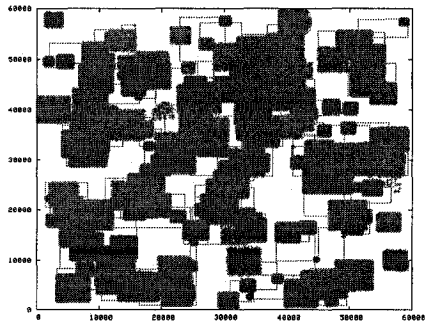


Figure 3.14: k-U based bucket partitioning for the rK-Hist histogram.

the image, and b) the large blocks of points represents the varying densities). As per the design logic of the hTree, its bucket partitions illustrate a pronounced grid-like pattern. While effective for densely populated regions of the space, it tends to create boxes containing significant dead space in regions of reduced uniformity. The mHist algorithm, on the other hand, tends to produced extreme striping patterns on the data set which clearly do not reflect the inherent clustering of the data set. genHist, in turn, also produces significant dead space, despite its ability to generate a more flexible grid.

By contrast, the r-tree/Hilbert curve/kU combination produces a remarkably accurate decomposition of the point space. As can be seen in Figure 3.14, dead space is largely ignored. Moreover, the algorithm is able to vary box volumes so as to

more accurately reflect clustering properties. The end result is a bucket generation mechanism that is far more likely than the alternatives to produce consistently lower estimation errors for arbitrarily defined multi-dimensional queries.

3.8 Review of Research Objectives

In Section 3.3, we identified a number of objectives for the present research. We now review those goals to confirm that they have in fact been accomplished.

1. **Minimize the "dead space" resident in each bucket, thereby improving the uniformity of point distribution.** The combination of r-tree construction techniques, the Hilbert curve, and the k-uniformity (kU) metric produces a remarkably accurate decomposition of the point space, providing uniform bucket distribution with highly reduced dead space.

2. **Maximize the accuracy in approximating the underlying data distribution of diverse datasets while being efficient in terms of running time.** The combination of the kU metric and the Sliding Window Algorithm produced optimized space decomposition. Moreover, we have shown that the presented methods require modest computational resources.

3. **Support straightforward integration with standard relational systems.** The rK-Hist model builds upon indexing models often found in data warehousing environments. Moreover, all input data is read from standard relational tables.

4. **Build upon proven, optimized algorithms, while exploiting the well studied r-tree based distributed multi-dimensional ROLAP indexing**

scheme, RCUBE. Our technique was conceived as part of a larger data warehousing project called Sidera, a framework that explores the design and development of parallel OLAP server technology. We reuse a number of the modules described in [21], which in turn has proven to use some of the most effective known methods for computing the datacube. We also incorporate the RCUBE algorithm into our new selectivity estimator.

3.9 Conclusions

In this chapter, we have described rK-Hist, a multi-dimensional histogram for selectivity estimation. We have presented an approach that uses an r-tree based histogram that exploit the Hilbert space filling curve to generate an initial space partitioning. It then uses a sliding window method, coupled with a new uniformity measure, to further improve the quality of the selectivity estimates.

As noted in Section 3.2, even though a variety of approaches have been proposed, the previous solutions leave room for significant improvement. Given the significance and size of the underlying problem, there would appear to be a genuine need for this type of research. In our case, we have contributed to the literature by providing a solution that can be seamlessly integrated into a relational OLAP framework (e.g., Sidera) and that, as will be demonstrated in the next chapter, provides greater accuracy than the most widely used current methods.

Chapter 4

Evaluation

4.1 Introduction

In this chapter, we provide experimental results that assess the performance of rK-Hist relative to the competing solutions. Specifically, we will compare the quality of rK-Hist to the traditional hTree, mHist, and the more recent genHist. We do not consider the dynamic methods such as stHoles since they serve a different purpose and since they have not been shown to outperform the best static methods across a broad range of test sets.

This chapter is organized as follows. Section 4.2 describes the hardware and software used throughout the development of the research discussed in this thesis. The different data sets, workloads, and query boxes will be presented in Section 4.3. Section 4.4 provides a detailed explanation of qGen, a random-driven query generator that we implemented to create the query boxes used in the testing of the diferent techniques. Section 4.5 provides the results from the tests that we performed and a detailed explanation of each of them.

4.2 Our Physical Model

In this section we present the physical architecture that was used for performance evaluation purposes. However, in terms of the current research, we note that MIMD computer systems were used for the construction of the Parallel Relational OLAP framework whose code represents the foundation for this thesis. Therefore, it is important to say that our design retains all the components that guarantee the applicability of this proposal in multi-processor environments.

Primary software components include:

- Fedora Core 5 (Red Hat Linux distribution)
- GNU C/C++/Fortran Compiler version 4.2.0
- LAM (Local Area MultiComputer)/MPI version 7.1.3
- Standard development tools like make, emacs, vi, automake, autoconf, etc.

Primary hardware components include:

- One processor Intel Pentium 4 3.2GHz processor
- 1 GB of RAM
- 160 GB ATA disk drive. 7200RPM

4.3 Experimental Setting

In terms of the data sets themselves, we use synthetic and real-life data. Synthetic data, however, allows us much greater flexibility in setting the primary test parameters. In this respect, we note that multi-dimensional histograms tend to perform

extraordinarily well in uniform spaces (very easy to partition) and massively skewed environments (the data set deteriorates to a small number of singularities). Real data sets commonly used for DW testing (e.g., the ubiquitous weather data sets) actually have very little skew and largely resemble a uniform space. So while they tend to produce very nice results, their use often says relatively little about the performance of the histograms in more challenging environments. For this reason, in addition to the real data sets, we also produce our own data sets using the standard zipfian skew function. Specifically, we generate both a moderately clustered data set D_1 with zipfian skew = 0.4, and a much more densely clustered data set D_2 with zipfian skew = 0.8. Both D_1 and D_2 contain one million tuples and dimensions with cardinalities of 1000–50000.

