Query Estimation Techniques in Database Systems

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

von
Diplom-Informatiker

Arnd Christian König

Saarbrücken,
im Dezember 2001

Contents

Abstract

The effectiveness of query optimization in database systems critically depends on the system's ability to assess the execution costs of different query execution plans. For this purpose, the sizes and data distributions of the intermediate results generated during plan execution need to be estimated as accurately as possible. This estimation requires the maintenance of statistics on the data stored in the database, which are referred to as *data synopses*.

While the problem of query cost estimation has received significant attention for over a decade, it has remained an open issue in practice, because most previous techniques have focused on singular aspects of the problem such as minimizing the estimation error of a single type of query and a single data distribution, whereas database management systems generally need to support a wide range of queries over a number of datasets.

In this thesis I introduce a new technique for query result estimation, which extends existing techniques in that it offers estimation for all combinations of the three major database operators *selection*, *projection*, and *join*. The approach is based on separate and independent approximations of the attribute values contained in a dataset and their frequencies. Through the use of space-filling curves, the approach extends to multi-dimensional data, while maintaining its accuracy and computational properties. The resulting estimation accuracy is competitive with specialized techniques and superior to the histogram techniques currently implemented in commercial database management systems.

Because data synopses reside in main memory, they compete for available space with the database cache and query execution buffers. Consequently, the memory available to data synopses needs to be used efficiently. This results in a *physical design problem* for data synopses, which is to determine the best set of synopses for a given combination of datasets, queries, and available memory. This thesis introduces a formalization of the problem, and efficient algorithmic solutions.

All discussed techniques are evaluated with regard to their overhead and resulting estimation accuracy on a variety of synthetic and real-life datasets.

Kurzfassung

Die Effektivität der Anfrage-Optimierung in Datenbanksystemen hängt entscheidend von der Fähigkeit des Systems ab, die Kosten der verschiedenen Möglichkeiten, eine Anfrage auszuführen, abzuschätzen. Zu diesem Zweck ist es nötig, die Größen und Datenverteilungen der Zwischenresultate, die während der Ausführung einer Anfrage generiert werden, so genau wie möglich zu schätzen. Zur Lösung dieses Schätzproblems benötigt man Statistiken über die Daten, welche in dem Datenbanksystem gespeichert werden; diese Statistiken werden auch als *Daten Synopsen* bezeichnet.

Obwohl das Problem der Schätzung von Anfragekosten innerhalb der letzten 10 Jahre intensiv untersucht wurde, gilt es weiterhin als offen, da viele der vorgeschlagenen Ansätze nur einen Teilaspekt des Problems betrachten. In den meisten Fällen wurden Techniken für das Abschätzen eines einzelnen Operators auf einer einzelnen Datenverteilung untersucht, wohingegen Datenbanksysteme in der Praxis eine Vielfalt von Anfragen über diverse Datensätze unterstützen müssen.

Aus diesem Grund stellt diese Arbeit einen neuen Ansatz zur Resultatsabschätzung vor, welcher insofern über bestehende Ansätze hinausgeht, als dass er akkurate Abschätzung beliebiger Kombinationen der drei wichtigsten Datenbank-Operatoren erlaubt: *Selektion*, *Projektion* und *Join*. Meine Technik basiert auf separaten und unabhängigen Approximationen der Verteilung der Attributwerte eines Datensatzes und der Verteilung der Häufigkeiten dieser Attributwerte. Durch den Einsatz raumfüllender Kurven können diese Approximationstechniken zudem auf mehrdimensionale Datenverteilungen angewandt werden, ohne ihre Genauigkeit und geringen Berechnungskosten einzubüßen. Die resultierende Schätzgenauigkeit ist vergleichbar mit der von auf einen einzigen Operator spezialisierten Techniken, und deutlich höher als die der auf Histogrammen basierenden Ansätze, welche momentan in kommerziellen Datenbanksystemen eingesetzt werden.

Da Daten Synopsen im Arbeitsspeicher residieren, reduzieren sie den Speicher, der für den Seitencache oder Ausführungspuffer zur Verfügung steht. Somit sollte der für Synopsen reservierte Speicher effizient genutzt werden, bzw. möglichst klein sein. Dies führt zu dem Problem, die optimale Kombination von Synopsen für eine gegebene Kombination an Daten, Anfragen und verfügbarem Speicher zu bestimmen. Diese Arbeit stellt eine formale Beschreibung des Problems, sowie effiziente Algorithmen zu dessen Lösung vor.

Alle beschriebenen Techniken werden in Hinsicht auf ihren Aufwand und die resultierende Schätzgenauigkeit mittels Experimenten über eine Vielzahl von Datenverteilungen evaluiert.

1 Introduction

> One of the symptoms of an approaching nervous breakdown is the belief that one's work is terribly important.
>
> – Bertrand Russell

1.1 Motivation

A central paradigm unique to database systems is the independence between the way a query is posed (specified by the user) and how it is executed (which is decided to the database system). For each query posed, the database system enumerates all possible ways of executing the query (or in case of very complex queries a suitable subset thereof), called *query plans*. Then the *optimal* plan, that is, the one with the lowest execution cost, is selected [GLSW93, GD87, GM93]. In order to select the optimal query plan for a query, the costs of query plans with respect to all affected resources (CPU time, I/O costs, necessary memory, communication overhead) have to be assessed accurately; these costs depend on the physical properties of the hardware the database runs on and the sizes of the intermediate results generated during the execution of the respective plan and, therefore, the data distribution in the queried base relations. A query plan itself is represented by a *physical operator tree* (Figure 1.1), with the cost of each operator being determined by the (size of the) data distribution resulting from the operator(s) at the child node(s). Whereas the properties of the underlying hardware are known at query-time, the relevant data sizes and distributions are not, making it necessary to *estimate* them, as determining them exactly would entail overhead similar to executing the queries themselves. In most cases, providing an estimation of the data size and distribution is sufficient, for only the ranks of the query plans with regard to their costs matter for choosing the correct plan. Providing this type of estimation is the topic of this thesis.

Besides *query optimization* there is a number of additional applications for this type of estimations:

- When executing queries in a data-mining or OLAP environment, these queries have the potential to monopolize a server. Therefore, knowledge of their running times is necessary to assign the proper priorities to each query or to decide whether spawning a long-running data-extraction query is worthwhile at all. Similarly, in scenarios with multiple servers, the cost estimations can be used to implement a load balancing mechanism, which distributes queries among the servers to improve overall system throughput [PI96]. Unlike the previous scenario, not only do the ranks of the query plans but also the absolute values of the estimated costs matter for these types of applications.

- For initial data exploration it may be sufficient for a database system or data mining platform to provide an approximate result for a resource-intensive query. Such approximations can be provided by the same techniques that estimate data distributions, for the approximate result can be generated from these.

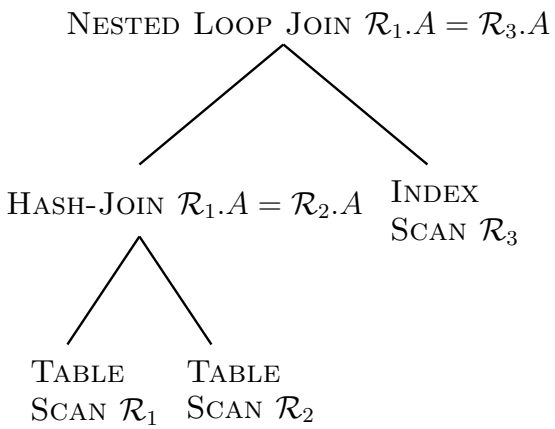

Figure 1.1: Example physical operator tree

- When global queries are issued to a mediator-based distributed information system, the mediator rewrites these into a combination of queries against physical information sources. The decision about which information sources to query and in which order to query them is dependent on the "information quality" [NL00] of each source, which can be estimated by techniques similar to the ones employed in the above scenario.
- Some useful operators that are not native to the SQL standard have to be mapped to expensive SQL queries to be executed by a database system. An example of this is "top-k" selection queries [DR99, BGM02], in which the user specifies target values for certain attributes, without requiring exact matches to these values in return. Instead, the result to such queries is typically a ranking of the k tuples that best match the given attribute values. Through the use of statistics on the data distributions present in the database, the fraction of the data examined to find the best k tuples can be reduced significantly [CG99].
- Finally, correct estimation of the size of views over a database plays a crucial role in the selection of materialized views [CHS01].

First attempts to provide this type of estimations were based on using standard assumptions on the distribution of data stored on the database, such as the *uniformity assumption* used in the System R optimizer [SAC+93]: all values between the lowest and highest value for an attribute are assumed to be present, with their frequencies distributed uniformly. Other assumptions considered include the *independence of attribute values* (two attributes A_1 and A_2 are independent, if for all values a the conditional probability of attribute A_1 having a certain value a given a value for attribute A_2 is equal to the probability of A_1 having value a) and *query uniformity* (i.e., all values are queried with the same frequency) [Chr84]. Since real-life data sets only rarely conform to these assumptions, this approach resulted in severely inaccurate cost estimation.

The general approach to providing the necessary estimations has been to store a statistical profiles of the data distributions stored in the database and in some cases of some intermediate query results as well. There exist a plethora of different techniques for representing this statistical profile, for which Gibbons et al. have coined the general term *"data synopses"* [GAB+98, GM99]. Each data synopsis represents a suitable approximation of the data distribution of one or more attributes of one of the base relations stored in the database. From these data synopses the necessary estimations are then computed at query-execution time.

There are a number of important issues to consider regarding the construction and use of data synopses.

- In all application scenarios outlined above, the data synopses need to support queries made from all three major logical database operators: *selection*, *projection*, and *join* (SPJ). This not only entails being able to provide an estimation for the result (size) of each query type, but also having effective means to bound the estimation error for *each* query type. Many of the existing approaches are geared towards minimizing the estimation error for range-selection queries only, and only very few techniques are capable of reducing the estimation error for join queries effectively.

- Whenever the base data in the database is made up of relations over more than a single attribute, the issue arises over which (combination of) attributes synopses should be constructed. If no synopsis over a combination of two attributes exists, information over the correlation between the values stored in these attributes is lost. For example, in a relation R with the attributes ORDER-DATE and SHIP-DATE, any query containing filter-conditions on both attributes (e.g. "`SELECT * FROM` R `WHERE` |SHIP-DATE−ORDER-DATE| ≤ 10") would potentially exhibit a significant estimation error. On the other hand, the accuracy of data synopses is generally adversely affected by to the number of attributes it spans. Therefore, choosing the optimal combination of synopses for a given relational schema, the data distributions in the base relations, and the query workload is a complex optimization problem. Because finding a "good" combination of synopses requires significant knowledge about the data distributions stored and the characteristics of the techniques used in connection with data synopses, the decision which synopses to construct should not be left to the database administrator, but instead has to be made by the database system automatically.

- In many cases it is not possible to optimize a query when it is first submitted to the database system (this is referred to as the query's *compilation time*), as some of the parameters which determine the exact query statement may not be instantiated yet. These parameters are then only specified immediately before the query is executed by the database (this is the query's *execution time*), so that any delay caused by the overhead of query optimization is noticeable to the user. Because the estimation is invoked (possibly multiple times) for every candidate plan (the number of which can be very large [Cha98]) at query execution time, the estimation process itself may not result in significant overhead. Therefore, data synopses are generally stored in main memory and thus have to be a small as possible, in order not to take away space otherwise reserved for data caches or buffers for intermediate query results. Consequently, the estimation accuracy relative to the amount of memory consumed becomes the critical measure when assessing the performance of data synopses. Furthermore, whenever multiple synopses are stored, the memory dedicated to data synopses overall has to be divided between them. This gives rise to a further optimization problem, which is closely connected to the question, which synopses to store overall.

The contribution of this thesis is to provide an overall framework addressing all of the issues discussed above.

1.2 Problem Description

This thesis is concerned with examining two problem areas in the applications discussed above. The first one is how to construct synopses for accurate estimation using as little storage and

computational overhead as possible. Based on these synopses the second problem area becomes how to automatically compute the optimal combination of synopses for a given set of relations, a (characterization) of the system workload, and limits on the amount of memory available for all synopses overall; this I refer to as the *physical design problem for data synopses*.

In the following, I will introduce the general approach for addressing both problems and describe by which criteria the proposed solutions will be evaluated. Throughout the thesis, I will describe the different requirements on and properties of data synopses using the query optimization as application scenario. However, whenever necessary I will describe the extensions necessary for other applications.

1.2.1 Synopsis Construction

All the outlined application scenarios require estimation of the result size for all major logical database operators (select-project-join). A number of approaches [SLRD93, CR94, LKC99, MVW98], provide a function $estimate_\Theta$ for each operator Θ, which takes the filter conditions of a given query and returns an integer representing the number of tuples of the result. For example, a range-selection estimator selecting all tuples in a interval $[a, b]$, with a, b chosen from the respective attribute's domain $\mathcal{D}$, is defined as

$$estimate_{range} : \mathcal{D} \times \mathcal{D} \mapsto \mathbb{N}.$$

Each of these estimators is then modelled as a function of two variables, resulting in a reduction of the estimation problem to a problem of the well-researched field of function estimation.

However, most real-life database queries consist of a combination of multiple operators, so the above approach infeasible: because now all possible combinations of operators correspond to separate *estimate* functions, the number of functions becomes too large. Instead, for each relation stored in the database, an approximate representation of the underlying data distribution is stored. Now, when executing a query operator, this operator is executed on the approximate data distribution. Any property of the approximate data distribution such as size (relevant for the approximation of the cost of join queries), the number of different attribute values (relevant for size estimation in queries with duplicate elimination or grouping), skew (relevant for providing sufficient memory for large join computation), etc. can be derived directly from the approximate data.

While this approach makes it possible to estimate virtually every query operator, it does not necessarily result in accurate estimation. I will discuss which requirements are necessary to ensure accurate estimation of the three major logical operators and how these requirements are reflected by different design choices for query estimation techniques.

Criteria for Success

In the query-optimization scenario, the benefit provided by a synopsis lies in the improvement in plan selection. However, this measure is heavily influenced by the particularities of both the query workload and the optimizer itself. Therefore, the *estimation accuracy* provided by a synopsis is used as a measure of its quality instead. Better estimation accuracy typically leads to better plan selection as well, but there is not necessarily a direct correspondence: in some situations, even extremely poor estimation may have no adverse effect on plan selection, if the ranking of the candidate plans remains unchanged.

The accuracy of estimation itself is important when using result-size estimation to estimate the overall running time of a resource-intensive query, in order to schedule it or decided whether

to spawn it at all, as the absolute running time is the deciding factor for this decision. For some types of decision-support queries, knowing the size of the final query result before issuing it is also useful in assessing whether a query returns compact enough information to be handled by a human user. If not, it may be necessary to add some aggregation or issue a different query altogether. In each case, the necessary information on query cost and result size need to be assessed as accurately as possible.

Because data synopses are stored in main memory, the accuracy of a synopsis has to be judged relative to the amount of memory its storage consumes. In addition to the memory overhead, the computational overhead required for (a) the construction and (b) the evaluation of a synopsis need to be considered as well.

1.2.2 Physical Design Problem for Data Synopses

The optimal combination of synopses is dependent on three factors: (a) the characteristics of the data stored in the database, (b) the current workload, and (c) the amount of memory available for synopses overall. Initially, all possible combinations of synopses that can be used to answer the given mix of queries are determined. Among them, the combination is selected that minimizes the estimation error for all attribute values selected by the given set of queries (with each error-term weighted by the number of times the corresponding attribute value is selected). Note that this does not only involve the selection of the optimal set of synopses, but also determining how the available memory is divided among them.

Criteria for Success

Estimating all possible combinations of synopses and their sizes results in additional overhead, which commercial database systems avoid by using much simpler schemes for synopsis selection. In most cases, only synopses over single attributes are employed, with all synopses of equal size (which severely limits the queries that can be estimated accurately in itself). In order to justify this additional overhead, the additional estimation accuracy gained through our more elaborate approach needs to be significant.

In addition, an algorithmic solution to the physical design problem would be a step towards eliminating synopsis selection from the responsibility of a database administrator. Since choosing a good set of synopses requires in-depth knowledge of the data and the techniques used for query estimation, and in addition this set needs to be periodically adapted to evolving workload properties, this task is not well-suited to human administration.

1.3 Contribution and Outline

The remainder of this thesis is divided into 5 chapters:

In Chapter 2, the design space for data synopses is discussed, with the different design alternatives illustrated in terms of previous approaches. With regard to most design alternatives, it is possible to identify one design choice as clearly superior; these choices are then incorporated into the design of my own approach to data synopses. In addition, based on this system of design alternatives, I construct a taxonomy for the plethora of techniques proposed in related work.

Chapter 3 describes how to solve the limited problem of query estimation for queries on a single attribute only. The salient property of the approach is that – unlike all other approaches to data

synopses – separate and independent approximations are used for the distribution of the attribute values and the distribution of their frequencies. Consequently, more memory can be assigned to the more difficult or important approximation problem. In both cases, the approximation is reduced to a problem of linear fitting, with the attribute values and their frequencies being approximated through linear spline functions. The resulting technique has been named *spline synopses*.

The chapter further elaborates the computation, storage, and use of spline synopses for query estimation. Special attention is given to join queries, because unlike all other operators, accurate approximation of the joining relations does not necessarily result in accurate join estimation. This leads to the introduction of special *join synopses* capturing the distribution of attribute values that appear in the join result. Finally, the efficiency and accuracy of spline synopses is evaluated in comparison to the current state of the art through experiments on various synthetic and real-life datasets. In the experiments, spline synopses consistently outperform the best histogram techniques, while being of identical computational complexity.

Chapter 4 then extends the previous approach to datasets and queries on multiple attributes. Here, the Sierpiński space-filling curve is used to map the multi-dimensional data to a linearly ordered domain, over which the approximation is then constructed. This approach leverages the advantages of one-attribute spline synopses (efficiency, fast computation, independent representation of attribute values and their frequencies) while avoiding the problems normally inherent to multi-dimensional synopses (distortion of the attribute-value domain, intractable overhead for synopsis construction). Again, the efficiency of the resulting synopses is evaluated through experiments. Again, spline synopses outperform histograms, while offering accuracy competitive with the best specialized techniques geared towards the limited class of range-selection queries only.

Chapter 5 finally discusses the physical design problem for data synopses, i.e., how to compute the optimal combination of synopses for given workload, data, and available memory. Initially, a formalization of the problem is developed through which the overall problem can be solved through the enumeration of all potential synopsis combinations and a minimization over the approximation errors computed during synopsis construction. This approach returns the best set of synopses as well as how much of the available memory is given to each of them. To reduce the number of synopsis combinations that need to be considered, the search space is pruned based on two general observations on the behavior of data synopses. Since these observation not only hold for spline synopses, but for a large class of data synopsis techniques, a limited form of the solution can be extended to these other techniques, as well. The practicability and potential impact of the proposed solution is evaluated on a subset of the queries and data of the *TPC-H* decision support benchmark.