As for the real-world data set we use in our experiments, this was defined in [32] and has already been adopted in different testing scenarios [59, 16, 45]. The original reports came from the National Meteorological Center for land stations and from the Comprehensive Ocean-Atmosphere Data Set (COADS) for ships. The data set contains over one million rows and includes (a) information related to cloud analysis, such as air temperature, pressure, winds, humidity, and visibility and (b) individual synoptic observations that have undergone some interpretation of the cloud information. We built a data subset from the original file with following characteristics:

- Number of data rows: 1015367.
- Number of Columns: 4.
- Number of Dimensions: 3.
- Cardinality per dimension: 97852, 36001, 1797.

Each test run consists of 1000 multi-dimensional queries randomly generated so as to encapsulate approximately 1% of the data space (query boxes have arbitrary shape and were built by using qGen, which will be further discussed in Section 4.4). Following the convention described in [5], we calculate the *average absolute error* as $error = \frac{1}{|W|} \sum_{q \in W} |q_{estimate} - q_{actual}|$ where $q_{estimate}$ is the estimate of the number of tuples in the result of q, q_{actual} is the actual number of tuples in the result of q and W is the query batch . Test results are averaged across a set of five runs. Finally, we note that all rK-Hist testing uses a θ ratio of 10%, as higher values rarely produce consistently superior results.

4.4 Query Generator

As mentioned above, all the techniques presented in this thesis were tested by using 1000 query boxes. Every query box contained either 1% or 5% of the total number of tuples. We generated those query boxes by using *qGen*, an in-house random-driven area-based query generator. In this section, we describe qGen and explain why this technique provides a fair testbed to perform our experiments.

Algorithm 4 describes the proposed approach. The technique works as follows. First, it defines a multidimensional point P_1 by using random values per every dimension. The boundary value of the random function is defined by the cardinality of each dimension. Next, it defines a second point P_2 by using the values of P_1. Having defined P_1 and P_2, qGen creates an initial query box and calculates its selectivity. If the selectivity of the new query box corresponds to the value we are looking for, a file is created storing the coordinates P_1 and P_2. Then, the algorithm will recursively create more boxes until it reaches N. Since generating query boxes that perfectly

Algorithm 4 qGen

Input: The number of queries N to be generated, the moving percentage per dimension α, the selectivity per query λ and an array with the cardinality per each dimension.

Output: N hyper-rectangles that cover a region with λ tuples.

1: **repeat**
2: Define a multidimensional point P_1 by using a random function.
3: Define a second point P_2 by using the values per each dimension found in the previous step plus $\alpha * cardinalityperdimension$.
4: **repeat**
5: Define a query box Θ with P_1 and P_2 and calculate its selectivity Ψ.
6: **if** Ψ is equal to λ ($\pm$ 0.5%) **then**
7: Create data file with the found coordinates P_1 and P_2.
8: Increment counter of query boxes in 1.
9: **else if** $\Psi > \lambda + 0.5\%$ **then**
10: Create a new set of points P_1 and P_2 from scratch.
11: **else if** $\Psi < \lambda$ - 0.5% **then**
12: Randomly select one of the points (P_1 or P_2).
13: Randomly select one dimension of the selected point.
14: **if** P_1 was selected **then**
15: Decrease the value of the dimension selected as follows $PreviousValue - (\alpha * CardinalityPerDimension)$.
16: **else if** P_2 was selected **then**
17: Increase the value of the dimension selected as follows $PreviousValue + (\alpha * CardinalityPerDimension)$.
18: **end if**
19: **end if**
20: **until** (Θ with selectivity λ is found) or (the boundaries of Θ are out of the dimension cardinalities).
21: **until** N query boxes are created

match with a given selectivity is time consuming, we allow a margin of ± 0.5% in order to speed up the generation of the complete set of query boxes. If the selectivity of the new query box is greater than λ (required selectivity) + 0.5%, the algorithm creates a new set of points from scratch. On the other hand, if the selectivity of the new query box is less than λ-0.5%, the algorithm a) selects one of the points (P_1 or P_2), and b) decreases or increases one of the dimension values of the selected point. Both selections are randomly made. The new query box is used as the input for the selectivity calculation and the process is repeated until we find the right query box or the boundaries exceed the cardinality values of every dimension.

qGen, by its nature as a random-driven query generator, allowed us to produce a large number of query boxes that favor no histogram technique, therefore making it very well suited to test arbitrary query ranges - which are indeed the most common in OLAP environments.

4.5 Actual Test Results

We begin by evaluating the estimation error for both D_1 and D_2. For each data set, we provide results for histograms constructed with 300 and 800 buckets. Note that all histograms store the same information internally so all use the same amount of memory. In Figure 4.1 (a) and (b), we see the results for the smaller 300 bucket histograms. For both data sets, across all the varying dimension count, a distinct advantage is clear for rk-Hist versus the three competing methods. In fact, for the moderately clustered set, rK-Hist produces approximately half the error generated by the next best method. Interestingly, the original (and fairly simple) hTree outperforms mHist and is competitive with the recent genHist algorithm. This result is perhaps

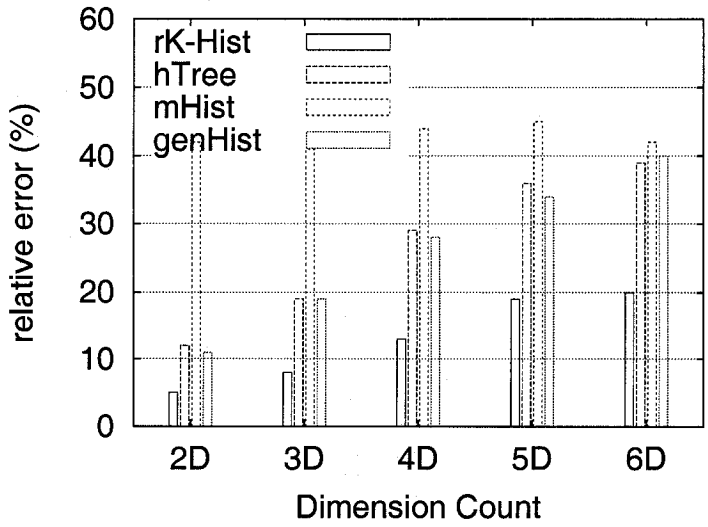
not surprising given the graphical partitioning results presented in Section 3.7. We note as well that mHist performs particularly poorly for the moderately skewed data set, which would certainly be a concern since D_1 is probably more indicative of many real world data sets than the more extreme D_2.