Finally, chapter 6 offers a summary of the work and outlines future areas of research. Preliminary results of my work have been published in [KW99, KW00, KW02].

1.3.1 Limitations of This Work

The techniques described in this thesis are limited to numerical data (for both discrete and real domains). Although string data can be mapped to $\mathbb{N}$ using enumeration schemes, some of the operators specific to strings (such as prefix- and substring-matching [JNS99]) can not be estimated accurately when approximating this data using spline synopses (or any other of

the approximation techniques discussed in the next section). This is due to the fact that these operators require the approximation technique to maintain the correlation between adjacent occurrences of characters (or substrings). Because the approximation techniques used query optimization are geared towards representing other types of correlation, string data requires a different type of approximation, such as (approximate) suffix trees [NP00, JNS99, Ukk93]. How to reconcile both classes of approximation techniques for estimation of queries requiring both types of functionality is an open research problem at this time.

2 Related Work

> One of the problems of taking things apart and seeing how they work – supposing you're trying to find out how a cat works – you take that cat apart to see how it works, what you've got in your hands is a non-working cat. The cat wasn't a sort of clunky mechanism that was susceptible to our available tools of analysis.
>
> – Douglas Adams

This chapter gives an overview over work related to the topics of this thesis. When describing the different approaches it is necessary to identify their distinguishing characteristics concerning details of how relational tables are approximated. Therefore, it is first necessary to introduce some notation in Section 2.1. In Section 2.2 the the necessary concepts regarding query processing are briefly reviewed. Section 2.3 provides a compact introduction to relevant aspects of query optimization. In Section 2.4 all previous approaches for result (size) estimation are introduced. In order to provide a classification of these as well as our own work, I initially discuss the design space for query estimation techniques by introducing the most important design alternatives in this domain. Previous approaches to the less studied problem of physical design of data synopses are then discussed in Section 2.5.

2.1 Notation

Relational database systems store data in *relations*, which can intuitively be thought of as tables with a certain number of rows and columns. Each row is referred to as a *tuple*, each column is referred to as an *attribute*. An example relation is shown in Figure 2.1.

Employee ID	**Name**	**Age**	**Salary**
1076	Lennon	35	50.000
1077	Harrison	49	75.000
1078	Watters	18	15.000

Table 2.1: Example of a relation

In the following work, I consider a set of relations $\mathcal{R} = \{R_1, \ldots, R_n\}$. Each relation $R_i \in \mathcal{R}$ has att_i attributes $Att(R_i) = \{R_i.A_1, \ldots, R_i.A_{att_i}\}$. The values for each of these Attributes $R_i.A_j$ are defined over a *domain* $\mathcal{D}_{R_i.A_j}$ of possible values. The *value set* $\mathcal{V}_{R_i.A_j} \subset \mathcal{D}_{R_i.A_j}, \mathcal{V}_{R_i.A_j} = \{v_1, \ldots, v_n\}$ is the set of values for A_j actually present in the underlying relation R_i. The *density* of attribute $R_i.A_j$ in a value range from a through b, $a, b \in \mathcal{D}_{R_i.A_j}$, is the number of unique values $v \in \mathcal{V}_{R_i.A_j}$ with $a \leq v < b$. The *frequency* f_k of value v_k is the number of tuples in $\mathcal{R}$ with value v_k in attribute $R_i.A_j$. The *cumulative frequency* c_k of value v_k is defined as

the sum of all frequencies f_l for tuples $v_l < v_k$ (i.e., $c_k = \sum_{l=1}^{k-1} f_l$). The *spread* s_i of value v_i is defined as $s_i = v_{i+1} - v_i$ (with $s_n = 1$). The *area* a_i of value v_i is defined as $a_i = f_i \cdot s_i$.

Definition 2.1 (Data Distribution) :
Based on the above, a (one-dimensional) *data distribution* over a single attribute $R_i.A_j$ is defined as the set of pairs $\mathcal{T}_{R_i.A_j} = \{(v_1, f_1), (v_2, f_2), \ldots, (v_n, f_n)\}$, with f_k being the frequency of value v_k. When it is clear or immaterial which relation and attribute are referred to, the subscript $R_i.A_j$ is dropped when defining data distributions, domains, and value sets.

To give an example, consider a unary relation R_1 over the domain $\mathcal{D}_{R_1.A_1} = \mathbb{N}$ containing the following tuples $t_1 = 10, t_2 = 10, t_3 = 20, t_4 = 31, t_5 = 40, t_6 = 40, t_7 = 40, t_8 = 70, t_9 = 90$. Then $\mathcal{V}_{R_1.A_1} = \{10, 20, 31, 40, 70, 90\}$ and $\mathcal{T}_{R_1.A_1} = \{(10, 2), (20, 1), (31, 1), (40, 3), (70, 1), (90, 1)\}$.

Definition 2.2 (Joint Data Distribution) :
A *joint data distribution* over d attributes $R_i.A_{j_1}, \ldots, R_i.A_{j_d}$ is a set of pairs $\mathcal{T}_{R_i.A_{j_1},\ldots,R_i.A_{j_d}} = \{(v_1, f_1), (v_2, f_2), \ldots, (v_n, f_n)\}$, with $v_t \in \mathcal{V}_{R_i.A_{j_1}} \times \cdots \times \mathcal{V}_{R_i.A_{j_d}}$ and f_k being the number of tuples with value v_k. In the following we use the general term *data distribution* to refer to both single-attribute and joint data distributions.

The *cumulative frequency* c_i of a value $v_i = (v_i^1, \ldots v_i^d)$ is defined as $c_i = \sum_{\substack{k \in \{1,\ldots,|\mathcal{T}_{R_i.A_{j_1},\ldots,R_i.A_{j_d}}|\} \\ v_k^1 \leq v_i^1 \wedge \ldots \wedge v_k^d \leq v_i^d}} f_k$.

It is sometimes necessary to strip tuples of some of their attributes. For this, I use the operator $|_A$ with the set of attributes A describing the attributes, whose values remain after the operation. For example, if t_1 denotes the first tuple in table 2.1, then $t_1|_{\{R.Age,R.Salary\}} = (35,\ 50.000)$.

Some approaches to data synopses provide their estimation in the form of an approximate data distribution. In order to distinguish between the original data and its approximation, the approximation of the attribute value v_i is denoted by $\hat{v}_i$ and correspondingly the approximation of frequency f_i by $\hat{f}_i$. Thus, an approximate data distribution is written as $\hat{\mathcal{T}}_{R_i.A_j} = \{(\hat{v}_1, \hat{f}_1), (\hat{v}_2, \hat{f}_2), \ldots, (\hat{v}_n, \hat{f}_n)\}$.

Definition 2.3 (Frequency Distribution, Cumulative Frequency Distribution) :
Many approaches reduce the part of the estimation problem to a problem of function estimation by treating a (joint) data distribution as a function $f : \mathcal{V}_{R_i.A_j} \mapsto \mathbb{N}$ with $f(v_i) = f_i$. This f is referred to as the *frequency distribution function.*

The *cumulative frequency distribution function* is defined analogously, but with $f(v_i) = c_i$.

While most of the mathematical notation in this thesis is standard, I use a notation introduced in [Ive62] that allows simplifying most case statements. The idea is to enclose a true-or-false statement in brackets, which is treated as a mathematical expression with result 1 if the statement is true, and result 0 if the statement is false. For example, [12 *is prime*] = 0 and $[10^2 = 100] = 1$.

As many variables used in this thesis have both subscripts and superscripts (e.g., $x_k^1, \ldots x_k^n$) these can easily be confused with exponents. For this reason, all exponents (when applied to variables) are separated from these by parentheses (i.e. $(x)^2 = x \cdot x$).

2.2 Query Processing

Throughout this thesis, I will concentrate on the estimation of three major query operators *Selection*, *Projection* and *Join*. The reason for this focus is that most queries specified in the SQL database language (which is the standard adopted by all commercial database systems [O'N94]) can be specified as a combination of these 3 operators. To illustrate this, it is first necessary to explain the basic SQL syntax. The basic form of an SQL query is a follows:

`SELECT [DISTINCT]`	*result−attributes*
`FROM`	*relations*
`[WHERE`	*selection-condition* `]`
`[GROUP BY`	*group-attributes* `[HAVING` *selection-condition*`]]`

with *relations* specifying a subset of $\mathcal{R}$; the set

$$\mathcal{A} := \bigcup_{R' \in relations} \{R'.A_1, \ldots, R'.A_{|Att(R')|}\}$$

denotes all attributes present in these relations. Then *result-attributes* is a subset of $\mathcal{A}$ specifying the attributes present in the query answer. Each *selection-condition* specifies a filter on the tuples selected. Only tuples, for which the filter condition holds are included in the query answer. A *selection-condition* is of the form

$$\textit{selection-condition} := [[\texttt{NOT}]\ A_1\ \theta\ A_2\ [\{\texttt{AND} \mid \texttt{OR}\}\ \textit{selection-condition}]|True|False]$$

with $A_1 \in \mathcal{A}$, $A_2 \in \mathcal{A} \cup \mathcal{D}_{A_1}$ and θ specifying a binary comparison operator from $\{<, \leq, =, \geq, >, \neq\}$. Depending on the underlying domains, some limited mathematical expressions are also possible for A_1 or A_2. In addition, it is possible to use results of different SQL queries (which are then referred to as *sub-queries*) in place of A_2 in a selection-condition. This extension gives SQL the expressive power of first-order predicate logic. While our approach is capable of estimating this type of nested queries, I will focus on selection conditions limited to propositional logic.

It is further possible to specify *aggregation operators* in on single members of the *result-attributes*, that return an aggregate of the values in the respective column. The possible operators are (a) Count, (b) Sum, (c) Avg, (d) Max, and (e) Min, which specify (a) the number of values, (b) the sum of the values, (c) the average of the values, (d) the highest value and (e) the lowest value in the column. These operators may also be used in a *selection-condition* following a `HAVING` clause.

The result of a query specified through an SQL statement is another table, which contains all tuples present in the cross-product of the relations in *relations*, except for the tuples that do not satisfy the *selection-condition* and the attributes not present in *result-attributes*. Thus, SQL queries are operations on tables that also have results in the form of tables. The `DISTINCT` keyword is optional. It indicates that the table computed as an answer to this query should not contain *duplicates* (i.e., two copies of the same tuple). If `DISTINCT` is not specified, then duplicates are not eliminated; so the result can be a multiset. Finally, the result of a query may be partitioned into groups using the `GROUP BY` and `HAVING` clauses. Here the set *group-attributes*$\subseteq \mathcal{A}$ specifies which combinations of attributes determine the grouping, i.e., all tuples that have the same value for the attributes in *group-attributes* are assigned to the same equivalence class. The *selection-condition* in the `HAVING` clause can specify additional conditions on the groups selected in the final query result. Note that expressions appearing there must have a *single* value per group; therefore, aggregation operators are often used here.

A query can now be answered using selection, projection and join operators, I now define their semantics with regard to data distributions. Data distributions are equivalent to the relations or query results they represent in the sense that it is possible to reconstruct these from the data distribution(s) over all attributes that either (a) are selected by the query (i.e., they are specified in the *result_attributes* clause) or (b) appear in the query's filter conditions (i.e., they are contained in at least one *selection-condition*). Thus, by defining the semantics of an operator when applied to a data distribution, it is implicitly defined when applied to (part of) a relation as well. The operator semantics are defined as follows:

Selection: A selection over a selection-condition P is defined as an operation $Select[P]$ mapping a (joint) data distribution $\mathcal{T}_A$ over a set of attributes $A \subseteq \mathcal{A}$ to a (joint) data distribution $\mathcal{T}'_A$ with

$$Select[P](\mathcal{T}_A) = \{(v_i, f_i) \mid (v_i, f_i) \in \mathcal{T}_A \wedge P(v_i)\}.$$

A special sub-class of this query class are *range-selection queries*, which are characterized by every attribute having filter condition of type

$$P := A_1 \leq constant_1 \wedge A_1 \geq constant_2.$$

Note that with the exception of aggregation operators, all selection conditions in the `WHERE` clause express predicates on the attribute values present in the data distribution. This means that any technique capable of estimating arbitrary selection conditions has to store an (approximate) representation of the attribute value distribution.

Projection A projection of a data distribution $\mathcal{T}_A$ over a set of attributes $A \subseteq \mathcal{A}$ onto a set of attributes $A' \subset A$ is defined as follows.

$$Project(\mathcal{T}_A) = \{(v', f') \mid v' = v_i|_{A'} \wedge v_i \in \mathcal{V}_A \wedge f' = \sum_{\substack{(v_i,f_i)\in\mathcal{T}_A \\ v'=v_i|_{A'}}} f_i\}.$$

Intuitively this means that all tuples in $\mathcal{T}_A$, that are indistinguishable when considering only the attributes in A' are combined into one tuple in the result returned by the projection.

Join A join is an operation over two data distributions $\mathcal{T}_A = \{(v_1, f_1), \ldots, (v_n, f_n)\}$, $\mathcal{T}_{A'} = \{(v'_1, f'_1), \ldots, (v'_h, f'_h)\}$ and a simple *selection-condition* P of the type $P := [A_1\ \theta\ A_2 \mid True]$ with $A_1 \in A$ and $A_2 \in A'$. The result is a data distribution $\mathcal{T}_{A\cup A'}$ over the union of the attributes of $\mathcal{T}_A$ and $\mathcal{T}_{A'}$ defined as

$$Join[P](\mathcal{T}_A) = \{(v, f) \mid v \in \mathcal{D}_{A\cup A'} \wedge v|_A = v_i \in \mathcal{V}_A \wedge v|_{A'} = v'_j \in \mathcal{V}_{A'} \wedge v_i|_{A_1}\ \Theta\ v'_j|_{A_2} \wedge f = f_i \cdot f'_j\}.$$

In the special case of the *selection-condition* being *True*, the join result is identical to the cross-product of the tables corresponding to $\mathcal{T}_A$ and $\mathcal{T}_{A'}$. Throughout this thesis I concentrate on joins with the selection condition of the form $A_1 = A_2$ (this type of join is called *equi-join*), as it is the most common and also the most difficult to approximate (this is discussed in detail in Section 3.4.3). To denote the data distribution resulting from an equi-join between two relations $R_i,\ R_j$ over the condition $R_i.A_k = R_j.A_l$ I use the notation $R_i \overset{A_k=A_l}{\bowtie} R_j$.

For the sake of simplicity, assume that all selection conditions are connected via `AND` (this is not really a limitation as all queries involving `OR` can be reformulated as the union of multiple queries that only use `AND`). Now the evaluation of an SQL query as described above query can conceptually be described through the use of the above operators.

1. First, for all sub-conditions $A_1\ \theta\ A_2$ in the *selection-condition* with A_1 and A_2 from different relations, compute the **join** over the involved relations. Then compute the cross-product (using the join operator) of the intermediate result(s) and all tables in *relations* which were not included in the first pass of joins. The result of these steps corresponds to the cross-product over all relations in *relations*, after all tuples not satisfying the sub-conditions between relations have been removed.
2. From this intermediate result, **select** all the tuples satisfying the remainder of the *selection-condition*.
3. Through **projection**, remove all the attributes not appearing in *result-attributes*.
4. If `DISTINCT` is specified, all frequencies in the resulting data distribution are set to 1.

While the above approach will result in the correct answer, it is in most cases not efficient, for some tuples may be created only to be removed later. In order to make the execution of a query more efficient, the single join, project, and select operations are reordered through the query optimizer in such a way that the query result stays the same. To assess the efficiency of query execution by different plans (i.e., operator orderings), information on the size, (and in some cases also the data distribution) of relations and intermediate results is needed.

Definition 2.4 (Result Estimation, Size Estimation, Selectivity) :

Estimation techniques used in query optimization can be distinguished by what type of estimate they return. When given a query and data distribution the query is applied to, most techniques return an estimate for the size (expressed as the number of tuples) of the result. This is referred to as query result *size estimation*. The fraction of tuples from the original data still present in the query result is referred to as the *selectivity* of the query.

In contrast, other techniques provide an estimation of the result data itself (in the form of a data distribution $\mathcal{T}$). All necessary aggregate information, such as the size, can than be computed from $\mathcal{T}$. This is referred to as query *result estimation*.

2.3 Query Optimization

Different approaches to query optimization can be classified by *when* the optimization for a given query is performed.

Static or *compile-time* optimizers (for example, IBM's System-R query optimizer [SAC+93]) optimize a query at the time it is being compiled, thereby avoiding additional overhead at run-time, and potentially assessing a larger number of query execution plans, because the optimization time is less critical. This, however, is problematic as parameters important to the cost computation (such as available disk bandwidth, processing power and search structures, set cardinality or predicate selectivity) may change between compile-time and run-time, resulting in sub-optimal query plans. More significantly, in many cases, not all parameters determining a query may be known at compile-time (for example, when query parameters are specified remotely via host-variables of a web session). In addition, when queries are specified through tools that generate SQL statements directly, compile and execution time may fall together. As a result, static optimization can only be applied to a limited number of application scenarios.

An alternative approach is *dynamic query optimization* [Ant93, CG94, AZ96] (used in Oracle Rdb), which chooses the execution plan at run-time, thereby benefitting from accurate knowledge of run-time resources, host variables, and result sizes for sub-queries that have already been computed. However, in order to restrict the time necessary for optimization, a large part of the

plan is still optimized statically [Ant93]. Thus, good estimation of intermediate result sizes is still necessary. Another alternative is *parametric query optimization* [INSS97] which (at compile-time) generates the optimal plan for several potential values of the parameters on which the choice of the optimal execution plan is contingent and (at run-time) selects among them.

A hybrid approach called *dynamic query re-optimization* [KD98] (implemented in the Paradise database system [PYK+97]), collects statistics at key points in the execution of complex queries, which are then used to optimize the remainder of the query by either improving resource allocation or changing the execution plan.

A different approach to dealing with inaccurate result-size estimation at the level of the optimizer is presented in [SLMK01]. IBM's DB2 optimizer [GLSW93] monitors query executions and uses the feedback to compute adjustments to cost estimates and statistics, which are then used in future optimizations. These future optimization are again subject to monitoring, resulting in a feedback loop. As the adjustments themselves have very limited expressive power (mostly factors that are multiplied to the original estimates) they cannot replace query estimation techniques; so this approach does not invalidate the need for query estimation that is accurate in the first place.

2.4 Query Result Estimation

2.4.1 Design Space for Data Synopses

Recent research in the area of data synopses has led to a plethora of techniques providing the necessary functionality. In order to categorize all available techniques for query estimation, I now describe a number of different features characteristic for the different approaches:

Size Estimation vs. Result Estimation

As mentioned before, many techniques estimate only the size of the query result, providing the estimation for a query operator $\Theta \in \{\textit{selection}, \textit{projection}, \textit{join}\}$ by approximating a function with dimensionality equal to the number of parameters of Θ representing the corresponding frequency distribution function. For example, a range-selection over a single attribute $R.A$ query selecting all values in the interval $[a, b)$ can be represented by the 2-dimensional *cumulative frequency function* f. For example, to estimate a range selection over a single attribute $R.A$, the *cumulative frequency function* f, the approximation of which we refer to as $\hat{f}$. Now, the result size of a range query selecting all values in the interval $[v_i, v_j)$ is then estimated[1] as $Sel_{[i,j)} := \hat{f}(v_j) - \hat{f}(v_i)$. The estimations for other operators are then provided using a different function for each operator. Because estimations provided by these different functions cannot be combined without additional information on the data distribution, additional functions would have to be constructed and approximated to provide estimations for queries using multiple operators. Because the number of different operator-combinations occurring in real-life database applications

[1] For d-dimensional distributions the result size of a selection $(v_i^1 \leq R.A_1 \leq v_j^1) \wedge \ldots \wedge (v_i^d \leq R.A_d \leq v_j^d)$ is estimated using the d-dimensional function $\hat{f}$ approximating the cumulative frequency as follows:

$$Sel_{[i,j)} := \sum_{\substack{x^k \in \{v_i^k, v_j^k\} \\ 1 \leq k \leq d}} \left(\prod_{l=1}^{d} -1^{[x^l = v_i^l]} \right) \cdot \hat{f}(x^1, \ldots, x^d) \Bigg).$$

is large, this approach is not practical for query optimization. Furthermore, the approach fails when the filter-predicates are more complex than the range-selection example above.

Example 2.1: Consider the query

`SELECT DISTINCT` R.ORDER-DATE `FROM` R `WHERE` $|$SHIP-DATE$-$ORDER-DATE$| \leq 10$.

While approaches that approximate the (*cumulative*) *frequency distribution function* are capable of estimating the number of tuples for any possible value of SHIP-DATE or ORDER-DATE, they have no representation of the value domain itself, so the (number of) actual values in the result cannot be estimated.

The problems discussed in the above example stem from the fact that these types of techniques – when given a range of attribute values – offer accurate estimation of the number of tuples (or values, when Θ =*projection*) in that range; however they do not offer any information on which attribute values v_i are contained in $\mathcal{V}$ (the only way this information can be gathered is by checking whether a given value has a frequency different from 0, which typically leads to significant overestimation of the number of unique values). Summing up, one may say that techniques that approximate the (cumulative) frequency function only offer good approximation of the value frequencies f_i, but not of the values v_i themselves. To make this distinction explicit, we introduce the following notation.

Definition 2.5 (Value Frequency Approximation, Value Density Approximation) :

When approximating a given data distribution $\mathcal{T}$, the technique used for representing the attribute value frequencies $f_1, \ldots, f_{|\mathcal{T}|}$ is referred to as the *value frequency approximation*.

The technique used for representing the attribute values $v_1, \ldots, v_{|\mathcal{T}|}$ occurring in $\mathcal{T}$ themselves is referred to as the *value density approximation*.

In order to deal with the above problems, a number of approaches use techniques capable of estimating the query result itself and not only its size, meaning they store the approximate data distribution $\hat{\mathcal{T}}$ of a relation stored in the database and apply operators directly to this distribution. The result of each operation is another approximate data distribution, to which further operators may be applied. Because the approximation contains information on both the attribute values and their frequencies, all potential filter conditions can be applied.

The potential drawback of this approach is the fact, that providing estimation of the query result itself potentially requires more computational overhead, than the query-result size because potentially the entire data-distribution has to be materialized (consequently, the computational overhead for computing result size estimation is typically lower). Still, only techniques capable of estimating the query result as well are capable of dealing with the wide range of selection predicates occurring in real-life applications; thus, only query result estimation techniques are suitable candidates for the support of query-optimization.

Support for different Operators

An estimation technique needs to be able to support estimations for all major operators: selection, projection, join. However, offering the functionality of estimating a query operator result is meaningless if the technique is not capable of bounding the corresponding estimation error effectively.

For example, a large number of techniques are geared towards minimizing the variance in frequency approximation error only, i.e. synopsis construction selects the synopsis with the smallest

error-function $f\_err := \sum_{l=0}^{|\mathcal{T}|}(f_i - \hat{f}_i)^2$. This minimizes the estimation error for a class of equality queries(equality queries of type $R.A = x$ under the assumption that only values $x \in \mathcal{V}_{R.A}$ are queried) and is effective in minimizing the error for range-selection queries [IP95]. However, for join queries this approach fails, as accurate placement of the approximate attribute values is not in any way reflected in the error-function f_err. Therefore, the number (and approximate value) of attribute values finding a join-partner when estimating an equi-join query may be significantly off, leading to large estimation errors when computing a join over a data distribution (see Sections 2.2). Adding more memory to the corresponding synopses does not necessarily alleviate this problem, but may make it worse. In essence, this rules out approaches minimizing f_err only, when join queries need to be estimated.

Because most research in the area of data synopses has been geared towards improving range-selectivity estimation, a large number of approaches result in problems of the described nature for join and projection queries, the specifics of which will be discussed in Sections 4.1 (for projection queries) and 3.4.3 (for joins). In summary, data synopses need not only support operators, but have to be able to reduce the corresponding errors effectively.

High-dimensional Data

When estimating queries specifying filter conditions on more than one attribute of a relation, it becomes necessary to take into account the correlation between multiple attributes of the data. To illustrate this, consider the following example: Given a table Employees, containing the attributes Height (in centimeters) and Weight (in kilograms). Now consider the query

$$\texttt{SELECT * FROM } \textsc{Employees} \texttt{ WHERE } \textsc{Height} < 160 \texttt{ AND } \textsc{Weight} \geq 100.$$

Assuming one has exact knowledge over both the data distribution $\mathcal{T}_{\textsc{Height}}$ and the data distribution $\mathcal{T}_{\textsc{Weight}}$, and the number of tuples satisfying the condition on Height is 20% of the relation and the number of tuples satisfying the condition on Weight is 25% of the relation. Still, without any additional information on the correlation present between the attributes, the only knowledge gained about the selectivity of the entire query is that its result contains between 0% and 20% of the relation. Many early approaches use the *attribute value independence assumption* [SAC+93] at this point, estimating the selectivity of the composite filter as $25/100 \cdot 20/100 = 5/100$, which often leads to significant estimation errors (in the example, the selectivity is likely to be closer to 0%). Consequently, even data synopses over single attributes that perfectly estimate the underlying distributions do not permit accurate estimation for this type of correlation.

In order to provide accurate estimations for this type of queries, the data synopsis has to take into account the *joint data distribution* of the queried attributes. This entails preserving the amount of *correlation* present in the data.

Definition 2.6 (Correlation) :

Correlation is defined for data points that have two values of different quantity; it measures inhowfar the value of one quantity can be used to predict the value of another. In the above example, Height is strongly correlated to Weight, as the taller a person is, the higher typically his weight. If two quantities are independent (for example: salary and shoe size), they are said to have zero or no correlation.

As this thesis is concerned with ordinal and particularly cardinal numeric attributes, it is possible to use numerical measures of the present correlation. Depending on the context, the *linear correlation coefficient* [PTVF96] or the *Spearman rank-order correlation coefficient* [PTVF96] are used. The latter is more robust with regard to quantities of radically

different domains, as it does not measure the absolute values of the quantities, but their rank within the respective domains.

For query (size) estimation two types of correlation are of importance: (a) the correlation between the multi-dimensional attribute values and their frequencies (this correlation is also known as the *skew* of the value distribution) and (b) the correlation between the values different attributes themselves. While most techniques focus on the first problem, both of them are equally important for query estimation, which will be discussed in more detail in Section 3.4.3.

Handling of Data Updates

Because the data distributions approximated through data synopses are subject to change whenever the corresponding relations or data sets are updated, these changes have to be propagated to the data synopses as well. For this reason, some approaches offer the option of *incremental updates* which either use feedback obtained from observing the actual query results or monitor updates, deletions and insertions on the data itself and propagate these to the relevant synopses.

Unlike the previously discussed features, where it was possible to identify a 'correct' design alternative, there is a trade-off between flexibility towards updates and estimation accuracy. Because efficient compression (i.e., data reduction) schemes are generally very sensitive to changes in the data to be compressed, a single insertion, deletion or update may require a complete reconstruction of the corresponding synopses. This means that substantial overhead for synopsis construction is incurred whenever data is changed, which is prohibitive in real-life applications. On the other hand, data synopses that immediately integrate updates immediately with only negligible overhead generally offer significantly less data compression and thus typically worse estimation accuracy within the given memory space. Furthermore, when too many updates are performed, synopses based on this type of techniques may even require complete reconstruction.

As the memory required to store data synopses is normally a critical resource, it is often preferable to construct synopses that do not immediately integrate changes to the data and reconstruct these synopses during low system load whenever the data has changed significantly. While these synopses degrade over time when not updated, their accuracy is typically superior to begin with, thereby freeing up more space for cache and buffer memory.

2.4.2 Approximation Techniques

The final characteristic feature of approaches providing data synopses is the technique they use for data reduction. In the following, I introduce the most important classes of techniques.

Histograms

Histogram techniques [PSC84, Poo97, PIHS96, JKM+98, MD88, PI97, BCG01] approximate a data distribution $\mathcal{T}$ by partitioning it into disjoint intervals called *buckets*. In one-dimensional histograms (for one attribute) each bucket stores the sum of the frequencies of the attribute values contained in the corresponding interval, the number of distinct values in the interval and the highest attribute value. In multi-attribute histograms each bucket stores the sum of the frequencies of the attribute values contained in the interval, and for each attribute the lowest and highest value contained as well as the number of distinct values[2]. There exist a large number of

[2] A more space-efficient way of storing multi-dimensional histograms is introduced in [DGR01].

different histogram variants; the different classes can be classified by three[3] parameters:

Sort Parameter Specifies which domain is partitioned into histogram buckets (the choices being attribute values, frequency, and area).

Partition Constraint A mathematical constraint on the source parameter that uniquely identifies the rule by which the domain of the sort parameter is partitioned.

Source Parameter A parameter derived from the distribution $\mathcal{T}$ (the choices being attribute value spread, frequency, cumulative frequency, and area) used in the partition constraint to identify the unique bucketization.

A complete taxonomy of histogram variants can be found in [Poo97]. A histogram characterized in this way is generally described in the following notation:

Partition-Constraint (Sort-Parameter, Source-Parameter)

For example, the well-know *Equi-depth* histograms, which choose their buckets in such a way that the sum of the frequencies in each bucket is (approximately) equal among all buckets, are written as *Equi-Sum (V,F)*.

The most important class of single-attribute histograms use the partitioning constraint *V-Optimal*, meaning the buckets are chosen in such a way that the variance between the actual value and its approximation of the source parameter is minimized (i.e., if the source parameter is frequency, the error function $f\_err := \sum_{l=0}^{|\mathcal{T}|}(f_i - \hat{f}_i)^2$ is minimized).

It can be shown that *V-Optimal(F,F)* histograms [IC93, Ioa93, IP95] are optimal for estimating the result size of tree join, equality join and selection queries. However, as this type of histogram uses frequency as sort-parameter, information on the exact position of each attribute value has to be stored explicitly to use this histogram type for estimation, resulting in $O(n)$ storage overhead and rendering this type of histogram pointless for real-life applications, as the corresponding data distribution $\mathcal{T}$ itself could be stored using less space. The histogram types generally used in practice are *V-Optimal(V,F)*, *V-Optimal(V,A)* and *Max-Diff(V,A)* histograms, as they offer the best performance for a wide range of data distributions and queries [PIHS96]. Once the data distribution $\mathcal{T}$ is known, construction of an histogram with m buckets requires $O(m \cdot |\mathcal{T}|^2)$ overhead for V-Optimal histograms [JKM+98] and $O(\log_2 m \cdot |\mathcal{T}|)$ for Max-Diff histograms [PIHS96]. Producing a query result estimation requires $O(|\mathcal{T}|)$ cost when histograms are used; if only the result-size is of interest, the cost is $O(m)$.

When constructing histograms over multi-attribute distributions, the problem of partitioning the attribute value domain in such a way that the variance of a source parameter over all buckets in minimized becomes NP-hard [MPS99]. Thus, any variant of *V-Optimal* histograms results in unacceptable computational overhead in practice. In [MD88] a multidimensional variant of the well-know *Equi-depth* histogram is introduced, which partitions the attribute value domain one dimension at a time into buckets enclosing (approximately) the same number of tuples. In [PI97] several alternatives were introduced, with the most accurate one being multidimensional *Max-Diff(V,A)* histograms, which do not base their partitioning on the full multi-dimensional distribution, but choose the partitioning based on one-dimensional projections of the frequency distribution.

[3] In [Poo97] a 4th parameter Partition Class is used, but as all histograms proposed in the literature are of the class *serial*. This means that that buckets group elements from $\mathcal{T}$ that are contiguous in order and that there is no restraint on the number of elements that may be assigned to a bucket.

All the discussed multidimensional histograms are static in the sense that they cannot adapt themselves to changes in the data distribution without reconstructing the histogram completely. To overcome this, [AC99, BCG01] introduce a new approach to histogram construction in that they observe the query workload and the query results and leverage this information for histogram construction. The resulting histograms are constructed on the fly and continuously updated through new query feedback. This process not only reflects the data, but also locality in the access pattern of the workload, and the histograms of [BCG01] are more expressive in that they allow overlap between buckets. Therefore, the resulting histograms often offer better accuracy than static multidimensional histograms.

While histograms are typically less accurate than the techniques discussed below, they are easy to construct, maintain and use. All types of histogram can be used to estimate a query result in from of a data distribution [IP99] and may thus be used for estimation of virtually any query. For these reasons, histograms are used in virtually any commercial database system [Poo97]. However, no histogram type is capable of providing both accurate estimation of attribute value frequency and attribute value density (see Section 3.2), which is problematic for estimating queries requiring accuracy in both domains (see Section 3.4.3).

Hybrid Techniques for Real-Valued Domains

[GKTD00] introduces a variant of multidimensional histograms aimed at continuous value domains (typically the set of real numbers $\mathbb{R}$) called GENHIST (GENeralized HIStograms). The partitioning algorithm constructs progressively coarser grids over the data set, with each cell containing information on the density of values in the corresponding interval. The algorithm then removes points from (combinations of) dense cells, storing an approximation of the removed tuples in a bucket. Not all tuples in the area are removed, but only sufficiently many so that the density decreases to the level of the neighboring cells. Thus creation of a new bucket has a smoothing effect on the remaining data, making it easier to approximate.

The resulting histograms show good estimation accuracy for range selectivity queries, but offer no support for other query types (because of the real domain, the authors assume that it is unlikely that a given attribute value will appear more than once – obviously, this assumption is violated in most non-real domains). For this reason they are not grouped with other histogram approaches in this classification.

In [BKS99], a different approach for real-valued value domains is introduced. Here, the so-called *probability density function* (PDF) $F(x) = \int_{-\infty}^{x} f(t)dt$ over the frequency distribution function f is modelled using *kernel* [Sco92] functions as density estimators. The partitioning algorithm introduces bucket boundaries at points where the PDF is not sufficiently smooth to be approximated using a kernel. Inside the buckets themselves, kernel estimators are used. This approach is generalized to higher dimensions in [GKTD00]. Again, the resulting technique is not suitable for queries other than range-selectivity estimation.

Parametric Techniques

Parametric techniques (also known as curve-fitting or regression models) [Chr83, Lyn88, SLRD93, CR94] approximate the data distributions using a mathematical distribution function with a limited number of free parameters (e.g., coefficients of polynoms). Values for these parameters are then chosen to fit the actual distribution. If the model is a good fit for the distribution, this provides an accurate and compact approximation; however, since the shape of the distribution is generally not known beforehand, this is often not the case.

To gain some degree of flexibility, the approaches of [SLRD93, CR94] use a general polynomial function and apply least-squares fitting to choose its coefficients. To avoid both overfitting and the corresponding fitting problem from becoming *ill-conditioned* [PTVF96], the degree of the polynomials still has to remain small (both techniques use polynomials of degree less than 10). In addition, the approach of [CR94] leverages feedback from the actual result sizes to re-fit the coefficients. Thus it is able to adapt to changes in the value distribution as well as to locality in the ranges of the distribution being queried by improving the estimation of attribute values that are queried more frequently.

Generally, parametric techniques only estimate result sizes, and are not capable of providing full result estimation.

Discrete Transforms

A different approach to achieve the data reduction necessary for synopsis construction is to treat the frequency distribution function f as a signal on which a discrete transformation is performed (such as the *fast fourier transform*(FFT) [PTVF96], the *discrete cosine transform*(DCT) [Lim90] or one of many possible *wavelet transformation* [SRS96]) returning $|\mathcal{T}|$ coefficients representing the original signal. The constructed data synopses then only retains the most significant m coefficients, from which an approximate frequency distribution function $\hat{f}$ is reconstructed at estimation time.

This approach is used in [MVW98], where the wavelet-transform (using Haar wavelets) is used on the cumulative frequency distribution function. To select the the m most significant wavelet coefficients, four different *thresholding* schemes are introduced, the first one selecting the coefficients by the size of their *absolute normalized value*[4] (thereby minimizing the error-function $f\_err := \sum_{l=0}^{|\mathcal{T}|}(f(v_i) - \hat{f}(v_i))^2$ [SRS96]), and the other three greedy variants choosing coefficients by their overall effect on the chosen error metric. The key feature here is that for all thresholding schemes only the approximation of the value frequencies is examined. There is no approximation of the attribute-value distribution, resulting in a technique unsuitable for projection and join queries. While this effectively rules out this approach in the context of query optimization, it can still effectively be used for approximate answering of OLAP range queries on multidimensional data cubes [VWI98, VW99]. In [MVW00], a framework for handling incremental updates is added. The overhead for constructing the wavelet decomposition over a data distribution $\mathcal{T}$ is $O(|\mathcal{T}|)$, selecting m final coefficients through thresholding requires at most $O(|\mathcal{T}|\cdot\log_2|\mathcal{T}|\cdot log_2\, m)$ operations and the estimation of the size range query has overhead $O(\min\{m, \log|\mathcal{T}|\})$.