In contrast, Figure 4.2 provides the results for the larger 800 bucket histograms. As one would expect, estimation error improves noticeably. In fact, for the moderately skewed data, rK-Hist is able to produce errors in the range of just 2%-4% in two to three dimensions. This is extremely low for a multi-dimensional histogram. Moreover, none of the other methods is even close to this range.

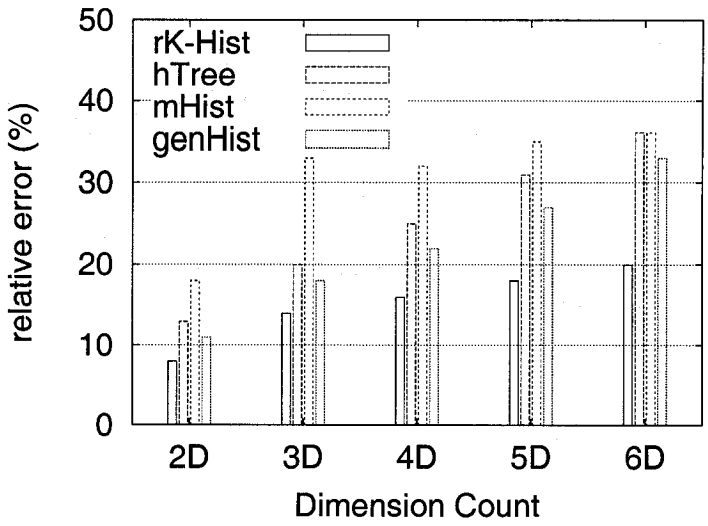
In Figure 4.3, we provide results for histograms constructed on real data sets. The experiment, in this case, uses a more realistic amount of buckets: 1,500 in Figure 4.3(a) and 2,500 in Figure 4.3(b) . The space required to store these buckets, considering the 4 dimension data set, is 53KB for 1500 buckets and 88KB for 2500 buckets. It is clear again that across all the varying dimension counts, a clear advantage is shown for rk-Hist versus the three competing methods.

It is important to note that, even for high dimensions, as shown in Figures 4.1, 4.2, and 4.3, rk-Hist histograms still produce better results than do hTree, mHist and genHist histograms. Overall, mHist has the highest error rate. This may be due to the way mHist recursively partitions the data set, it assigns too many buckets to the densest tuple clusters, and almost none to the rest of the data domain, degrading the overall histogram accuracy.

As noted, our query workloads are defined so that individual queries encapsulate about 1% of the total point space. Though this is in keeping with most previous research in the area, certain papers (e.g., genHist) have been evaluated against larger

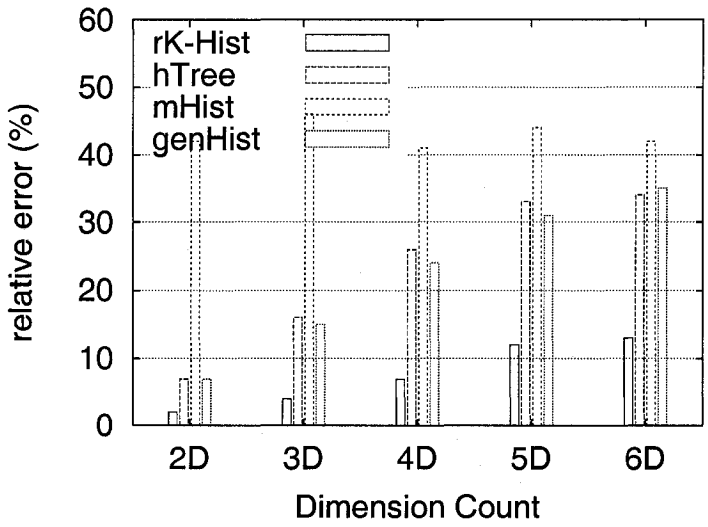


(a)

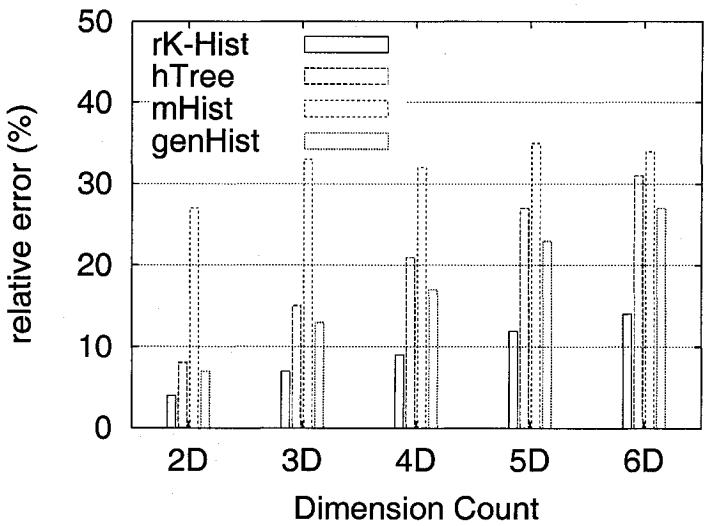


(b)

Figure 4.1: Estimation error for 300 bucket histograms for (a) zipf = 0.4 and (b) zipf = 0.8.