To overcome the limitations of the previous approach, [CGRS00] approximates an frequency distribution function f extended by adding the tuple $(v', 0)$ for all $v' \in \mathcal{D}, v' \notin \mathcal{V}$ to the data distribution. To exploit multi-dimensional locality, a different decomposition scheme is used; still the retained coefficients are selected by their absolute normalized values. While this approach makes it possible to process aggregation, projection, and join operators, the approximation of the attribute value distribution is only a by-product of the fitting of the frequency distribution, meaning that the estimation error for this type of queries is not minimized (this phenomenon is discussed in more detail in Section 3.4.3). The significant contribution of this work is that all operators are defined on the wavelet coefficients and also have wavelet coefficients as results, thereby eliminating the need to materialize intermediate results, and thus speed up the estimation of complex queries significantly.

[4] This means that a coefficient at resolution i is divided by $\sqrt{2^i}$. This way, coefficients that play a role in the reconstruction of a large number of frequencies have a correspondingly higher value.

The basic approach of [LKC99] is similar to [MVW98] but a discrete cosine transformation is used here instead, as this transformation results in better energy compaction than the Haar transform [Lim90, RY90]. The final coefficients are then selected via *geometrical zonal sampling* meaning that the coefficients are chosen by their geometric position, and not by their value. Because the DCT is linear transform[5], incremental updates are realized by computing the coefficients of the inserted/deleted data and adding/subtracting them to/from the stored coefficients. In contrast to [MVW00] any changes to the data do not result in a different set of coefficients being retained, but is only reflected in the value of the coefficients selected earlier. Because no information on the attribute value distribution is stored, this approach is limited to selectivity estimation for range-selection queries.

Probabilistic Models

The techniques introduced in [DGR01, GTK01] approximate the frequency distribution function by treating it as a probability distribution, which in turn is modelled through *probabilistic relational models* [DGR01] or *statistical interaction models* [GTK01]. As only the value frequency distribution is modelled, this results in the limitations discussed above; to make join approximation possible [DGR01] extend their model by introducing binary indicator variables for attribute values involved in equality joins which indicate whether or not the tuple is present in the join result.

2.4.3 A Classification of All Approaches

Summing up the different design alternatives discussed in Section 2.4.1, a data synopsis used for query estimation should have the following properties: It needs to be able to provide *estimation of the query result* (and not only its size) , be able to express the correlations found in *multi-dimensional data distributions*, support selection, projection, and join queries, and must be able to effectively minimize the estimation error for each query type. While the ability to update the synopsis incrementally can save the cost incurred by synopsis recomputations, it generally results in less efficient data reduction[6].

Table 2.2 characterizes all approaches discussed above with regard to these criteria. Each column signifies the ability of a given approach to provide the functionality stated at the top of the column. Whenever this is the case, I reference the paper(s) in which this is described. Whenever the same paper appears in multiple columns, the approach introduced in it supports the features of multiple columns. On the other hand, the features of different papers are typically not compatible. For example, although both [SLRD93, CR94] are parametric techniques, only [SLRD93] supports multi-dimensional data and only [CR94] supports incremental updates. The only exception to this rule are histogram techniques: all of them support all 3 major query operators (the corresponding techniques being discussed in [PIHS96]) and all of them can be used for result estimation (as discussed in [IP99]).

The most important fact expressed through this classification is that even though query estimation has been extensively studied, no single technique is capable of bounding the estimation error for selection, projection and join queries effectively (this will be discussed in more detail

[5]The DCT is linear in the following sense: Let F_c be the DCT, γ, δ scalars and x, y general k-dimensional vectors. Then it holds that $F_c(\gamma \cdot x + \delta \cdot y) = \gamma \cdot F_c(x) + \delta \cdot F_c(y)$.

[6]The fact that the histograms of [BCG01] outperform multi-dimensional *Max-Diff* histograms is not a contradiction to this, but merely a reflection of the suboptimal partitionings generated through the *MHIST-n* partitioning algorithm. With regard to estimation accuracy, all other discussed techniques also outperform *Max-Diff* histograms by a wide margin.

	SPJ-queries supported	*SPJ-queries optimized*	*Multidimensional distributions*	*Query Result Estimation*	*Incremental Updates*
Histograms	*all* ([Poo97])		[PI97, AC99] [BCG01]	*all* ([IP99])	[GMP97, AC99] [DIR00, BCG01]
Hybrid Techniques			[GKTD00]		
Parametric	[SLRD93, CR94]		[SLRD93]		[CR94]
Discrete Transforms	[CGRS00]		[VW99, CGRS00] [LKC99, VWI98]	[CGRS00]	[LKC99, MVW00]
Probabilistic Models			[DGR01, GTK01]		
Spline Synopses	√	√	√	√	

Table 2.2: Functionality of the different approaches

in Sections 3.2 and 3.4.3). In addition, only histograms and [CGRS00] are capable of providing query result estimation. Because both criteria are essential to accurate query estimation in the context of query optimization, query cost estimation is still considered an open issue in this context [CG94, Cha98].

As for the problem of capturing the correlation present in multidimensional distributions, a large number of approaches exist that are capable of maintaining multidimensional correlations. While histogram-based approaches suffer from a variant of the "curse of dimensionality" causing unacceptable computational overhead for some histogram types [MPS99] and rapid decline in estimation accuracy with rising number of dimensions (this is discussed in detail in Section 4.1), most other techniques offer accurate estimation of multi-dimensional range-queries. However, the only non-histogram techniques capable of estimating joins and projections are [SLRD93, CGRS00].

Spline Synopses [KW99, KW00] offer both query result and size estimation for selection, projection and join queries and consequently support estimation of all major query types. They can provide estimation for queries over multiple correlated attributes. In addition, they are capable of minimizing the estimation error for range-selections (see Section 3.5.2), projections (see Section 3.5.2) and join-queries (see Section 3.4.4) efficiently. While spline synopses are not capable to incorporate changes to the data distribution without recomputing the entire synopsis, this is a consequence of the efficiency of the techniques used forn data reduction. Spline synopses compute the *optimal* representation (i.e., the representation with the smallest error) for any dataset they approximate. As this is a global optimum over the entire data, small changes in the data may result in a completely different approximation, thereby causing the need for complete recomputation.

2.4.4 Sampling

Sampling-based techniques [HS92, HNSS95, Haa96, CMN98, GGMS96, LS95] provide result-size estimation by collecting a small random sample from the data and then scaling the result.

This approach works particularly well for certain types of aggregation: *count*, *sum* and *average*. Here the sample constitutes an unbiased estimator, with the expected value for performing the aggregation over the sample being equal to the result of the query itself. In addition, it is possible to leverage the *central limit theorem* [Nel95] to obtain confidence intervals for the estimated answer.

On the other hand, queries, where infrequent attribute values may significantly influence the query result (e.g., join estimation, as discussed in [CMN99]), or estimation of the number of distinct values for an attribute (as discussed in [CCMN00]) require a very large sample to guarantee a low estimation error. This is problematic, as the process of sampling the disk-resident data is slow. This is made worse by that fact that even to obtain a small sample, often a large number of database pages need to be read, as the tuples stored on a given page are often highly correlated and thus do not constitute a random sample. As an alternative, some systems create a random sample of the data called a reservoir sample [Vit85] and update it whenever updates, insertions or deletions occur [GMP97, GM98]. This reservoir can then be accessed much more efficiently. While the reservoir sample could be stored in main memory and serve as a type of data synopsis there, the size of such a sample is normally prohibitively large.

In summary, the applications of sampling and data synopses do not overlap significantly, as data synopses are mainly concerned with the approximate representation of data distributions, and sampling with the approximate acquisition of them. Thus, I do not compare our approach to sampling techniques in this thesis. There is, however, a chicken-and-egg relationship between sampling and data synopses. In many cases, when scanning an entire data set results in too much overhead, sampling is employed to obtain an approximate data distribution from which a data synopsis for the given relation is then constructed [CMN98]. In addition, a reservoir sample (on disk) can be used to incrementally reconstruct a data synopsis in main memory at much reduced cost [GMP97]. But data synopses can also aid sampling: when using sampling to estimate the result of a join query (without computing the join itself), partial statistics on the high frequency values are vital to the efficiency of the sampling algorithm [CMN99].

2.5 Physical Design of Data Synopses

The problem of selecting the optimal combination of synopses for a given combination of data and workload has not received much attention until recently. This is problematic for in virtually all application scenarios for data synopses more than one synopsis is needed. While the techniques described above have resulted in increasingly accurate estimation, to a large degree the efficiency thus gained is lost through the use of overly simple schemas for selecting synopses and allocating their memory.

The reconciliation of different synopses as well as dedicated synopses for join queries (a uniform random sample over a foreign key join) was initially considered in [AGPR99]. These ideas are generalized in this thesis, as [AGPR99] is limited to samples as base synopses and a data warehouse environment with a central fact table connected (via foreign keys) to the respective dimension tables. An extension of this approach to incorporate workload information (in the form of access locality) can be found in [GLR00], but it is also limited to the above application scenario.

The reconciliation problem for spline synopses was first discussed in our own work [KW00], where a dynamic programming approach is proposed to minimize the error for a given set of synopses. However, this work offers no solution regarding which set of synopses to construct and does not take into account the characteristics of the workload. A similar approach for histograms was proposed in [JJOT01], extending [KW00] by offering heuristics that reduce the overhead of the dynamic programming problem.

[CN00] considers a limited version of the problem: a set of synopses for query optimization are selected, based on whether or not they make a difference in plan selection. However, the approach is limited in a number of ways. Most importantly, synopsis selection is a series of yes-

or-no decisions, with no consideration of the effect that variations of the size of a synopsis may have. This also has the consequence that the overall memory allotted to the selected synopses is utilized in a sub-optimal way. Furthermore, there is no consideration of special, dedicated (join-) synopses which do not constitute a (sub-)set of the attributes of a single relation.

The physical design problem for data synopses resembles the index selection problem for physical database design, which has been intensively studied [FST88, CWBC95, RS91, CN99]. However, there are a couple of fundamental differences between these two problems that make the synopses problem much more difficult: (1) a synopsis not only chooses an attribute combination on which it maintains statistics, but its quality is highly dependent on the memory allocated to it (e.g., for a certain number of buckets in a histogram), and (2) a synopsis on a proper subset of attributes over which another synopsis is already maintained can be advantageous because of different approximation accuracies for different query types, whereas with B-tree indexes an index on a subset of an already indexed attribute combination would (to a first approximation) be superfluous.

3 Approximation of a Single Attribute

> Size matters not. Look at me. Judge me by my size, do you?
>
> – Yoda

In this chapter I address, the problem of building an approximation for a data distribution $\mathcal{T} = \{(v_1, f_1), \ldots, (v_n, f_n)\}$ over a single attribute. The approach is similar to histogram techniques in that the underlying value domain $\mathcal{V}$ is partitioned into buckets, which store a summary of the data points contained in them. The way this is done in histograms has significant drawbacks: the partitioning of $\mathcal{V}$ via buckets determines the quality of approximation for both attribute value frequency and value density. Thus, choosing a partitioning that results in a good approximation of both domains is often impossible, as these optimization goals are in conflict. Histograms therefore are generally aimed at minimizing the estimation error for either the attribute value frequencies (all histograms with source-parameter *Frequency*) or attribute value density (source-parameter *Values*)[1].

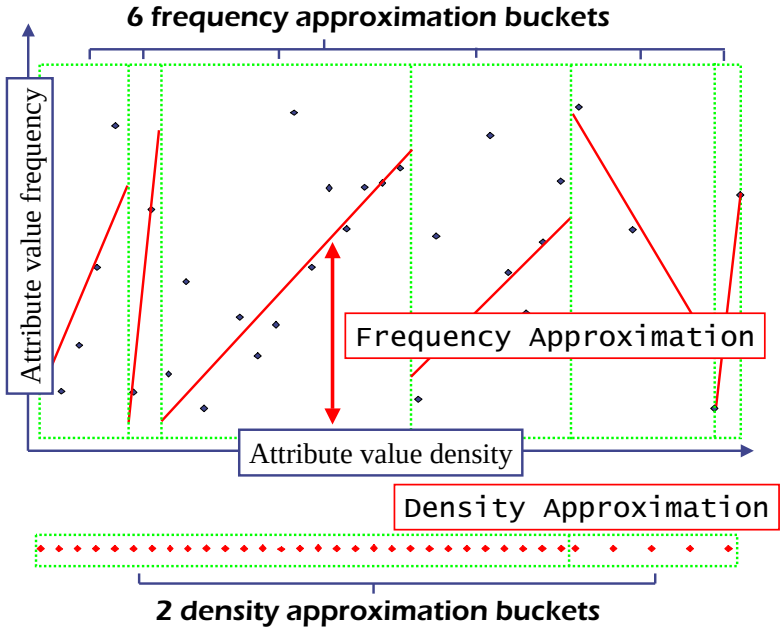

Figure 3.1: Example of a spline synopsis

Consequently, spline synopses construct two different, independent approximations (and thus

[1]Histograms with source parameter *Area* use an error function that aims to reconcile the approximation of both domains, but merely place bucket boundaries at the points with the largest gaps in *Area*, thereby often failing to construct a good partitioning for either domain.

partitionings) for capturing the frequency and the density distribution of an attribute (see Figure 3.1). Not only does this eliminate the above problems, but as the number of buckets for each distribution is not fixed a priori, it also allows us to assign more memory to the more difficult (or more important) of the two approximation problems.

The construction of the approximation of attribute-value frequencies is described in Section 3.1 and the approximation of attribute-value density in Section 3.2. Section 3.3 then describes how to combine the two different approximations and how to choose their sizes for a limited amount of available memory. In Section 3.4, I then discuss how to use the resulting spline synopsis for estimation. It turns out that accurate estimation of join queries requires additional information. This leads to an additional type of dedicated *join synopsis*, which is discussed in Sections 3.4.3 – 3.4.6. The chapter concludes with an experimental evaluation of join synopses in Section 3.5.

In the following, it is assumed that there is a limit on the number of buckets, M, of the synopsis representing $\mathcal{T}$. The number of buckets for the two approximations is assumed to be set at m (for frequency approximation) and m' (for density approximation) with $M = m + m'$. It will be shown that by generating the optimal partitioning for a given number of buckets, the optimal partitioning for all smaller numbers of buckets is also generated at negligible additional cost. This property is exploited in Section 3.3.2, where I discuss how to choose the values of m and m' in adaption to the difficulty of the corresponding approximation problems.

3.1 Approximation of the Attribute-Value Frequencies

To approximate a given value-frequency distribution $\mathcal{T}$, the (observed) value set $\mathcal{V}$ is partitioned into m disjoint buckets, $b_i = [v_{low_i}, v_{high_i})$ in the following manner, where low_i and $high_i$ denote the *subscripts* of the values from $\mathcal{V}$ (i.e., not the actual values) that form the left and right bounds of the (left-side closed and right-side open) value interval covered by the bucket:

$$\forall i \in \{1, 2, \ldots, m-1\} : high_i = low_{i+1} (low_1 = 1,\ high_m = n + 1). \tag{3.1}$$

For the sake of a simple, uniform notation, $\mathcal{T}$ is extended by an artificial tuple $(v_{n+1}, 0)$ with $v_n < v_{n+1}$.

Unlike histograms, spline synopses approximate the frequency in an interval by a linear function, resulting in a linear spline function [dB78, Die93] over the m buckets. Because my interest is in the compact representation of distributions and not in the features more advanced forms of splines offer (smoothness, differentiability), I have chosen simple linear spline functions for this task. In contrast to previous approaches to spline-based histograms [PIHS96] these are not required to form a continuous approximation of $\mathcal{T}^{\mathcal{C}^+}$, thereby simplifying the problem of finding the optimal partitioning (see Subsection 3.1.3) and increasing the resulting approximation accuracy as well.

Using a linear function $f_i(x) = \alpha_i \cdot x + \beta_i$ to approximate the frequencies for the values x in a bucket leads to an improvement in accuracy, depending on the *linear correlation* [PTVF96] of the data within a bucket. First, I define $\bar{v}_{[low,high)} := \frac{1}{high-low} \sum_{l=low}^{high-1} v_l$ as the *average attribute value* within $[v_{low}, v_{high})$; analogously, I define the *average frequency* $\bar{f}_{[low,high)} := \frac{1}{high-low} \sum_{l=low}^{high-1} f_l$. The *linear correlation* for interval b_i is then defined as

$$r_{[low_i,high_i)} := \frac{\sum_{l=low_i}^{high_i-1} (v_l - \bar{v}_{[low_i,high_i)})(f_l - \bar{f}_{[low_i,high_i)})}{\sqrt{\sum_{l=low_i}^{high_i-1} (v_l - \bar{v}_{[low_i,high_i)})^2} \sqrt{\sum_{l=low_i}^{high_i-1} (f_l - \bar{f}_{[low_i,high_i)})^2}} \tag{3.2}$$

For each interval b_i, $r_{[low_i,high_i)} \in [-1,1]$. In traditional histograms, the frequency in a bucket b_i is approximated by $\bar{f}_{[low_i,high_i)}$. Using the sum of the squares of the error at each frequency as an error metric, this results in the error for bucket b_i if a histogram were used:

$$histogram\_err_{[low_i,high_i)} = \sum_{l=low_i}^{high_i-1} (f_l - \bar{f}_{[low_i,high_i)})^2.$$

In a spline-based synopsis, using the same error metric the estimation error can be written as:

$$spline\_err_{[low_i,high_i)} = \sum_{l=low_i}^{high_i-1} (frq_i(v_l) - f_l)^2 \tag{3.3}$$

Equation 3.3 is minimized to determine the values for α_i and β_i. At its minimum, derivatives of $spline\_err_{[low_i,high_i)}$ with respect to $\alpha_i,\ \beta_i$ vanish:

$$\frac{\partial spline\_err_{[low_i,high_i)}}{\partial \alpha_i} = 0 \quad \text{and} \quad \frac{\partial spline\_err_{[low_i,high_i)}}{\partial \beta_i} = 0$$

Solving these equations for $\alpha_i,\ \beta_i$ results in

$$\begin{aligned} \beta_i &= \frac{\left(\sum_{l=low_i}^{high_i-1}(v_l)^2\right)\left(\sum_{l=low_i}^{high_i-1} f_l\right) - \left(\sum_{l=low_i}^{high_i-1} v_l\right)\left(\sum_{l=low_i}^{high_i-1} v_l \cdot f_l\right)}{\sum_{l=low_i}^{high_i-1}(v_l)^2 - \left(\sum_{l=low_i}^{high_i-1} v_l\right)^2} \text{ and} \\ \alpha_i &= \frac{\left(\sum_{l=low_i}^{high_i-1} v_l \cdot f_l\right) - \left(\sum_{l=low_i}^{high_i-1} v_l\right)\left(\sum_{l=low_i}^{high_i-1} f_l\right)}{\sum_{l=low_i}^{high_i-1}(v_l)^2 - \left(\sum_{l=low_i}^{high_i-1} v_l\right)^2}. \end{aligned} \tag{3.4}$$

By inserting the above terms for $\alpha_i,\ \beta_i$ into Equation 3.3, it is possible to express the error for this type of estimation as a function of the error in histograms and the linear correlation $r_{low_i,high_i)}$ of the values in b_i:

$$spline\_err_{[low_i,high_i)} = (1 - r_{[low_i,high_i)}{}^2) \cdot histogram\_err_{[low_i,high_i)}. \tag{3.5}$$

Obviously, $spline\_err_{[low_i,high_i)}$ is always less than or equal to $histogram\_err_{[low_i,high_i)}$. However, because of the separation of frequency and density approximation, the storage requirements for histograms and spline synopses cannot be compared directly. The resulting trade-off will be further examined by means of experiments in Section 3.5. Summing up the error over all buckets in the synopsis, the overall error becomes:

$$spline\_err = \sum_{i=1}^{m}\left((1 - r_{[low_i,high_i)}{}^2) \cdot histogram\_err_{[low_i,high_i)}\right). \tag{3.6}$$

3.1.1 Fitting the Frequency Function within a Bucket

For the derivation of this basic building block suppose it is assumed that the boundaries of a bucket are already fixed. For each bucket $b_i = [v_{low_i}, v_{high_i})$ the linear approximation $frq_i(x) = \alpha_i \cdot x + \beta_i$ of the attribute frequency that minimizes the squared error in Equation 3.3 has to be computed. As the Formulas 3.4 are suceptible to roundoff error, I use a numerically more stable method for the computation of the final values of the coefficients.

Using Definition 3.3, finding frq_i that minimizes the error becomes a problem of *linear least squares* fitting [PTVF96]; i.e. the data $(v_l, f_l)_{l=low_i,\ldots,high_i-1}$ is fit via the linear function

$frq_i(x) = \alpha_i \cdot X_1(x) + \beta_i \cdot X_0(x)$ with $X_1(x) = x, X_0(x) = 1$ and the optimal $\alpha_i,\ \beta_i$ to be determined. To do this, define the *design matrix* A of $(high_i - low_i) \times 2$ components by $A_{l,h} = X_{l-1}(v_h)$, for $l = 1, 2$ and $h = low_i, \ldots, high_i - 1$. Furthermore, define vector $b = (f_{low_i}, \ldots, f_{high_i-1})^t$. Now the fitting problem can be rephrased to: find $a = \binom{\alpha_i}{\beta_i}$ so as to minimize

$$spline\_err_i = |A \cdot a - b|^2.$$

This problem can now be solved by Singular Value Decomposition (SVD) [PTVF96, BDF+97] in the following way: Let the SVD of A be

$$A = U \times \Lambda \times V^t \quad \text{, with } \Lambda = \begin{bmatrix} s_1 & & \\ & \ddots & \\ & & s_{high_i - low_i} \end{bmatrix}$$

with $s_1, \ldots, s_{high_i - low_i}$ being the singular values of A (i.e., the Eigenvalues of $A \times A^T$). Then a can be expressed as

$$a = \sum_{l=1}^{2} \left(\frac{U_{(i)} \cdot b}{s_i} \right) V_{(i)},$$

with $V_{(i)}$ and $U_{(i)}$ being the i-th column of V and U, respectively. While computing the SVD of the design matrix causes considerable overhead, it is only computed m times, namely, once for each bucket of the final partitioning.

3.1.2 Computation of the Error for a Single Bucket

Because of Formula 3.5, it is possible to compute the error for fitting an interval of frequency values by a linear function without calculating the function frq itself, thereby making the computation of the the optimal partitioning of $\mathcal{V}$ computationally efficient. For this purpose, it is necessary to initially compute and store the following prefix-sums for $i = 1, \ldots, n$, requiring $O(n)$ steps:

$$ff[i] := \sum_{l=0}^{i} (f_l)^2 \qquad vv[i] := \sum_{l=0}^{i} (v_l)^2 \qquad vf[i] := \sum_{l=0}^{i} f_l \cdot v_l$$

$$f[i] := \sum_{l=0}^{i} f_l \qquad v[i] := \sum_{l=0}^{i} v_l$$

These sums stored for the entire duration of the synopsis construction and span all values in $\mathcal{V}$ and their frequencies. Using the prefix sums, it is possible to compute the corresponding sums for intervals; for example the sum of all frequencies in the bucket $b_i = [v_{low_i}, v_{high_i})$ can be computed as $v_{[low_i, high_i)} := v[high_i - 1] - v[low_i] = \sum_{l=low_i}^{high_i - 1} v_l$ in constant time (i.e. a single subtraction). Using this notation for the above sums also when restricted to an interval, it is now possible to

compute the linear correlation for a bucket $b = [v_a, v_c)$ in constant time:

$$
\begin{aligned}
r_{[a,c)} &= \frac{vf_{[a,c)} - f_{[a,c)}\bar{v}_{[a,c)} - v_{[a,c)}\bar{f}_{[a,c)} + (c-a)\bar{v}_{[a,c)}\bar{f}_{[a,c)}}{\sqrt{vv_{[a,c)} - 2\bar{v}_{[a,c)}v_{[a,c)} + (c-a)(\bar{v}_{[a,c)})^2}\sqrt{ff_{[a,c)} - 2\bar{f}_{[a,c)}f_{[a,c)} + (c-a)(\bar{f}_{[a,c)})^2}} \\
&= \frac{\sum_{l=a}^{c-1} v_l f_l - \bar{v}_{[a,c)}\sum_{l=a}^{c-1} f_l - \bar{f}_{[a,c)}\sum_{l=a}^{c-1} v_l + \sum_{l=a}^{c-1}\bar{f}_{[a,c)}\bar{v}_{[a,c)}}{\sqrt{\sum_{l=a}^{c-1} {v_l}^2 - 2\bar{v}_{[a,c)}\sum_{l=a}^{c-1} v_l + \sum_{l=a}^{c-1}(\bar{v}_{[a,c)})^2}\sqrt{\sum_{l=a}^{c-1} {f_l}^2 - 2\bar{f}_{[a,c)}\sum_{l=a}^{c-1} f_l + \sum_{l=a}^{c-1}(\bar{f}_{[a,c)})^2}} \\
&= \frac{\sum_{l=a}^{c-1}(v_l - \bar{v}_{[a,c)})(f_l - \bar{f}_{[a,c)})}{\sqrt{\sum_{l=a}^{c-1}(v_l - \bar{v}_{[a,c)})^2}\sqrt{\sum_{l=a}^{c-1}(f_l - \bar{f}_{[a,c)})^2}}\text{, corresponding to equation } (3.2).
\end{aligned}
$$

Now, the resulting error for a bucket b_i (Equation 3.5) can easily be computed as

$$spline\_err_{[a,c)} = (1 - {r_{[a,c)}}^2) \cdot (ff_{[a,c)} - 2 \cdot \bar{f}_{[a,c)} \cdot f_{[a,c)} + (c-a) \cdot \bar{f}^2_{[a,c)}).$$

As a consequence, after initially using $O(n)$ operations and $O(n)$ space for the prefix-sum computation, the approximation error for value frequencies for a bucket can be computed in constant time.

3.1.3 Partitioning of $\mathcal{V}$

To arrive at the best possible estimation of attribute value frequencies, it is necessary to determine the partitioning (i.e. the boundaries for m buckets) that minimizes the overall error for frequency approximation (Formula 3.6). When arbitrary partitionings and continuous splines of arbitrary degree are considered, this is known as the *optimal knot placement problem* [dB78], which – due to its complexity – can be practically solved only approximatively by heuristic search algorithms [Die93, Jup78].

In our case, however, only linear splines are used and only members of $\mathcal{V}$ are candidates for bucket boundaries. Since the value for each $high_i$ is either low_{i+1} or v_{n+1} (see Definition 3.1), it is only necessary to determine the optimal lower bucket boundaries to compute:

$$f\_opt\_err := \min_{\substack{(low_2,\ldots,low_m)\in\mathcal{V}^{m-1} \\ low_1 \le low2 \le \ldots \le low_m}} \sum_{l=1}^{m}(1 - {r_{[low_l, high_l)}}^2) \cdot histogram\_err_{[low_l, high_l)} \tag{3.7}$$

Because the resulting spline function is allowed to be discontinuous at the boundaries of the chosen intervals $b_1, \ldots, b_m$, fitting the data in a bucket can be addressed separately for each bucket b_i. The main improvement in efficiency does, however, result from the fact that the following *principle of optimality* (also known as the Bellman principle) holds for the partitioning problem:

Theorem 1. *If for $l \ge 2$: $(low_l, low_{l+1}, \ldots, low_m) \in \mathcal{V}^{m-l+1}$ is an optimal partitioning of $[v_{low_{l-1}}, v_{high_m})$ using $m - l + 2$ buckets, then $(low_{l+1}, low_{l+2}, \ldots, low_m) \in \mathcal{V}^{m-l}$ is an optimal partitioning of $[v_{low_l}, v_{high_m})$ using $m - l + 1$ buckets.*

Proof: Because $(low_l, low_{l+1}, \ldots, low_m)$ is optimal, it minimizes

$$E := \sum_{i=l-1}^{m} spline\_err_{[low_i, high_i)} = spline\_err_{[low_{l-1}, high_{l-1})} + \sum_{i=l}^{m} spline\_err_{[low_i, high_i)}.$$

Now assume that $(low_{l+1}, \ldots, low_m)$ is not optimal, i.e. there exists a partitioning $(low'_{l+1}, \ldots, low'_m)$ with $\sum_{i=l}^{m} spline\_err_{[low'_i, high'_i)} < \sum_{i=l}^{m} spline\_err_{[low_i, high_i)}$. But then $(low_l, \ldots, low_m)$ is not optimal either, for the partitioning $(low_l, low'_{l+1}, low'_{l+2}, \ldots, low_m)$ results in the overall error $E' = spline\_err_{[low_{l-1}, high_{l-1})} + \sum_{i=l}^{m} spline\_err_{[low'_i, hig'h_i)} < E$. □

Because of this property, the problem of finding an optimal partitioning can be formulated as a *dynamic programming problem* [Sni92]. To do this, I first formulate a recursive redefinition of Formula 3.7. Define

$$f\_opt\_err_{high,\bar{m}} := \text{the optimal overall error for fitting } [v_1, v_{high}) \text{ by } \bar{m} \text{ buckets.}$$

Trivially, $f\_opt\_err_{high,1} = spline\_err_{[1,high)}$. Because of Theorem 1, the error for the optimal fitting over multiple buckets can be computed as

$$f\_opt\_err_{high,\bar{m}} = \min_{l \in \{1,2,\ldots,high\}} f\_opt\_err_{l,\bar{m}-1} + spline\_err_{[l,high]}. \tag{3.8}$$

Then the overall error produced by the optimal partitioning is

$$f\_opt\_err_{n,m} = \min_{l \in \{1,2,\ldots,n\}} f\_opt\_err_{l,m-1} + spline\_err_{[l,n]}.$$

Computing the optimal partitioning

Using the above definition, the optimal partitioning of $\mathcal{V}$ can be computed in the following way: $f\_opt\_err_{high,\bar{m}}$ is computed for all $\bar{m} = 1, \ldots, m$ (outer loop) and $high = 1, \ldots, n+1$ (see Algorithm 1)[2]; these intermediate results are stored. Note that this means,that by solving the partitioning problem for m buckets, the solution for any number of buckets less than m is also computed at no additional cost. Each computation requires finding the minimum among i values, each of which can be computed in constant time. Because the intermediate results are stored, this requires $O(n)$ operations. Therefore the overall running time is $O(m \cdot n^2)$ and the computation requires $O(m)$ storage (the values for $f\_opt\_err_{1,\dot{m}} \ldots, f\_opt\_err_{n,\dot{m}}$ can be dropped once $\bar{m}$ becomes greater than $\dot{m} + 1$).

The computation of each $f\_opt\_err_{high,\bar{m}}$ can be further sped up by using the idea proposed in [JKM+98] to prune the search space through the use of upper bounds for the values of f_opt_err. To do this, an upper bound S_0 for the size of $f\_opt\_err_{high,\bar{m}}$ is used. S_0 can be obtained by computing the error corresponding to an *equi-width* partitioning, where all buckets are chosen so that they cover intervals of roughly the same size. This error can be computed in $O(j)$ steps. When now computing $f\_opt\_err_{high,\bar{m}}$ as defined in Equation 3.8, by iterating over all possible values of l, all values of l for which $spline\_err_{[l,high)} > S_0$ can be ignored, as they cannot result in an optimal partitioning. To this end the largest value for l is initially located (using a binary search), such that $spline\_err_{[l,high)} > S_0$, which is referred to as l_0. When computing $f\_opt\_err_{high,\bar{m}}$, it is now sufficient to consider only values above l_0 for l, thereby improving performance.

[2] In all algorithms I have ignored the special case occurring if $n < 2m$. In this case there is sufficient storage space to store $\mathcal{T}$ directly and thus no approximation is necessary.

1: **for** $l = 2$ to $n + 1$ **do**
2: $f\_opt\_err_{l,1} := spline\_err_{[1,l)}$
3: **end for**
4: **for** $\bar{m} = 2$ to m **do**
5: **for** $high = 2m + 1$ to $n + 1$ **do**
6: $minimum =$ `MAXFLOAT`; // The following loop solves equation 3.8
7: **for** $l = 2(m - 1)$ to $high - 1$ **do**
8: **if** $minimum > f\_opt\_err_{l,\bar{m}-1} + spline\_err_{[l,high)}$ **then**
9: $minimum := f\_opt\_err_{l,\bar{m}-1} + spline\_err_{[l,high)}$ // New minimum found? Yes!
10: $next\_split_{high,\bar{m}} := l$
11: **end if**
12: **end for** $f\_opt\_err_{high,\bar{m}} = minimum$
13: **end for**
14: **end for**
15: // Compute the final partitioning
16: $low_1 := 1$
17: $current\_split := next\_split_{n+1,m}$
18: **for** $\bar{m} = m - 1$ downto 2 **do**
19: $low_{\bar{m}} := next\_split_{current\_split,m}$
20: $current\_split := low_{\bar{m}}$
21: **end for**

Algorithm 1: Computing the optimal partitioning

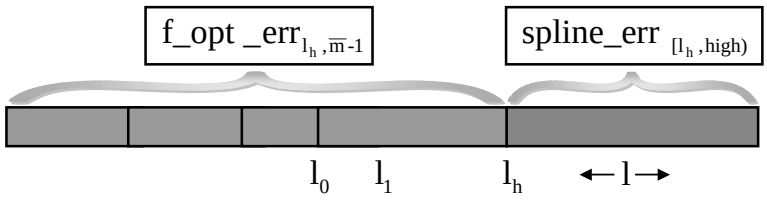

Figure 3.2: Using upper bounds

In addition, because all possible values for l must be larger than l_0, it holds that the value of $f\_opt\_err_{l_0,\bar{m}-1}$ is a lower bound for $f\_opt\_err_{l,\bar{m}-1}$ in the optimal solution. Consequently, $S_1 = S_0 - f\_opt\_err_{l_0,\bar{m}-1}$ is an upper bound for $spline\_err_{[l,high)}$ in the optimal solution. Therefore, the algorithm can now search for the largest value of l, such that $spline\_err_{[l,high)} > S_1$, which is referred to as l_1. Again, it holds that all values for l must be larger than l_1.

Repeating this construction for each iteration an upper bound

$$S_h := S_0 - f\_opt\_err_{l_{h-1},\bar{m}-1}$$

is computed and using binary search the corresponding l_h is detected until $l_h = l_{h-1}$. Then this l_h is used as a limit on l when computing the minimization for $f\_opt\_err_{high,\bar{m}}$. The relationship between $f\_opt\_err_{l_{h-1},\bar{m}-1}$, $spline\_err_{[l_h,high)}$ and the different values for l is depicted in Figure 3.2

While the worst-case complexity for the algorithm remains the same, the actual run-time improves dramatically in most cases, as far less *spline_err* values need to be computed. I refer to this partitioning algorithm as *OPTIMAL*. This algorithm is identical to Algorithm 1 except for the initial value chosen for l in line 7. The computation of the initial value of l is described in Algorithm 2. To obtain the complete *OPTIMAL* algorithm, Algorithm 2 needs to be inserted

between lines 6 and 7 in Algorithm 1 and the final value for l_h used as the lower bound for the loop in line 7.

```
1: // Compute S_0
2: S_0 := 0
3: width = (high − 1)/m̄
4: for i = 0 to m̄ − 1 do
5:    S_0 := S_0 + spline_err_[1+i·width,1+(i+1)·width)
6: end for
7: h := 0
8: repeat
9:    // Now compute the l_h corresponding to S_h
10:   left := 1
11:   right := high − 2
12:   while right ≥ left do
13:      mid := (left + right)/2
14:      if spline_err_mid,high > S_h then
15:         left := mid + 1
16:      else
17:         right := mid − 1
18:      end if
19:   end while
20:   l_h := right · [spline_err_left,high ≤ S_h] + left · [spline_err_left,high > S_h]
21:   h := h + 1
22:   S_h := S_0 − f_opt_err_l_{h−1},m̄−1
23: until l_h = l_h − 1
```

Algorithm 2: Computing the initial value for l

Greedy Partitioning

Despite the above improvements, the cost for computing an optimal partitioning could still be unacceptable when n is large. In these cases, a greedy method for the partitioning of $\mathcal{V}$ can be used, which results in a partitioning that is close to optimal while being much more efficient. This algorithm first partitions $\mathcal{V}$ into $n/2$ trivial bucket and then merges the ones that lead to the smallest increase in the overall error, until only m buckets are left (see Algorithm 3). Here, the increase in the error by merging two buckets $b_1 = [v_a, v_b), b_2 = [v_b, v_c)$ is denoted as

$$error_{[a,b),[b,c)} := spline\_err_{[a,c)} - (spline\_err_{[a,b)} + spline\_err_{[b,c)}).$$

The algorithm consists of 2 loops; the **for** loop has $n/2$ iterations in which the error of merging the trivial buckets is computed, which can be done in constant time. The **repeat** loop is executed $(n/2) - m$ times (each repetition reduces the number of buckets by one, there are $n/2$ buckets initially, and m upon termination), and executes 4 different types of operations:

1. Removing an item from the *priority queue* Q. An implementation of priority queues based on *Fibonacci heaps* [MNU97] is used in the algorithm, allowing the removal of an item in a queue of size n in $O(\log_2 n)$ time.