(a)



(b)

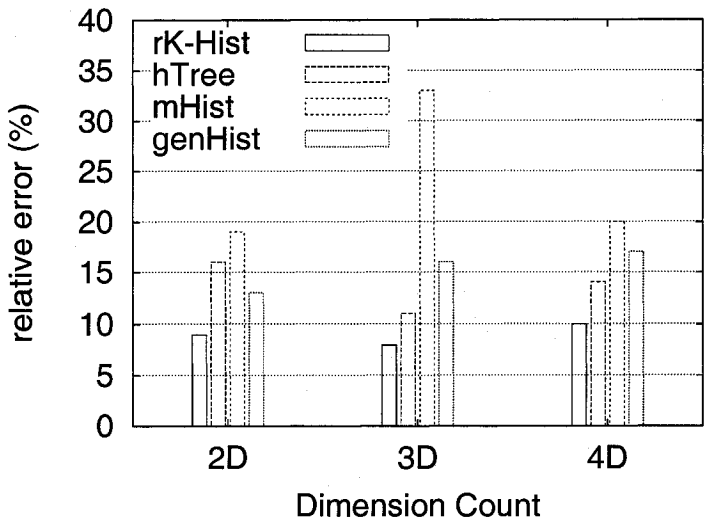
Figure 4.2: Estimation error for 800 bucket histograms for (a) zipf = 0.4 and (b) zipf = 0.8.

ranges. In general, this simplifies the problem and tends to produce lower (perhaps unrealistic) error rates. Nevertheless, in Figure 4.4(a) we provide a comparison on D_1 (800 buckets) between the effect of using 1% ranges versus 5% ranges in the query batches. As expected, all algorithms improve, with rK-Hist providing single digit error rates up to six dimensions. genHist, while not as impressive, also performs well in this test. In general, Figure 4.4(a) illustrates the robustness of rk-Hist across different workloads.

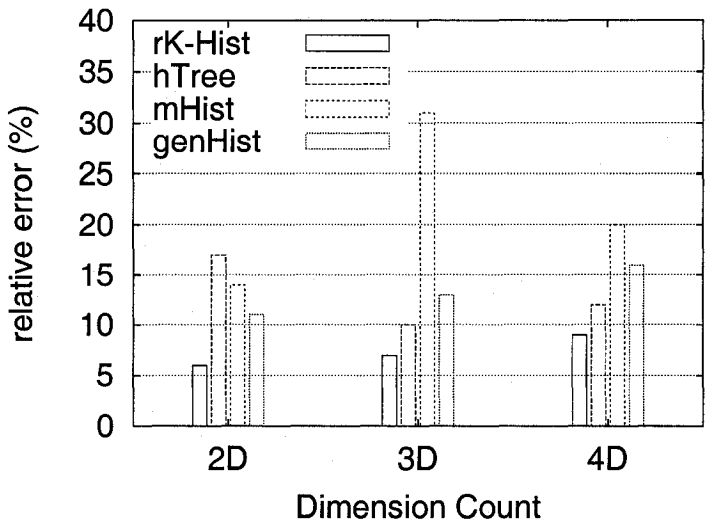
Figure4.4(b) looks more closely at the effect of increasing dimensions for rK-Hist. In this case, we examine rK-Hist as dimension count grows from 2 to 10, using the 800 bucket histogram and the D_1 data set. There is of course an obvious growth in estimation error as we move into high dimensions, with errors increasing by approximately 30%-50% with each additional dimension. We note, however, that even at 10 dimensions, rK-Hist is competitive with the rates that other techniques produce in 5-6 dimensions.

In Figure 4.5(a), we examine the impact of extending the naive r-tree histogram with the sliding window algorithm and the kU-partitioning. Again, we use the D_1 data set with 800 buckets for the comparison. There are two things to note. First, even the naive algorithm performs effectively relative to the numbers produced by the competing algorithms. Second, the estimation error for the naive algorithm is between 15% to 50% higher than rK-Hist, depending upon the dimensions count. So while the basic algorithm represents a reasonably good starting point, the improved partitioning produced by the sliding window and the kU measure results in a histogram that is vastly superior, particularly in the commonly utilized 2-4 dimension range.

We also studied the effect on histogram accuracy in terms of the storage space

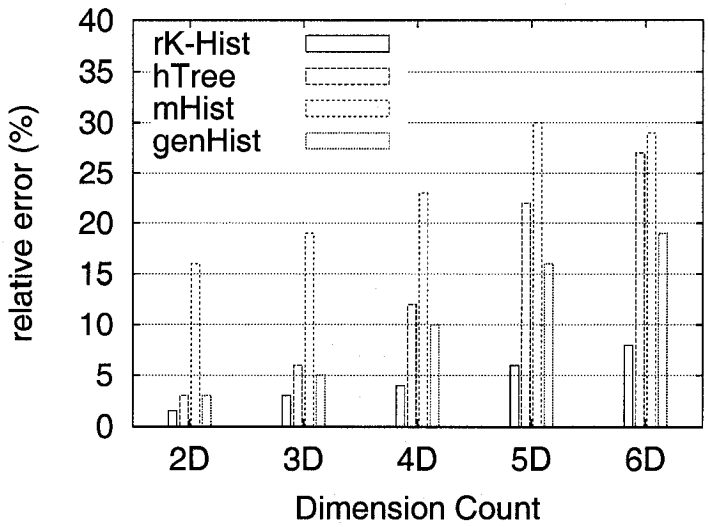


(a)

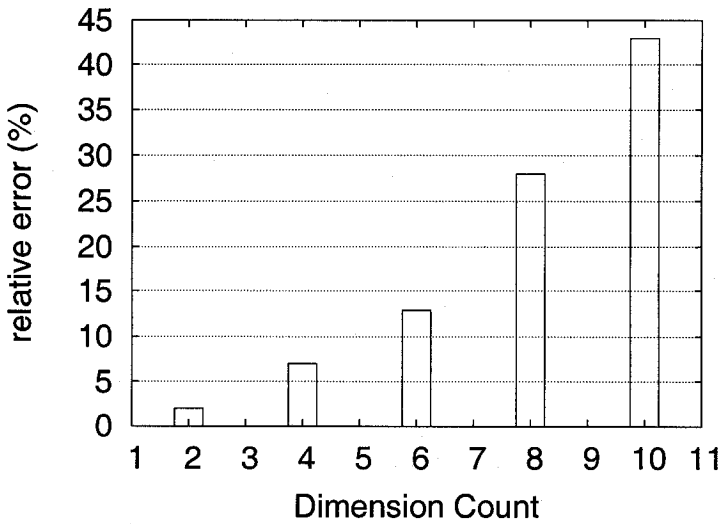


(b)

Figure 4.3: Estimation error on real data sets for (a) 1,500 buckets and (b) 2,500 buckets.