2. Merging two buckets, requiring constant time.

1: Partition $\mathcal{V}$ into $\frac{n}{2}$ buckets $b_i = [v_{2i-1}, v_{2i+1})$.
2: **for** $l = 0$ to $\frac{n}{2}$ **do**
3: Compute the $error_{[2l-1,2l+1),[2l+1,2l+3)}$ resulting from merging the buckets b_l and $b_l + 1$ and insert the error value into *priority queue* Q.
4: **end for**
5: **repeat**
6: Remove the minimal $error_{[a,b),[b,c)}$ from Q.
7: Merge the buckets corresponding to $[a, b)$ and $[b, c)$.
8: Remove $error_{[a',a),[a,b)}$, $error_{[b,c),[c,c')}$ from Q.
9: Calculate the error of joining the new bucket with its left and right neighbor (if these exist); insert the resulting $error_{[a',a),[a,c)}$, $error_{[a,c),[c,c')}$ into Q.
10: **until** there are only m buckets left.

Algorithm 3: *GREEDY-MERGE*

3. Calculating the error resulting from a merge, requiring constant time.

4. Inserting an item into Q, again requiring $O(\log_2 n)$ time.

Since each operation is carried out no more than three times, computing a greedy partitioning is of complexity $O(n \log_2 n)$. The algorithm requires $O(n)$ storage. By performing additional $m-1$ merge operations (i.e., until only one bucket remains), the greedy partitionings for all numbers of buckets less than m is obtained at very low additional cost. I will refer to this algorithm as *GREEDY-MERGE*.

In addition, I consider a second greedy partitioning-algorithm (see Algorithm 4, which takes the opposite approach: Initially, all tuples are grouped in one bucket. Then the split that leads to the greatest reduction in the overall error (formula 3.6) is computed and executed, resulting in an additional bucket. This is repeated, until (after $m-1$ splits) m buckets remain. Each time the location of a split is computed, at most $O(n)$ possible split-locations need to be considered. These splits are then stored in a priority queue holding at most $m-1$ values (as only one split-location is stored for each bucket). Thus the inner loop (lines 12-35) of the algorithm require $O(n \cdot \log_2 m)$ overhead. The loop itself is repeated $O(m)$ times, and lines 1-11 compute the optimal split location in the initial bucket ($O(n)$ operations). Consequently, the overall algorithm requires $O(m \cdot n \log_2 m)$ time and $O(n)$ space. Like the optimal partitioning, the greedy partitionings of $\mathcal{V}$ for less that m buckets are computed at no additional cost. I will refer to this algorithm as *GREEDY-SPLIT*.

3.2 Approximation of the Value Density

Accurate approximation of the density distribution is of critical importance for the estimation of all queries involving grouping and aggregation. Therefore, the value densities in $\mathcal{V}$ are approximated in addition to and independently of the value frequencies, such that:

1. the approximation captures the same number of values as $\mathcal{V}$ (although it should require less values to represent $\mathcal{V}$) and

2. the squared deviation between the actual attribute values $v_l \in \mathcal{V}$ and their approximated counterparts $\hat{v}_l$ is minimized.

```
1: all_buckets := ∅
2: Insert b = [v_1, v_{n+1}) into all_buckets
3: minimum :=MAXFLOAT
4: // Find the optimal split for the initial bucket
5: for i = 3 to n − 1 do
6:    if minimum > error_{[1,i)[i,n+1)} then
7:       minimum := error_{[1,i)[i,n+1)}
8:       min_index := i
9:    end if
10: end for
11: Insert the tuple (minimum, min_index) into the priority queue Q
12: for i = 3 to m do
13:    Remove the tuple (min, ind) from Q with the smallest min
14:    Remove the corresponding bucket b' = [v_low, v_high) with low < min < high from
       all_buckets
15:    Insert b_left = [v_low, v_ind) and b_right = [v_ind, v_high) into all_buckets
16:    if i ≠ m then
17:       // Find the optimal split for the new buckets
18:       minimum :=MAXFLOAT
19:       for i = low + 1 to ind − 1 do
20:          if minimum > error_{[low,i)[i,ind)} then
21:             minimum := error_{[low,i)[i,ind)}
22:             min_index := i
23:          end if
24:       end for
25:       Insert the tuple (minimum, min_index) into the priority queue Q
26:       minimum :=MAXFLOAT
27:       for i = ind + 1 to high − 1 do
28:          if minimum > error_{[ind,i)[i,high)} then
29:             minimum := error_{[ind,i)[i,high)}
30:             min_index := i
31:          end if
32:       end for
33:       Insert the tuple (minimum, min_index) into the priority queue Q
34:    end if
35: end for
36: // Now all_buckets contains the final buckets
```

Algorithm 4: *GREEDY-SPLIT*

Analogously to the approach for frequency distributions, $\mathcal{V}$ is partitioned into m' disjoint intervals which are referred to as *density buckets* $db_i = [v_{dlow_i}, v_{dhigh_i})$, with $dlow_i$ and $dhigh_i$ denoting the subscripts of the values from $\mathcal{V}$ that constitute the bucket boundaries. Analogously to Formula 3.1, for all $i \in \{1, 2, \ldots, m' - 1\}$ the partitioning requires:

$$dhigh_i = dlow_{i+1} \text{ and } dlow_1 = 1,\ dhigh_{m'} = n + 1.$$

Note that the number of buckets, m', can be chosen independently of the number of buckets, m, for the frequency distribution of the same attribute. Using the squared deviation of attribute

values, the error within bucket db_i is

$$density\_err_{[dlow_i,dhigh_i)} = \sum_{l=dlow_i}^{dhigh_i} \big(v_l - \hat{v}_l\big)^2. \tag{3.9}$$

In order to estimate the value distribution within a bucket, histograms use the *equi-spread assumption* [Poo97], meaning that they assume that the values are spread evenly over the bucket's width. This means that the j-th value in a density bucket $db = [dlow, dhigh)$ containing D values is approximated as $\hat{v} = v_{dlow} + (j-1) \cdot \frac{v_{dhigh} - v_{dlow}}{D}$. Thus, denoting the number of values in bucket db_i by D_i, the error of the bucket becomes:

$$histogram\_density\_err_{[dlow_i,dhigh_i)} = \sum_{l=0}^{D_i-1} \left(v_{dlow_i+l} - \left(v_{dlow_i} + \big(l \cdot \frac{v_{dhigh_i} - v_{dlow_i}}{D_i} \big) \right) \right)^2. \tag{3.10}$$

Using the upper and lower values to estimate the approximate spread in buckets has the disadvantage that even when the estimation of v_{dhigh} and v_{dlow} is correct[3], all possible partition constraints do not consider the width of a bucket $v_{dhigh} - v_{dlow}$ as a *source parameter* and thus are not capable of identifying the best partitioning with regards to attribute density estimation. To illustrate this, consider the following example:

Example 3.1: Consider the set of attribute values $\mathcal{V} = \{0, 1, 7, 9, 10, 13, 15\}$, which is approximated using a 2-bucket *V-optimal(V,S)* histogram. This means that the final partitioning of the value domain is based on the variance of the value spreads in the resulting buckets:

$$o\_variance := \sum_{i=1}^{2} \left(\sum_{j=dlow_i}^{dhigh_i-1} s_i - \Big(\frac{1}{dhigh_i - dlow_i} \sum_{j=dlow_i}^{dhigh_i-1} s_i \Big) \right)^2.$$

However, minimizing the overall variance does not necessarily result in the optimal estimation of the value domain. If $\mathcal{V}$ is partitioned into the 2 buckets $b_1 = [0, 9)$, $b_2 = [9, 15)$ the overall variance becomes $o\_variance_1 = 33/2$ and the overall estimation error $error_1 := histogram\_density\_err_{[0,9)} + histogram\_density\_err_{[9,15)} = 145/4$. On the other hand, the partitioning $b_1 = [0, 10)$, $b_2 = [10, 15)$ results in the overall variance $o\_variance_2 = 18$ and the overall estimation error $error_2 := histogram\_density\_err_{[0,10)} + histogram\_density\_err_{[10,15)} = 47/2$. Thus $o\_variance_1 < o\_variance_2$, but $error_1 > error_2$. Similar examples can be constructed for all other combinations of *partitioning constraint*, *source parameter* and error-function (e.g., if the linear deviation of the attribute values is used as a measure of the density error (Equation 3.9)).

The above problem has the additional unwanted side-effect, that a density-estimation may become worse if it receives additional memory.

In order to minimize the error defined in Equation 3.10, the term $\frac{v_{dhigh_i} - v_{dlow_i}}{D_i}$, which denotes the "gap" between adjacent values in the approximation, is treated as a control variable, denoted Δ_i, whose value should be chosen optimally. Note that the best value of Δ_i may be larger than both $\frac{v_{dhigh_i} - v_{dlow_i}}{D_i}$ and $\frac{v_{dhigh_i+1} - v_{dlow_i}}{D_i}$ if the density distribution is skewed towards values closer to the interval's upper boundary.

[3] This is not necessarily the case, as histograms with the sort parameter *values* typically only store the highest value in a bucket and estimate the lowest value as the successor of the highest value contained in the previous bucket [Poo97].

The formula for the resulting bucket error then is obtained from equation 3.10 by replacing $\frac{v_{dhigh_i} - v_{dlow_i}}{D_i}$ with Δ_i:

$$density\_err_{[dlow_i, dhigh_i)} = \sum_{l=0}^{D_i-1} \left(v_{dlow_i+l} - \left(v_{dlow_i} + l \cdot \Delta_i \right) \right)^2. \tag{3.11}$$

The parameter Δ_i should be chosen such that the bucket's error is minimized, i.e.

$$\frac{\partial density\_err_{[dlow_i, dhigh_i)}}{\partial \Delta_i} = 0.$$

Computing the derivative and solving this equation for Δ_i yields

$$\Delta_i = -3 \frac{-v_{low_i} D_i + v_{low_i} (D_i)^2 + \sum_{l=1}^{D_i-1} (-2l \cdot v_{low_i+l})}{D_i \left(2(D_i)^2 - 3D_i + 1\right)}, \tag{3.12}$$

and this indeed results in the minimum value for Equation 3.11.

The optimal density error for a bucket can now be computed by substituting v_{opt_i} into equation 3.11. The overall optimal error for the entire attribute-value density synopsis then is the sum of the errors over all buckets 1,..., m':

$$d\_opt\_err_{n,m'} = \sum_{i=1}^{m'} density\_err_{[dlow_i, dhigh_i)}. \tag{3.13}$$

Using the prefix-sums defined in Section 3.1.2 it is possible to evaluate Equation 3.12 and thus $density\_err_{[dlow_i, dhigh_i)}$ in constant time. Therefore, finding an optimal partitioning of $\mathcal{V}$ and computing the optimal values for the control parameters Δ_i is mathematically equivalent to the partitioning and per-bucket parameter fitting problem for frequency distributions that were solved in the previous section. Thus the same dynamic programming algorithm or, alternatively, one of the greedy heuristics can be applied to compute an optimal density synopsis with a given number m' of buckets. I refer to the resulting error as $d\_opt\_err_{1,m'}$.

3.3 Synopsis Storage

In the following I will describe what information is stored in the buckets used for attribute-value frequency and attribute-value density approximation and how the memory M available to the entire synopses is divided between the two approximation types (Section 3.3.2). A similar approach is used to divide the available memory between multiple spline synopses (Section 3.3.3). To ensure accurate estimation of value frequencies, it is also important to match up the different approximations to some degree, which is being discussed in Section 3.3.4.

3.3.1 Storage Requirements

The buckets for both frequency and density approximation store 3 values each. Frequency-approximation buckets store:

(a) the leftmost attribute value in the bucket, and
(b) the two coefficients α_i and β_i.

Density-approximation buckets store:

(a) the leftmost attribute value in the bucket,
(b) the fitting-parameter Δ_i, and
(c) the number D_i of unique attribute values contained in the bucket.

Assuming each value is stored using a constant length number, the storage requirements for both bucket types are identical.

3.3.2 Reconciling Frequency and Density Synopses for One Attribute

So far I have shown how to minimize the error for each aspect separately, but assumed that the number of buckets, m for frequency and m' for density synopses, is fixed and a priori given for each synopsis. To reconcile both synopses for an attribute, the problem is how to divide the available memory space between them, i.e., how to choose the values of m and m' under the constraint that their sum should not exceed a given constant M. Intuitively, we are interested in allocating the memory in such a way that the aspect that is more difficult to approximate is given a higher number of buckets. This amounts to minimizing the combined error of both approximations. However, to make sure that we do not to compare "apples with oranges" (i.e., value frequencies versus actual data values), it has to be taken into account that the two approximations refer to domains with possibly radically different scales. Therefore, the error metrics are first normalized for the two classes of synopses thereby defining a relative error for each aspect. As the normalizing factor, I use the highest possible approximation error for each domain, i.e. the maximal difference between the actual value f_i (or v_i) and it's approximation $\hat{f}_i$ (or $\hat{v}_i$, respectively), when the respective domain is approximated using one bucket only. Intuitively, these factors represent the "difficulty" of approximating each domain. I refer to these factors as f_domain and v_domain; the relative error for a single value v_i then becomes $\left(\frac{v_l - \hat{v}_l}{v\_domain}\right)^2 = \frac{(v_l - \hat{v}_l)^2}{(v\_domain)^2}$. The relative error for a single frequency is $\left(\frac{f_j - frq(v_j)}{f\_domain}\right)^2 = \frac{(f_j - frq(v_j))^2}{(f\_domain)^2}$.

In addition to this normalization it is desirable to consider the relative importance of the two synopses. With a query workload that is dominated by range or exact-match queries, frequency estimations are obviously the key point. On the other hand, with a workload that requires many estimations of the result size of projection and grouping queries, the density synopses become more important. This consideration will be discussed in more detail in Chapter 5 in the context of the physical design problem for data synopses.

To divide the available memory, I exploit the fact that computing the optimal partitioning of either approximation type for j buckets, using the algorithms of Section 3.1.3, also generates the optimal partitionings for $1, \ldots, j-1$ buckets at no additional computational cost. The same holds for the GREEDY-SPLIT heuristics; and for GREEDY-MERGE the additional output can be generated at very small extra cost (with j additional merge operations). Thus, with a given amount of memory for a total of M buckets, I compute the optimal partitionings and resulting relative errors for each approximation with up to $M-1$ buckets, and then divide the available memory in such a way that the sum of the combined relative error terms

$$f\_d\_err = \frac{f\_opt\_err_{1,m}}{(f\_domain)^2} + \frac{d\_opt\_err_{1,m'}}{(v\_domain)^2} \tag{3.14}$$

is minimized under the constraint that $m \geq 1$, $m' \geq 1$ and $m + m' = M$. This is a trivial combinatorial problem, solvable in $O(M)$ steps.

3.3.3 Reconciling Multiple Synopses

When the overall available memory M needs to be divided between h different synopses over the attributes $R_{i_1}.A_{j_1},\ldots,R_{i_h}.A_{j_h}$, the sizes of the single frequency and density approximations are again chosen in such a way that the combined relative error over all synopses is minimized. Thus, if $f\_d\_err_{R_{i_l}.A_{j_l}}$ denotes the combined relative error for the approximation of attribute $R_{i_l}.A_{j_l}$, the sizes of each separate approximation is choose so that the term

$$overall\_f\_d\_err_M := \sum_{l=1}^{h} f\_d\_err_{R_{i_l}.A_{j_l}}$$

is minimized. Similar to the partitioning problem for $\mathcal{V}$, the bellman principle holds, thereby allowing to compute the optimal memory allocation using $O(h \cdot M^2)$ operations. While a good approximative solution can again be computed using greedy heuristics, it is generally the case that $M \ll n$, and therefore the time for computing the optimal memory partitioning insignificant (see Section 3.5.3). Note that solving the minimization problem also gives the values for $overall\_f\_d\_err_{\tilde{M}}$ for all values $\tilde{M} \leq M$.

3.3.4 Matching Frequency and Density Synopses

While the above techniques allow us to minimize the weighted sum of the approximation error for attribute value density and frequency, this is not always sufficient to guarantee a low estimation error for the queries described in the previous subsection.

In order to illustrate the problem, I examine a value-frequency pair (v_l, f_l) from a data distribution $\mathcal{T}$ and its approximation $\left(\hat{v}_l, \hat{f}_l\right)$. Since frequencies are approximated as linear spline functions $frq_i, i = 1,\ldots,m$ over $\mathcal{V}$, $\left(\hat{v}_l, \hat{f}_l\right)$ can be written as $\left(\hat{v}_l, frq_k(\hat{v}_l)\right)$ for some $k \in \{1,\ldots,m\}$. When approximating attribute-value frequencies, the error of Formula 3.3 is minimized and the frequencies at the original attribute values $v_1,\ldots,v_n$ are fit, rather than their approximations $\hat{v}_1,\ldots,\hat{v}_n$. In most cases (when v_l and $\hat{v}_l$ belong to the same frequency bucket k, with $frq_k(x) = \alpha_k \cdot x + \beta_k$) this still results in accurate approximation, since the approximation error

$$\begin{aligned} \text{for } f_l &:= |f_l - frq_k(\hat{v}_l)| \\ &= |f_l - frq_k(v_l + (\hat{v}_l - v_l))| \\ &= |\underbrace{f_l - frq_k(v_l)}_{\text{minimized by frequency approx}} + \alpha_k \cdot \underbrace{(\hat{v}_l - v_l)}_{\text{minimized by density approx}}|. \end{aligned}$$

The problem becomes apparent either for large values of α_k (resulting in a large second term in the above formula) or when v_l and $\hat{v}_l$ belong to different frequency buckets. In the latter case (demonstrated in Figure 3.3), the frequency of v_l is estimated as $frq_k(v_l)$, whereas the frequency of $\hat{v}_l$ by $frq_t(\hat{v}_l)$, $t \neq k$. Now, frq_k is chosen to fit all points in $[low_k, high_k)$, among them (v_l, f_l), optimally, thereby reducing the error $|f_l - frq_k(v_l)|$. The function frq_t is fitted to a different set of points, not including (v_l, f_l), and therefore a possibly poor estimator for f_l.