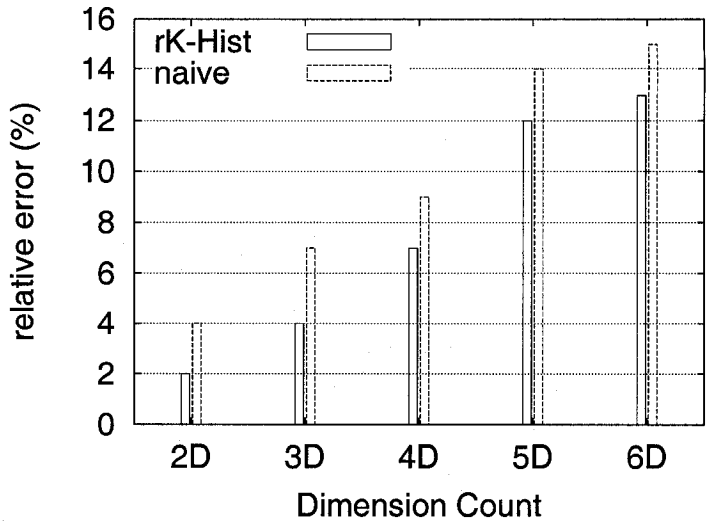


(a)

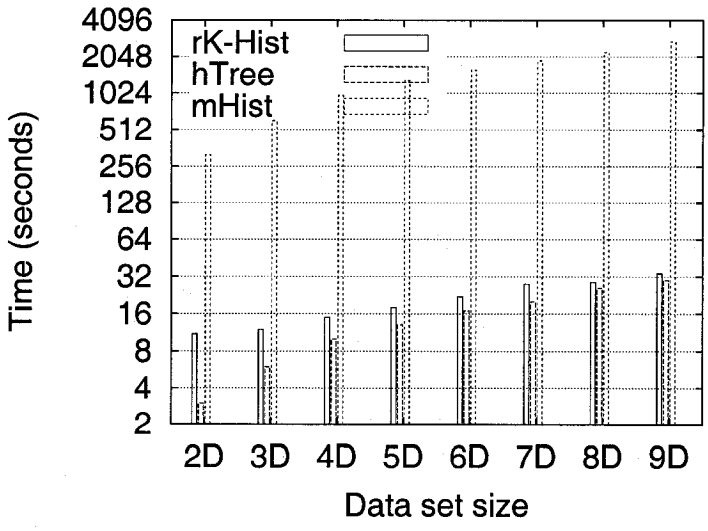


(b)

Figure 4.4: (a) 1% versus 5% query ranges (b) dimension counts ranging from 2 to 10.



(a)



(b)

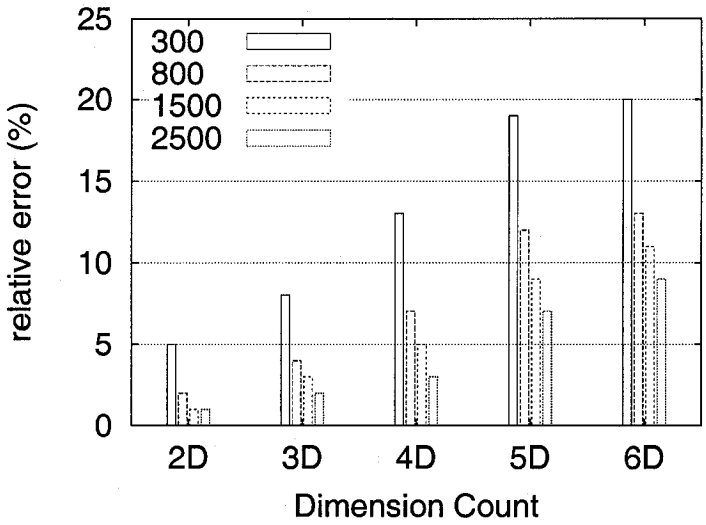
Figure 4.5: (a) rK-hist versus naive r-tree histogram (b) the relative construction cost of the four algorithms (logscale on the y-axis).

utilized. Figure 4.6 shows the normalized absolute error for the D_1, and D_2 data sets for varying the number of buckets per histogram (i.e., histogram size) and varying the number of dimensions. The errors are presented for histograms using from 300 to 2,500 buckets. As noted, rK-Hist scales quite well. As the number of buckets increases, the error decreases almost in the same proportion. For instance, in Figure 4.6 (a) when the number of buckets for the two-dimensional dataset increases from 300 to 800 (2.6 times), the error drops off by approximately 60%, from a normalized error of 5% to 2%. It is clear that when a very small error value is reached the improvement achieved by increasing the storage space is negligible (e.g., the error between 1500 – 2500 buckets in the 2-dimensional dataset of Figure 4.6 (a)).

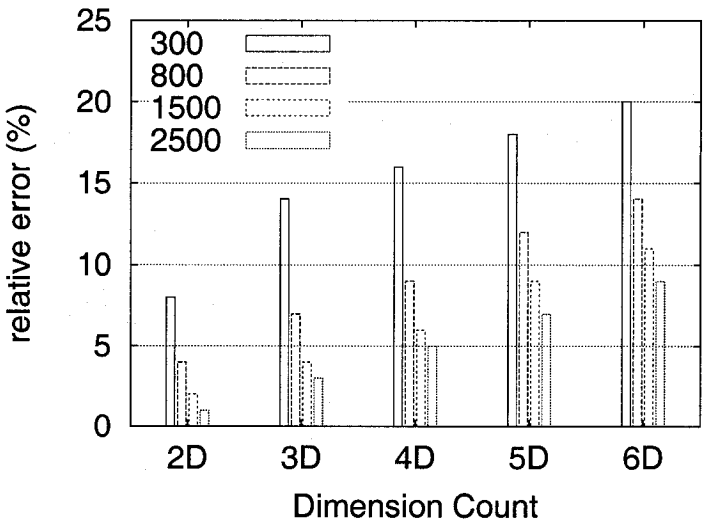
It is worth noting that all experiments allocate the same amount of memory for all histograms techniques. The size of individual buckets remains the same across all the approaches, requiring $2 * d * \beta$ values for the bucket boundaries plus β frequency values, where d refers to the number of dimensions and β to the number of buckets.

In Figure 4.7, we show the effect on histogram accuracy of the storage space for the real data set. The errors are presented for histograms using from 300 to 2,500 buckets. Again, we can see that (a) rK-Hist scales fairly well as the number of buckets increases. and (b) the results are consistent with those of Figures 4.6(a) and 4.6(b).

Finally, we compare the computational costs of three of the main algorithms across dimension counts from 2 to 9. Note that we do not include genHist in this test because, even with a logarithmic y-axis, the enormous times for genHist make it difficult to produce a meaningful graph. Figure 4.5(b) therefore illustrates the results for hTree, mHist, and rK-Hist. Not surprisingly, hTree is the most efficient method with its fairly trivial recursive grid partitioning. That being said, rK-Hist



(a)



(b)

Figure 4.6: rK-hist scalability for (a) zipf = 0.4 and (b) zipf = 0.8.

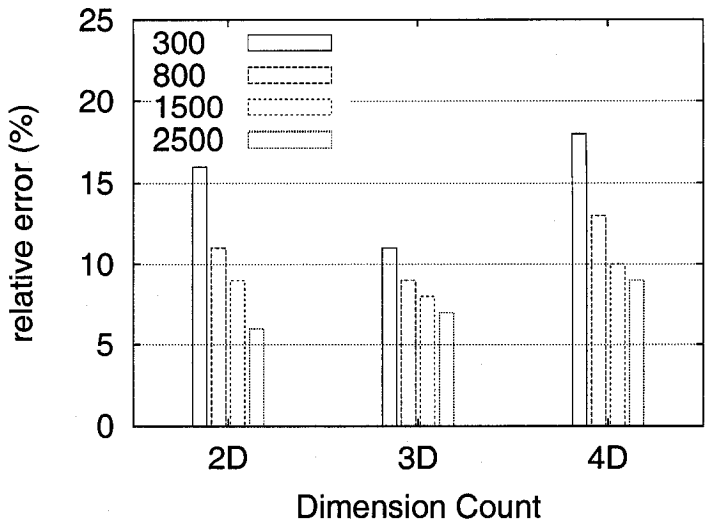


Figure 4.7: rK-hist scalability for the real data set.

is quite competitive, a result reflecting the efficiency of the sliding window and kU methods. mHist, by contrast, is almost two orders of magnitude more expensive by 9 dimensions. For comparative purposes, the genHist algorithm takes more than 12 hours to complete at nine dimensions. While it has been suggested that sampling could be used to reduce the cost, this approach would quite possibly lead to increased estimation error.

4.6 Conclusions

In this chapter, we presented multiple experiments that allow the reader (a) to gain a deeper understanding of the advantages of rK-Hist and (b) to compare and contrast our solution with those that represent the current state of the art.

Accuracy in the approximation, construction efficiency, and high scalability are some of rK-Hist's features that can be appreciated by analyzing the different results obtained using both synthetic and real-world data sets.

Chapter 5

Conclusions

5.1 Summary

For many years histograms have been utilized in database environments to produce concise representations of larger data distributions. The resulting estimates are fundamental to both query approximation and selectivity estimation. However, while errors in single dimension environments are quite impressive, multi-dimensional distributions have proven to be far more challenging. In this thesis, we present rK-Hist, an r-tree based histogram that exploits the Hilbert space filling curve to generate an initial space partitioning. It then uses a sliding window method, coupled with a new uniformity measure, to further improve the quality of the selectivity estimates. Experimental testing against a number of existing methods demonstrates consistent and significantly superior results in terms of estimation quality.

We consider the following three topics as the most important achievements in this thesis:

1. **Introduction of the kU metric.** Since we found that density can be a relatively poor measure of point distribution, a problem that is exacerbated as dimensionality increases, we present a new metric called k-uniformity (kU) that

minimizes the dead space between bucket points and more accurately measures the quality of tuple distribution.

2. **Implementation of rK-Hist.** We integrated the kU metric with a multi-dimensional r-tree partitioning model where the distribution of blocks is governed by the Hilbert space filling curve. Experimental results demonstrate that our new r-tree/kU histogram, rK-Hist, produces estimation errors that significantly improve upon the current state of the art.

3. **Integration of our work with the Sidera framework.** rK-Hist represents an extension of the previous research work carried out by Sidera's members. It works on an existing multi-processor platform that targets the relational database model (ROLAP). As we mentioned previously, by "piggy-backing" on top of an existing indexing model known as RCUBE, we are able to provide effective selectivity estimation in conjunction with powerful indexing functionality in multi-dimensional settings.

5.2 Future Work

The research described in this thesis represents the core of a robust multi-dimensional histogram model. However, the work undertaken to date also points to new research initiatives that would significantly extend the functionality of the current design. Below we identify a number of these possibilities.

- *Integration with Sidera's Query Engine.* It was made clear throughout this thesis that histograms are considered the most important component in the decision-making process of a query optimizer. We note that while rK-Hist

has shown superior results in terms of estimation quality when compared to a number of existing methods, the integration of rK-Hist inside the Sidera's query engine is essential. This would increase the robustness of the engine providing (a) a mechanism to support the optimizer's decisions while choosing among different query execution plans and (b) a component to be used with multiple operators supplying accurate selectivity estimates.

- *Logical and Physical Operators* . Operators, also known as iterators, are a set of fundamental building blocks that implement a single basic operation such as scanning data from a table, filtering or aggregating data, using a type of index or joining two data sets (e.g., by using hash, merge or nested loop strategies). Even though our current indexing model, RCUBE, has proven to be an excellent mechanism to retrieve data pages, we note, following the actual commercial DBMS and OLAP Servers, that a more extensive set of operators would give the optimizer a larger number of query plan possibilities to make good decisions and, in turn, would give rK-Hist a new set of building blocks on which to work.