To avoid this problem, I use $\hat{\mathcal{T}} = \{(\hat{v}_1, f_1),\ldots,(\hat{v}_n, f_n)\}$ for the approximation of the frequency domain, thereby minimizing the approximation error for the attribute values which will later be used in estimation. So density is approximated first, and the frequency approximation then uses the approximate density rather than the original distribution. However, $\hat{\mathcal{T}}$ depends on the number of buckets m' used for the approximation of the attribute value density, which in turn depends on the error $f\_opt\_err_{1,m}$ (see Formula 3.8) made when fitting $\hat{\mathcal{T}}$. A straightforward solution would be to compute the resulting $\hat{\mathcal{T}}$ for all possible $M-1$ values of m', fit them, and

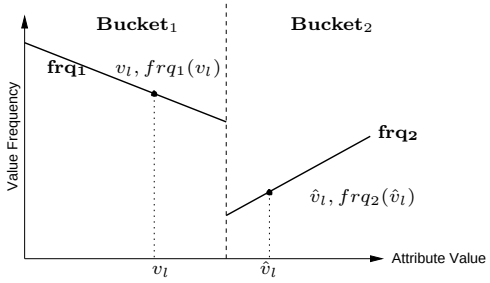

Figure 3.3: The need for matching approximations

then determine the optimal combination of density and frequency error. This, however, would increase the computational complexity of our approach by the factor of M. Therefore, I use a simpler approach. First, the error $f\_opt\_err_{1,m}$ for fitting $\mathcal{T}$ is computed and used to determine the corresponding numbers of buckets, m, m', for frequency and density approximation. Then $\hat{\mathcal{T}}$ is re-computed for this value for m', which is then used to compute the optimal approximation of the value frequency, with the resulting spline being the final synopsis. Thus the optimal partitioning of $\mathcal{V}$ for value frequency approximation is only computed twice, instead of M times. Furthermore, since the $f\_opt\_err_{1,m}$ values for fitting $\mathcal{T}$ and for fitting $\hat{\mathcal{T}}$ typically close to each other, both approaches tend to result in the same values for m and m' and therefore in identical approximations.

3.3.5 Weighted Approximation

In Chapter 5 it will become necessary to solve a modified version of the approximations described, for the estimation for each value v_i and frequency f_i will be assigned a relative weight w_i. This means that the error-functions to be minimized when approximating a bucket $b_i = [low_i, high_i)$ are

$$\sum_{l=low_i}^{high_i} w_l(frq_i(v_l) - f_l)^2 \text{ and } \sum_{l=low_i}^{high_i} w_l(\hat{v}_l - v_l)^2.$$

While this results in changes to some details for the approximation (the prefix-sums used need to be weighted as well, and the formulas for evaluating the error for a bucket in constant time change), the computational properties and the overall approach remain the same.

3.4 Using Spline Synopses for Query Estimation

In the following I describe how to use the resulting synopses for estimating the result sizes of various types of basic queries and for approximative query answering. The focus is on the following query types: projections or, equivalently as far as result-size estimation is concerned, grouping with aggregation, range queries on a single attribute, and joins, with projections and joins possibly restricted to a specified value range (i.e., in conjunction with a range filter). The extension to complex operator-trees is then straightforward: the initial approximate query-results at every branch of the operator tree are then subjected to the further operators using the operator semantics described in Section 2.2.

Because the estimation of join queries is inherently difficult when using synopses for the two joining relations only (independent of which approximation technique is used), special join

synopses are introduced, which provide additional information on the number and value of the attribute values that find join-partners in the two relations.

3.4.1 Projections (and Grouping Queries)

When estimating the (number of) unique values resulting from a projection over the interval $[a, b)$, there are three cases to differentiate:

- If $[a, b)$ corresponds to the boundaries of a density bucket db_i (i.e., $a = v_{dlow_i}$ and $b = v_{dlow_{i+1}}$), the resulting value distribution is

  $$\hat{\mathcal{V}} := \left\{ v_{dlow_i} + l \cdot \Delta_i \middle| l = 0, \ldots, D_i - 1 \right\}.$$

  If only the size of the query result is of interest, the estimation corresponds to the number of values in bucket i:

  $$P^i_{[a,b)} = D_i.$$

  The notation $P^i_{[a,b)}$ is used to indicate that the query over $[a, b)$ is answered using only bucket i.

- If $[a, b)$ is completely contained in density bucket db_i (i.e., $a \geq v_{dlow_i}$ and $b < v_{dlow_{i+1}}$), the resulting value distribution is computed analogously, the only difference being that only values between a and b are considered:

  $$\hat{\mathcal{V}} := \left\{ v_{dlow_i} + l \cdot \Delta_i \middle| l = \left\lceil \tfrac{a - v_{dlow_i}}{\Delta_i} \right\rceil, \ldots, \left\lfloor \tfrac{b - v_{dlow_i}}{\Delta_i} \right\rfloor \right\}.$$

  Again, when only the result size is of interest, the estimation returns only the number of unique values occurring in $\hat{\mathcal{V}}$:

  $$P^i_{[a,b)} = \left\lceil \tfrac{b - v_{dlow_i}}{\Delta_i} + 1 \right\rceil - \left\lfloor \tfrac{a - v_{dlow_i}}{\Delta_i} \right\rfloor.$$

- Finally, an approximate projection query for $[a, b)$ that spans more than one density bucket can easily be estimated by clipping $[a, b)$ against the buckets and building the union of the estimations for the intervals $[a, v_{dlow_j}), [v_{dlow_j}, v_{dlow_{j+1}}), \ldots [v_{dlow_{j+l}}, b)$. Thus, the estimation problem is reduced to the previous formulas.

Consequently, any projection query requiring the approximation of an approximate value distribution $\hat{\mathcal{V}}$ can be estimated using $O(|\hat{\mathcal{V}}|)$ operations. If only the result size is of interest, it can be estimated using $O(m')$ operations. Whenever approximating values over discrete value domains (e.g., integer domains), one can improve the estimation quality by rounding to the next discrete value.

3.4.2 Range Selections

For purpose of range selections, it is first necessary to introduce a unified notation for the spline functions $frq_1, \ldots, frq_m$ that approximate the frequency distribution.

Definition 3.1 (Frequency Approximation Function) :
The frequency approximation function described through the attribute frequency approximation of an attribute $R_i.A_j$ is a mapping $frequency_{R_i.A_j} : \mathcal{D}_{R_i.A_j} \mapsto \mathbb{N}$ defined by

$$frequency_{R_i.A_j}(x) := \sum_{l=1}^{m} frq_l(x) \cdot [low_l \leq x < high_l].$$

If the result it not in $\mathbb{N}$, it is rounded to the next integer. When it is clear, which relation and attribute are referred to, the subscript $R_i.A_j$ is dropped.

To estimate the query result for a range query with range $[a, b)$, first the approximative density distribution $\hat{\mathcal{V}}$ is computed for $[a, b)$ using the above technique for projection queries (see Section 3.4.1), and then the frequency distribution of the range query result is computed as $\hat{\mathcal{F}} = \{frequency(\hat{v}) \mid \hat{v} \in \hat{\mathcal{V}}\}$.

When only the number of tuples in $[a, b)$, i.e., the query result size, is to be estimated, the computation can be sped up in the following way. Here, it is necessary to differentiate two cases:

- The interval $[a, b)$ is completely contained in a density bucket db_i (i.e., $a \geq v_{dlow_i}$ and $b \leq v_{dlow_{i+1}}$).In this case, the estimation of the result size requires the evaluation of $\tilde{m}$ terms, with $\tilde{m}$ corresponding to the number of buckets $b_l, \ldots, b_{l+\tilde{m}-1}$ (used for frequency estimation) that intersect the interval $[a, b)$. For these buckets I now estimate the number of tuples in the corresponding intervals $[a, v_{low_{l+1}}), [v_{low_{l+1}}, v_{low_{l+2}}), \ldots, [v_{low_{l+\tilde{m}-1}}, b)$. For each interval $I = [start, end)$ that corresponds to the intersection of density-bucket db_i and frequency-bucket b_j, the number of tuples is estimated as

$$\begin{aligned} S^{i,j}_{[start,end)} : &= \sum_{l=\lceil start - v_{dlow_i}/\Delta_i \rceil}^{\lfloor end - v_{dlow_i}/\Delta_i \rfloor} frq_j\big(v_{d_{low_i}} + l \cdot \Delta_i\big) \qquad (3.15) \\ &= \Big(\Big\lceil \tfrac{end - v_{dlow_i}}{\Delta_i} \Big\rceil - \Big\lfloor \tfrac{start - v_{dlow_i}}{\Delta_i} \Big\rfloor + 1\Big)(\beta_j + \alpha_j \cdot v_{dlow_i}) \\ &\quad + \tfrac{\Delta_i \alpha_j}{2}\Big(\Big\lceil \tfrac{end - v_{dlow_i}}{\Delta_i} \Big\rceil - \Big\lfloor \tfrac{start - v_{dlow_i}}{\Delta_i} \Big\rfloor + 1\Big)\Big(\Big\lceil \tfrac{end - v_{dlow_i}}{\Delta_i} \Big\rceil - \Big\lfloor \tfrac{start - v_{dlow_i}}{\Delta_i} \Big\rfloor\Big). \end{aligned}$$

  The importance of the second equality is that it means $S^{i,j}_{[start,end)}$ can be evaluated in constant time. The estimation over $[a, b)$ can now be computed as

$$S^{i}_{[a,b)} := S^{i,j}_{[a,v_{low_{l+1}})} + \Big(\sum_{h=l+1}^{l+\tilde{m}-2} S^{i,h}_{[v_{low_h}, v_{low_{h+1}})}\Big) + S^{i,l+\tilde{m}-1}_{[v_{low_{l+\tilde{m}-1}}, b)},$$

  requiring $O(\tilde{m})$ operations.

- When interval $[a, b)$ spans multiple density-buckets the estimation problem can be reduced to the previous formulas by clipping $[a, b)$ against the buckets and building the union of the estimations for the intervals $[a, v_{dlow_j}), [v_{dlow_j}, v_{dlow_{j+1}}), \ldots [v_{dlow_{j+l}}, b)$. Then the selectivity can be computed as

$$S_{[a,b)} := S^{j}_{[a,v_{dlow_j}} + \Big(\sum_{h=1}^{l-1} S^{j+h}_{[dlow_{j+h}, dlow_{j+h+1})}\Big) + S^{j+l}_{[dlow_{j+l}, b)}. \qquad (3.16)$$

  For all intervals $[start, end)$ for which the term 3.15 is evaluated the values for $start, end$ come from the set

$$Bounds := \{\max\{start, v_{low_1}\}, v_{low_2}, \ldots, v_{low_m}, v_{dlow_2}, \ldots, v_{dlow_{m'}}, \min\{end, v_{n+1}\}\}$$

  with $\nexists middle \in Bounds \therefore start < middle < end$ (otherwise the intervals $[start, middle)$ and $[middle, end)$ would be evaluated instead). Because there are only $m + m'$ such combinations possible, expression 3.16 can be evaluated using at most $O(m + m')$ evaluations of term 3.15.

Consequently, any range-selection query requiring the approximation of an approximate data distribution $\hat{\mathcal{V}}$ can be estimated using $O(|\hat{\mathcal{V}}|)$ operations. If only the result size is of interest, the estimation requires $O(m + m')$ operations.

Because spline synopses provide query result estimation, they can be used to estimate any filter condition other than range-specifications as well.

3.4.3 The Inherent Difficulty of Join Estimation

In principle, the data synopses constructed so far can also be used when estimating a join query between two relation R_1 and R_2: first the approximative density distributions $\hat{\mathcal{V}}_1$ and $\hat{\mathcal{V}}_2$ would be computed and the join between them would be performed, resulting in the approximate density distribution $\hat{\mathcal{V}}_{join}$. The join result's frequency distribution would be computed as $\hat{\mathcal{F}}_{join} = \{frequency_1(\hat{v}) \cdot frequency_2(\hat{v}) \mid \hat{v} \in \hat{\mathcal{V}}_{join}\}$, with $frequency_1$ and $frequency_2$ being the frequency functions of the two join attributes.

This approach is similar to how joins are estimated by histograms and in [CGRS00] (the only techniques besides spline synopses capable of both query result and join estimation) in that first a join is performed on the approximate attribute value distribution and the frequencies of the values with join partners are then computed as the product of the corresponding frequency approximations. Unfortunately, this approach does not lead to accurate join estimation. The reason for this is that even small errors incurred when approximating the attribute value domain can lead to drastic changes in the number and value of attribute values that find a join partner[4]. Therefore, minimizing the average error in the approximation is not sufficient to ensure accurate join estimation.

Example 3.2: To illustrate this point, consider the approximation of the equi-join between two data distributions $\mathcal{T}_1 = \{(v_1, f_1), \ldots, (v_n, f_n)\}$ and $\mathcal{T}_2 = \{(v_1, f'_1), \ldots, (v_k, f'_k)\}$ via spline synopses. Assume that $\mathcal{T}_1$ is approximated exactly. If the approximation of the attribute density of $\mathcal{T}_2$ has very low error, but is not exact, i.e., at least a single value v_i and its approximation $\hat{v}_i$ differ by more than 1, then it will not find its correct join-partner. In case it does not find a join-partner at all (i.e. $\nexists j \in \{1, \ldots, n\} : \hat{v}_i = v_j$, the join is then underestimated by at least $f_i \cdot f'_i$.

The same problem exists for histograms, especially since most histogram types are geared towards minimizing the estimation error for attribute value frequency only. To limit the estimation error when either f_i or f'_i are very large, *high-biased* [IP95] and various kinds of *end-biased* [Poo97] histograms have been proposed, which keep the attribute values with the highest (and in the case of *end-biased* histograms also the lowest) frequencies in singleton buckets storing only a single attribute value. Thereby, these attribute values and frequencies are estimated correctly. Still, this approach fails whenever an attribute value involved in a join is not in a singleton bucket for both histograms. Especially in foreign-key joins, where all values are keys in one of the joining relations and thus only have frequency 1, most high-frequency tuples in the joining relation tend to be matched up with tuples not in singleton buckets.

Example 3.3: To illustrate that the same problem arises also for join estimation via wavelets, consider the following example:

$$\mathcal{T}_1 = \{(v_1, 2), (v_2, 0), (v_3, 7), (v_4, 2)\} \quad \mathcal{T}_2 = \{(v_1, 10), (v_2, 10000), \ldots\}$$

[4] The only exception to this are scenarios involving band-joins [DaJB93] with sufficiently large sizes of the band interval.

Even if the approximation keeps all coefficients necessary to represent $\mathcal{T}_2$ and drops only a single coefficient of the representation of $\mathcal{T}_1$, the approximation of the join between the two distributions exhibits a large error, for the approximation $\hat{\mathcal{T}}_1 = \{(v_1, 1), (v_2, 1), (v_3, 7), (v_4, 2)\}$ now joins the 10000 $\mathcal{T}_2$ tuples with value v_2. The reason for this phenomenon is the fact that the *thresholding scheme* employed in [CGRS00] minimizes the overall mean squared error $\sum_{i=1}^{\mathcal{T}}(f_i - \hat{f}_i)^2$ for each relation, which minimizes the error regarding range selection queries, but disregards accurate join estimation.

In general, regardless of which data approximation technique is used, it holds that it is not feasible to estimate arbitrary join queries from approximations of the joining base relations with acceptable accuracy. This phenomenon was first discussed extensively in [CMN99] in the context of sampling:

Theorem [CMN99]: Assume the samples S_1, S_2 of two joining attributes $R_1.A_1, R_2.A_2$, with S_1 sampling the fraction $1/\epsilon_1$ of R_1, and S_2 sampling the fraction $1/\epsilon_2$ of R_2. Both samples do not capture the entire data distribution, i.e. $\epsilon_1 > 1$ and $\epsilon_2 > 1$. Then it is not possible to generate a sample of the join $R_1 \overset{A_1=A_2}{\bowtie} R_2$ for any fraction $1/\epsilon_j > 0$ from S_1, S_2.

A similar result can be formulated for all other estimation technqiues as well. In the case of spline synopses it holds that

Theorem 2. *Given two synopses over two relations $R_1.A_1, R_2, A_2$, approximating data distributions $\mathcal{T}_1 = \{(v_1, f_1) \ldots, (v_n, f_n)\}, \mathcal{T}_2 = \{(v'_1, f'_1) \ldots, (v_k, f'_k)\}$ with the attribute value domain being $\mathbb{N}$ and rounding used in estimation. Assume that $\mathcal{T}_1$ is captured exactly by a spline synopsis. Further assume that the attribute frequency approximation for $\mathcal{T}_2$ is exact, and the attribute density approximation has error $d\_opt\_err_{1,m'} \leq (\min\{n, k\})$, with m' being the number of buckets used in the density approximation for $\mathcal{T}_2$. Then the estimation of the number of values that find a join partner may be off by $no\_match := \min\{n, k\} - m'$, potentially resulting in an error for estimating the number of tuples in the join result of*

$$join\_error\_max := \max_{\substack{i_1,\ldots,i_{no\_match} \in \{1,\ldots n\} \\ \forall t \in \{1,\ldots,no\_match-1\}: i_t < i_{t+1}}} \left(\max_{\substack{l_1,\ldots,l_{no\_match} \in \{1,\ldots k\} \\ \forall t \in \{1,\ldots,no\_match-1\}: i_t < i_{t+1}}} \left(\sum_{j=1}^{no\_match} f_{i_j} \cdot f'_{l_j} \right) \right). \tag{3.17}$$

Proof: To show this, I first illustrate how to construct example data distributions $\mathcal{T}_1, \mathcal{T}_2$ for a given number of buckets m' used in density approximation for $\mathcal{T}_2$ so that the join-estimation error is equal to the one stated in Term 3.17. Then I show that this term constitutes an upper bound for the join error, i.e., that it is not possible to construct distributions resulting in greater join error.
Let $\mathcal{T}_2 = \{(v'_1, f'_1) \ldots, (v_k, f'_k)\}$ with $k = 3 \cdot m$ and the attribute values defined as follows:

$$\text{for } l = 0, \ldots m' - 1 : v'_{3l+1} = 4l \cdot k, v'_{3l+2} = v'_{3l+1} + 1, v'_{3(l+1)} = v'_{3l+1} + 5$$

Consequently, the final partitioning always groups the values $v'_{3l+1}, \ldots, v'_{3(l+1)}$ in a single bucket (otherwise there is a least one bucket containing values v'_{3l+1}, v'_{3l} and either v'_{3l+2} or v'_{3l-1} for some $l \in \{l = 1, \ldots m' - 1\}$, resulting in an estimation error of $d\_opt\_err_{1,m'} \geq (k)^2$; thus, this is not the optimal partitioning).

This partitioning results in an error of less than 2 per bucket and thus $d\_opt\_err_{1,m'} \leq (k)$ and the value for the fitting parameter $\Delta_i = 1.8$ for any bucket. Therefore the approximation $\hat{v}'_i, i = 1, \ldots, k$ of the attribute values has the following properties:

$$\text{for } l = 0, \ldots m' - 1 : \hat{v}'_{3l+1} = v'_{3l+1}, \hat{v}'_{3l+2} = v'_{3l+2} - 1, \hat{v}'_{3(l+1)} = v'_{3(l+1)} - 1.$$

So out of three values in each bucket, only the first is approximated correctly. Now choose the frequencies $f'_i, i = 1, \ldots, k$ in $\mathcal{T}_2$ in such a way, that for $f_i = 1$ for all $i \mod l = 1$ (i.e. the corresponding value is approximated exactly), and $f_i > 1$ otherwise. The the $2m' = \min\{k, n\} - m'$ values that are not estimated exactly do not find a join partner when estimating the join.