- *Dynamic Generation of rK-Hist* . Despite the success of "static" histograms, there are still opportunities to increase their applicability and/or their effectiveness. Techniques like the ones presented in [50, 1], which propose to build histograms without examining the data sets, but rather by just analyzing query results (workload-aware histogram techniques) have proven to be quite efficient and could possibly be implemented on top of rK-Hist. Further research in this area would tell us if this is an advantageous strategy and at what point in time (e.g., after 10000 rows are either inserted or updated) the processing of such improvement would be appropriate.

5.3 Final Thoughts

Taken as a whole, the research presented in this thesis supports the notion of what we have called the *rK-Hist*. In keeping with the general philosophy of database management systems, our approach provides sophisticated computational functionality. Moreover, given the results presented in this thesis, and the fact that rK-Hist can be integrated so cleanly with one of the most common multi-dimensional indexing models, we believe the current method represents an extremely attractive option for selectivity estimation and approximate query answering in multi-dimensional environments. Finally, given the importance of the problem itself, both from a commercial and academic perspective, we are confident that the current research represents a significant and meaningful contribution to the database literature.

Bibliography

[1] Ashraf Aboulnaga and Surajit Chaudhuri. Self-tuning histograms: building histograms without looking at data. *Special Interest Group on Management Of Data, SIGMOD Record*, 28(2):181–192, 1999.

[2] B. Babcock and S. Chaudhuri. Towards a robust query optimizer: a principled and practical approach. In *Proceedings of the 2005 ACM Special Interest Group on Management Of Data, SIGMOD international conference on Management of data*, pages 119–130, New York, NY, USA, 2005. ACM Press.

[3] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The r-tree: an efficient and robust method for points and rectangles. *ACM Special Interest Group on Management Of Data, SIGMOD*, pages 322–331, 1990.

[4] J. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[5] N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: a multidimensional workload-aware histogram. *Special Interest Group on Management Of Data, SIGMOD Record*, 30(2):211–222, 2001.

[6] Francesco Buccafurri, Domenico Rosaci, Luigi Pontieri, and Domenico Saccà. Improving range query estimation on histograms. In *Proceedings of the 18th International Conference on Data Engineering, 26 February - 1 March 2002, San Jose, CA*, pages 628–638. IEEE Computer Society, 2002.

[7] cgmLab Portal. http://www.cgmlab.org.

[8] Surajit Chaudhuri. An overview of query optimization in relational systems. In *PODS - Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, pages 34–43, 1998.

[9] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *Special Interest Group on Management Of Data, SIGMOD Record*, 26(1):65–74, 1997.

[10] Ying Chen, Frank K. H. A. Dehne, Todd Eavis, and Andrew Rau-Chaplin. Pnp: Parallel and external memory iceberg cubes. In *International Conference on Data Engineering, ICDE*, pages 576–577, 2005.

[11] Ying Chen, Andrew Rau-Chaplin, Frank Dehne, Todd Eavis, D. Green, and E. Sithirasenan. cgmolap: Efficient parallel generation and querying of terabyte size rolap data cubes. In *International Conference on Data Engineering, ICDE*, page 164, 2006.

[12] S. Christodoulakis. Implications of certain assumptions in database performance evaluation. *ACM TODS*, 36(1):163186, 1984.

[13] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[14] ORACLE Corporation. *Oracle9i Database Performance Tuning Guide and Reference Release 2 (9.2).* Oracle9i Database Online Documentation, Release 2 (9.2), http://www.lc.leidenuniv.nl/awcourse/oracle/index.htm, 2002.

[15] Transaction Processing Performance Council. http://www.tpc.org/.

[16] David Cueva. *A Hilbert Space Compression Framework.* Master Thesis, Concordia University, 2007.

[17] Frank Dehne, Todd Eavis, Susanne E. Hambrusch, and Andrew Rau-Chaplin. Parallelizing the data cube. *Distributed and Parallel Databases Journal: Special Issue on Parallel and Distributed Data Mining*, 11(2):181–201, 2002.

[18] Frank Dehne, Todd Eavis, and Andrew Rau-Chaplin. Parallel multi-dimensional rolap indexing. In *CCGRID*, 2003.

[19] Frank Dehne, Todd Eavis, and Andrew Rau-Chaplin. Parallel querying of rolap cubes in the presence of hierarchies. In *DOLAP*, pages 89–96, 2005.

[20] Kalen Delaney. *Inside Microsoft SQL Server 2000.* Microsoft Press, 2001.

[21] Todd Eavis. *Parallel Relational OLAP.* PhD thesis, Dalhousie University, 2003.

[22] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. In *Proceedings of the eighth ACM Symposium on Principles of Database Systems*, pages 247–252, New York, NY, USA, 1989. ACM Press.

[23] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. *SIGACT-SIGMOD-SIGART symposium on principles of database systems*, pages 247–252, 1989.

[24] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[25] Sanjay Goil and Alok N. Choudhary. High performance multidimensional analysis of large datasets. In *DOLAP '98, ACM First International Workshop on Data Warehousing and OLAP, November 7, 1998, Bethesda, Maryland, USA, Proceedings*, pages 34–39. ACM, 1998.

[26] S. Guha, K. Shim, and J. Woo. REHIST: Relative Error Histogram Construction Algorithms. *International Conference on Very Large Data Bases, VLDB*, pages 300–311, 2004.

[27] D. Gunopulos, G. Kollios, V. Tsotras, and C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. *ACM Special Interest Group on Management Of Data, SIGMOD*, pages 463–474, 2000.

[28] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for olap. In *Proceedings of the Thirteenth International Conference on Data Engineering*, pages 208–219, 1997.

[29] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In Beatrice Yormark, editor, *Special Interest Group on Management Of Data, SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984.

[30] P. Haas and A. Swami. Sequential sampling procedures for query size estimation. *ACM Special Interest Group on Management Of Data, SIGMOD*, pages 194–205, 1992.

[31] P. J. Haas, J. F. Naughton, S. Seshadri, and A. N. Swami. Selectivity and cost estimation for joins based on random sampling. *Journal of Computer and System Sciences*, 52(3):550–569, 1996.

[32] Carole Hahn. Edited synoptic cloud reports from ships and land stations over the globe, 1982-1991. *Carbon Dioxide Information Analysis Center World Data Center-A for Atmospheric Trace Gases OAK RIDGE NATIONAL LABORATORY*, 1996.

[33] D. Hilbert. Ueber die stetige abbildung einer line auf ein flchenstck. *Mathematische Annalen*, 38(3):459460, 1891.

[34] C. Hoare. Algorithm 63 (partition) and algorithm 65 (find). *Communications of the ACM*, 4(7):321–322, 1961.

[35] W.H Inmon. *Building the Data Warehouse.* Wiley Comp., 1996.

[36] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 174–185, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[37] Yannis E. Ioannidis. The history of histograms (abridged). In *International Conference on Very Large Data Bases, VLDB*, pages 19–30, 2003.

[38] Yannis E. Ioannidis and Viswanath Poosala. Balancing histogram optimality and practicality for query result size estimation. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM Special Interest Group on Management Of Data, SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 233–244. ACM Press, 1995.

[39] H. Jagadish. Linear clusertering objects with multiple attributes. *ACM Special Interest Group on Management Of Data, SIGMOD*, pages 332–342, 1990.

[40] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 275–286. Morgan Kaufmann, 1998.

[41] Raj Jain and Imrich Chlamtac. The p algorithm for dynamic calculation of quantiiles and histograms without storing observations. *Commun. ACM*, 28(10):1076–1085, 1985.

[42] I. Kamel and C. Faloutsos. On packing r-trees. *Conference on Information and Knowledge Management, CIKM*, pages 490–499, 1993.

[43] Arnd Christian König and Gerhard Weikum. Combining histograms and parametric curve fitting for feedback-driven query result-size estimation. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 423–434. Morgan Kaufmann, 1999.

[44] Robert Kooi. *The Optimization of Queries in Relational Databases.* PhD thesis, 1980.

[45] Laks V. S. Lakshmanan, Jian Pei, and Yan Zhao. Qc-trees: an efficient summary structure for semantic olap. In *Proceedings of the 2003 ACM Special Interest Group on Management Of Data, SIGMOD international conference on Management of data*, pages 64–75, New York, NY, USA, 2003. ACM Press.

[46] J. Lee, D. Kim, and C. Chung. Multi-dimensional selectivity estimation using compressed histogram information. *ACM Special Interest Group on Management Of Data, SIGMOD*, pages 205–214, 1999.

[47] S. Leutenegger, M. Lopez, and J. Eddington. STR: A simple and efficient algorithm for r-tree packing. *International Conference on Data Engineering, ICDE*, pages 497–506, 1997.

[48] M. Muralikrishna and David J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In Haran Boral and Per-Åke Larson, editors, *Proceedings of the 1988 ACM Special Interest Group on Management Of Data, SIGMOD International Conference on Management of Data, Chicago, Illinois, June 1-3, 1988*, pages 28–36. ACM Press, 1988.

[49] S. Muthukrishnan, V. Poosala, and T. Suel. Partitionings in two dimensions: Algorithms, complexity, and applications. *International Conference on Database Theory, ICDT*, pages 236–256, 1999.

[50] Surajit Chaudhuri Nicolas Bruno and Luis Gravano. Stholes: a multidimensional workload-aware histogram. In *Proceedings of the 2001 ACM Special Interest Group on Management Of Data, SIGMOD international conference on Management of data*, pages 211–222, New York, NY, USA, 2001. ACM Press.

[51] G. Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36(1):157–160, 1890.

[52] Gregory Piatetsky-Shapiro and Charles Connell. Accurate estimation of the number of tuples satisfying a condition. In Beatrice Yormark, editor, *Special Interest Group on Management Of Data, SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 256–276. ACM Press, 1984.

[53] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 486–495, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[54] Viswanath Poosala, Yannis E. Ioannidis, Peter J. Haas, and Eugene J. Shekita. Improved histograms for selectivity estimation of range predicates. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM Special Interest Group on Management Of Data, SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996.*, pages 294–305. ACM Press, 1996.

[55] N. Roussopoulos, Y. Kotidis, and M. Roussopoulos. Cubetree: Organization of the bulk incremental updates on the data cube. In *Proceedings of the 1997*

ACM Special Interest Group on Management Of Data, SIGMOD international conference on Management of data, pages 89–99, New York, NY, USA, 1997. ACM Press.

[56] N. Roussopoulos and D. Leifker. Direct spatial search on pictorial databases using packed r-trees. In *ACM Special Interest Group on Management Of Data, SIGMOD*, pages 17–31, 1985.

[57] B. Schnitzer and S. T. Leutenegger. Master-client r-trees: A new parallel r-tree architecture. In *11th International Conference of Scientific and Statistical Database Management*, pages 68–77, 1999.

[58] T. Sellis, N. Roussopoulos, and C. Faloutsos. The r+-tree - a dynamic index for multidimensional objects. *International Conference on Very Large Data Bases, VLDB*, pages 507–518, 1987.

[59] Yannis Sismanis, Nick Roussopoulos, Antonios Deligianannakis, and Yannis Kotidis. Dwarf: Shrinking the petacube. In *Special Interest Group on Management Of Data, SIGMOD Conference*, 2002.

[60] U. Srivastava, P. Haas, V. Markl, M. Kutsch, and T. Tran. Consistent histogram construction using query feedback. *International Conference on Data Engineering, ICDE*, pages 39–51, 2006.

[61] Ahmad Taleb. *A Framework for the Manipulation of OLAP Hierarchies.* Master Thesis, Concordia University, 2007.

[62] N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic multidimensional histograms. In *ACM Special Interest Group on Management Of Data, SIGMOD*, pages 428–439, 2002.

[63] A. Rajaraman V. Harinarayan and J. D. Ullman. Implementing data cubes efficiently. In *Proceedings of the 1996 ACM Special Interest Group on Management*

Of Data, SIGMOD international conference on Management of data, pages 205–216, New York, NY, USA, 1996. ACM Press.

[64] J. Vitter, M. Wang, and B. Iyer. Data cube approximation and histograms via wavelets. *Conference on Information and Knowledge Management, CIKM*, pages 96–104, 1998.