Finally, define $\mathcal{T}_1 := \{(v'_1, f'_1), \ldots, v'_k, f'_k\}$ (i.e. identical to $\mathcal{T}_2$); then the attribute values corresponding to the $\min\{k, n\} - m'$ largest frequency join with each other. So the join is estimated as $join\_estimate = k/3$ with the real size being $join\_exact = k/3 + join\_error\_max$. Analogously, if $\mathcal{T}_1$ is defined as $\mathcal{T}_1 = \{(\hat{v}'_1, f'_1), \ldots, \hat{v}'_k, f'_k\}$ the estimated join size is $k/3 + join\_error\_max$ and the correct result size $k/3$.

On the other hand, the join error cannot grow beyond the value of Term 3.17, when the frequency distribution is captured correctly. This is due to the fact that no matter how large the error for value density estimation, the smallest values in each bucket are always estimated correctly. As a consequence, in the above example at least m' values find the correct join partner. Therefore the estimation of the number of values that find a join partner may be off by $\min\{k, n\} - m'$ at most and thus the overall error for estimating the result size is limited to the $\min\{k, n\} - m'$ largest combinations of frequencies. □

The key result here is that the estimation error for join estimation is not limited directly by (a function of) the estimation error for the attribute frequencies or attribute densities, i.e., it is not possible to formulate an upper bound for the join error $join\_error\_max$ of the nature $join\_error\_max \leq g(f\_opt\_err_{n,m}, d\_opt\_err_{n,m'})$ for some function g. For this reason, I now introduce join-synopses dedicated to capturing the distributions present in the join result, thereby limiting the error for join-estimation. Through these join synopses it is then possible to limit the join-estimation error by limiting the estimation error for value frequency and value density estimation.

3.4.4 Join Synopses

In order to provide accurate estimation for equi-joins and also other types of joins, I extend the idea proposed in [APR99] for sampling to capture join-result distributions in a special class of join synopses. This approach is feasible because there is typically only a small number of attribute combinations (when compared to the number of all possible attribute combinations) that are used in join-operations. Therefore, the number of additional join synopses typically is of cardinality similar to the number of synopses itself.

There are two types of join synopses: *join density synopses*, which approximate the distribution attribute values common to both join attributes and thus present in the join themselves (with the corresponding frequencies computed via the base relations) and *full join synopses* which capture the frequency distribution of the join as well.

Definition 3.2 (Join Density Synopses) :

A *join density synopsis* for the join $R_1 \overset{A_1=A_2}{\bowtie} R_2$ is an attribute density approximation of the set $\mathcal{J} = \{v_1, \ldots, v_l\}$ of values that are present in both the data distribution of $R_1.A_1$ and that of $R_2.A_2$.

When estimating a join $R_1 \overset{A_1=A_2}{\bowtie} R_2$ query using a *join density synopsis*, initially the approximate value distribution $\hat{\mathcal{J}} = \{\hat{v}_1, \ldots, \hat{v}_l\}$ is computed. Then the value frequencies are computed using the frequency functions $frequency_{R_1.A_1}, frequency_{R_2.A_2}$ (see Section 3.4.2) described by the spline functions used to approximate the frequency distributions of the joining attributes:

$$\hat{\mathcal{F}} = \{frequency_{R_1.A_1}(\hat{v}) \cdot frequency_{R_2.A_2}(\hat{v}) \mid (\hat{v}) \in (\hat{\mathcal{J}})\}.$$

As a result, the worst-case error for join estimation is reduced significantly.

Theorem 3. *Given two synopses over two relations $R_1.A_1, R_2.A_2$ which approximate the data distributions $\mathcal{T}_1 = \{(v_1, f_1) \ldots, (v_n, f_n)\}, \mathcal{T}_2 = \{(v'_1, f'_1) \ldots, (v_k, f'_k)\}$ with the attribute value domain being $\mathbb{N}$ and rounding used in estimation. Then the subscripts of the joining values in $R_1.A_1$ and $R_2.A_2$ are $\mathcal{I} = \{(i,j) \mid \ v_i = v'_j, i \in \{1, \ldots n\}, j \in \{1, \ldots k\}\}$.*

Consider the set of values finding a join partner $\mathcal{J} = \{v_i \mid \exists j \in \{1, \ldots k\} : (i,j) \in \mathcal{I}\}$. When $R_1 \overset{A_1=A_2}{\bowtie} R_2$ is estimated using a join density synopsis approximating $\mathcal{J} \subseteq \mathcal{V}_{R_1.A_1}$ as $\hat{\mathcal{J}} = \{\hat{v}_{l_1}, \ldots, \hat{v}_{l_{|\mathcal{J}|}}\}$, the largest possible error is the size of the join estimation can be described as

$$join\_error\_max := \sum_{(i,j)\in\mathcal{I}} |frequency_1(\hat{v}_i) \cdot frequency_2(\hat{v}_i) - f_i \cdot f'_j|. \tag{3.18}$$

Proof: The size of the join result $R_1 \overset{A_1=A_2}{\bowtie} R_2$ is the sum of the product of the frequencies of the values finding a join-partner: $join\_size := \sum_{(i,j)\in\mathcal{I}} f_i \cdot f'_j$. The size of its estimation is then the estimation of the above sum $join\_size\_est := \sum_{(i,j)\in\mathcal{I}} frequency_1(\hat{v}_i) \cdot frequency_2(\hat{v}_i)$. If the estimation of all the frequencies in the join result are either over- or underestimated (i.e., $\forall (i,j) \in \mathcal{I} : f_i \cdot f'_j \geq frequency_1(\hat{v}_i) \cdot frequency_2(\hat{v}_i) \vee \forall (i,j) \in \mathcal{I} : f_i \cdot f'_j \leq frequency_1(\hat{v}_i) \cdot frequency_2(\hat{v}_i)$), the difference $|join\_size - join\_size\_est|$ is equal to Term 3.18. □

Thus, the join estimation error results from inaccuracies in the attribute frequency approximation only. The error in join size estimation is bounded by the sum of the errors in estimation of the frequency approximation for all frequencies $f^{join}_{i,j} = f_i \cdot f'_j$ present in the join result.

However, the above formulation conceals one additional problem: while join density synopses result in correct estimation of the number of joining attribute values (over the entire value domain), the fact that the frequencies present in the join result are estimated through the frequency approximation of the base relation results in the matching problem discussed for range-selection queries in Section 3.3.4. Unlike the case of the matching problem for range-selection queries, using the approximate attribute values as additional points for the spline-fitting problem is not possible without making the frequency approximation more difficult, for the approximate values in the approximation of joining tuples $\hat{\mathcal{J}}$ are not necessarily a subset of the approximate values in the approximation of the attribute density $\hat{\mathcal{V}}$ for either relation. Thus, for every join considered in the fitting of value frequencies, the number of values for which the fitting is computed potentially increases by $|\hat{\mathcal{J}}|$, significantly increasing the resulting overhead and decreasing the accuracy of the resulting frequency approximation.

This problem can be avoided by separating the approximation of frequencies in the base relations from the frequency approximation for the join result.

Definition 3.3 (Full Join Synopses) :

A *full join synopsis* for a join $R_1 \overset{A_1=A_2}{\bowtie} R_2$ is an approximation of the data distribution $\mathcal{T}_J = \{(v_1, f_1), \ldots, (v_l, f_l)\}$ of the join result. It includes both attribute frequency and density approximation.

Theorem 4. *Given two synopses over two relations $R_1.A_1, R_2.A_2$ which approximate the data distributions $\mathcal{T}_1 = \{(v_1, f_1) \ldots, (v_n, f_n)\}, \mathcal{T}_2 = \{(v'_1, f'_1) \ldots, (v_k, f'_k)\}$ with the attribute value domain being $\mathbb{N}$ and rounding used in estimation. Then the subscripts of the joining values are $\mathcal{I} = \{(i,j) \mid \ v_i = v'_j, i \in \{1, \ldots n\}, i \in \{1, \ldots k\}\}$.*

Consider the set of values finding a join-partner $\mathcal{J} = \{v_i \mid \exists j \in \{1, \ldots k\} : (i,j) \in \mathcal{I}\}$. Thus the data distribution of the join result is $\mathcal{T}_J = \{(v_i, f_{i,j}) \mid (i,j) \in \mathcal{I} \wedge f_{i,j} = f_i \cdot f'_j\}$. When $R_1 \overset{A_1=A_2}{\bowtie} R_2$ is estimated using a full join synopsis approximating $\mathcal{T}_J$ as $\hat{\mathcal{T}}_J = \{(\hat{v}_i, \hat{f}_{i,j}) \mid (i,j) \in \mathcal{I}\}$, the largest possible error for the join size estimation is

$$join\_error\_max := \sum_{(i,j) \in \mathcal{I}} |\hat{f}_{i,j} - f_i \cdot f'_j|. \tag{3.19}$$

Proof: This proof is analogous to the one of Theorem 3. The size of the join result $R_1 \overset{A_1=A_2}{\bowtie} R_2$ is the sum of the product of the frequencies of the values finding a join-partner: $join\_size := \sum_{(i,j) \in \mathcal{I}} f_i \cdot f'_j$. The size of its estimation is then the sum of the approximate frequencies in the approximation of $\mathcal{T}_J$: $join\_size\_est := \sum_{(i,j) \in \mathcal{I}} \hat{f}_{i,j}$. If the estimation of all the frequencies in the join result are either over- or underestimated (i.e., $\forall (i,j) \in \mathcal{I} : f_i \cdot f'_j \geq \hat{f}_{i,j} \vee \forall (i,j) \in \mathcal{I} : f_i \cdot f'_j \leq \hat{f}_{i,j}$, the difference $|join\_size - join\_size\_est|$ is equal to Term 3.19. □

Because of the drawbacks of density join synopses discussed above, I only consider full join synopses in the following sections.

The overall drawback of using additional join synopses is that they compete for the limited memory reserved for synopses overall. Thus, by creating and storing additional join synopses, the quality of frequency and density approximation is reduced elsewhere. For this reason, join synopses should only be constructed in cases, where "normal" join estimation as described above is sufficiently inaccurate to justify the required space. In the following (Section 3.4.5), I introduce a self-tuning algorithm which computes join synopses only in this case. To assess the error for join estimation for query result estimation, it is necessary to define a new error metric (Section 3.4.4). Based on the values of this error the algorithm then chooses for which joins to construct join synopses and chooses their sizes (Section 3.4.5). Finally, the improvement resulting from this approach and the different types of join synopses is evaluated (Section 3.4.6).

Computing the Join Error

To assess the improvement from the use of additional join synopses it is necessary to define an error metric for a join estimation, i.e., the error between a join result J and its approximation $\hat{J}$.

For the purpose of selectivity estimation, it might be sufficient to use the deviation in the number of tuples; however, this is not sufficient for other purposes, since the approximate join-result might be subject to further operations. This means that both the deviation in the individual tuple frequencies and the deviation in the attribute density distribution of the join-attribute values that are present in the join result have to be taken into account. In addition, the approximated and the actual join result may contain a different number of unique attribute values. A priori it is unclear how to match the values in the approximated join with the values that occur in the actual result. Simply matching the i-th value in each distribution is not satisfactory, even if J and $\hat{J}$ contained the same number of values.

Example 3.4: To illustrate this point, consider the case where $J = \{(v_1, f_1), \ldots, (v_{n-1}, f_{n-1})\}$ and $\hat{J} = \{(v_2, f_2), \ldots, (v_n, f_n)\}$, which would incur a large error if tuples were simply

matched pairwise in the above orders, although the join result and its approximation match perfectly except for one value.

Instead, individual tuples should be matched by their join-attribute-value distances to compute the error between the approximation and the actual result. For tractability this is done in a greedy fashion:

1. First, match all tuples of equal attribute values, i.e. all $(v_a, f_a) \in J, (\hat{v}_b, \hat{f}_b) \in \hat{J}$ where $v_a = \hat{v}_b$.
2. Then, match the tuples $(v_a, f_a) \in J, (\hat{v}_b, \hat{f}_b) \in \hat{J}$ where the distance between v_a and $\hat{v}_b$ is minimal. This is repeated until only tuples from either J of $\hat{J}$ remain.
3. Finally $||J| - |\hat{J}||$ unmatched tuples remain. Each is matched with an artificial value with frequency 0, located in the center of the value set of the join attribute.

Using a merge-based approach, it is possible to compute this matching efficiently. Initially, both J and $\hat{J}$ are merged into a single list sorted by the attribute value of each tuple; in this process, all exact matches are eliminated (step 1). For each matching in step 2, only the up to $2 \cdot \min\{|J|, |\hat{J}|\}$ distances between tuples adjacent in the resulting list, and coming from different sets are considered; these are stored in a *priority queue* [MNU97]. Each time a pair of tuples is matched, these are eliminated from the list, and at most one new distance needs to be computed. Therefore the computational complexity of the three steps is (with $p = \min\{|J|, |\hat{J}|\}$ and $l = \max\{|J|, |\hat{J}|\}$): $O(p)$ (steps 1&3) and $O(l \log_2 l)$ (step 2). All steps have to be executed once for each possible allocation of join buckets, so if M denotes the overall number of buckets and k the number of different attributes in the approximation, the matching has to be carried out $M - 2 \cdot k$ times, resulting in the overall complexity of $O((M-k) \cdot (p + l \log_2 l))$.

Once this matching is obtained, the overall error is computed analogously to the error for frequency and density distributions, namely, by computing the relative difference in value and frequency for each matching pair of tuples. The normalizing factors *f_join_domain* and *v_join_domain* are computed as before (see Section 3.3.2).

$$
\begin{aligned}
\mathit{join\_error} \quad := \quad & \sum_{\text{all matched tuples } (a,b)} \left(\frac{v_a - \hat{v}_b}{\mathit{v\_join\_domain}}\right)^2 + \left(\frac{f_a - \hat{f}_b}{\mathit{f\_join\_domain}}\right)^2 \\
+ \quad & \sum_{\text{all unmatched tuples } q \text{ in } J} \left(\frac{v_q - (v_{|J|} - v_0)/2}{\mathit{v\_domain}}\right)^2 + \left(\frac{f_q - 0}{\mathit{f\_join\_domain}}\right)^2 \\
+ \quad & \sum_{\text{all unmatched tuples } \hat{q} \text{ in } \hat{J}} \left(\frac{\hat{v}_{\hat{q}} - (v_{|J|} - v_0)/2}{\mathit{v\_domain}}\right)^2 + \left(\frac{\hat{f}_{\hat{a}} - 0}{\mathit{f\_join\_domain}}\right)^2
\end{aligned}
$$

3.4.5 Integrating Join Synopses

Join synopses have been shown to limit the worst-case estimation error for join-queries; still, because they compete for the memory used by spline synopses, they should only be added, when the estimation of the corresponding join improves significantly through this. Integrating dedicated join synopses for selected equijoins again means solving the problem of how to divide memory between the approximations over all synopses. In the following, I explain how to determine whether a join synopsis for a particular equijoin $R_i \overset{A_c=A_d}{\bowtie} R_j$ should be included and how many buckets this join synopsis receives. As before, I assume that there is memory for a total of M buckets. Further, let $\mathcal{V}_{join}$ denote the density distribution of the join result.

Assuming that the problem of dividing up the memory for the density and frequency synopses for all k attributes (see Section 3.3.3) is solved already. This means that the optimal partitioning for each attribute and corresponding errors for all numbers of buckets (overall) less or equal to M are known. Now the error resulting from estimating the join using no additional join synopsis is computed. For this purpose, I compute the approximate join $R_i \overset{A_c=A_d}{\bowtie} R_j$ using the straightforward join computation described in Section 3.4.3 and then compute the resulting error, which is referred to as $standard\_join\_err$.

Furthermore, for $l = 1, \ldots, M - 2k$, the join approximation resulting from using l buckets for an additional join synopsis is computed as follows. For each l, the algorithm computes the join-synopsis using l buckets, and the resulting approximate data distribution $\hat{\mathcal{T}}_l$. Based on $\hat{\mathcal{T}}_l$, the corresponding $join\_error$ is computed, which I refer to as $join\_err_l$. Also, I define $join\_err_0 := standard\_join\_err$.

Since essentially the same error metrics are used for join-result and base-data distributions, their values may be compared directly. Then, the problem of partitioning the available memory again means finding l such that the overall error

$$final\_syn\_err := overall\_f\_d\_err_{M-l} + join\_err_l \text{ is minimal.}$$

For multiple joins, the optimization can be extended analogously to Section 3.3.3; the problem of partitioning the available memory M for k synopses and h possible joins is again a dynamic programming problem, requiring $O(M^2 \cdot (2 \cdot k + h))$ operations.

3.4.6 Experimental Evaluation of Join Synopses

As shown in Section 3.4.4, the use of join synopses has significant impact on limiting the worst-case error for join estimation. To examine how much the use of dedicated join synopses also impacts join estimation in practice, I consider the following experimental setup. Given two single-attribute relations R_1 and R_2 corresponding to the data distributions $\mathcal{T}_1$, $\mathcal{T}_2$ over the value domains $\mathcal{D}_{R_1} = \mathcal{D}_{R_2} := \{1, \ldots, 100\}$. $\mathcal{T}_1$ are $\mathcal{T}_2$ defined so that both distribution contain 40 randomly chosen values with $|\mathcal{V}_{R_1} \cap \mathcal{V}_{R_2}| = 20$ (i.e., 20 values find a join partner in the equi-join $R_1 \bowtie R_2$). The frequencies of the joining tuples are chosen uniformly between 10 and 50.

For these distributions, I then construct two different combinations of synopses for a given amount of memory M:

(a) A synopsis of each $\mathcal{T}_1$ and $\mathcal{T}_2$.

(b) A synopsis of $\mathcal{T}_1$ and $\mathcal{T}_2$ and a full join synopsis of $R_1 \bowtie R_2$.

The sizes for the different synopses are chosen in such a way that the synopses for $\mathcal{T}_1$ and $\mathcal{T}_2$ have equal size (if $M \div 3$ is uneven, $\mathcal{T}_1$ receives the extra bucket) and the size of the join synopsis is computed as detailed in Section 3.4.5. I then measure the error for estimating the number of tuples in $R_1 \bowtie R_2$ for different values of M in Figure 3.4. Even for trivial join-synopses (when 18 values of storage are used, the join-synopsis has only a single bucket for each frequency and density estimation), the use of a join-synopsis reduces the estimation error by several orders of magnitude.

This is consistent with other experiments on similar scenarios conducted with and without join-synopses. As a consequence, when discussing the physical design problem for data synopses in Chapter 5, join synopses will be used for all joins occurring in the workload.

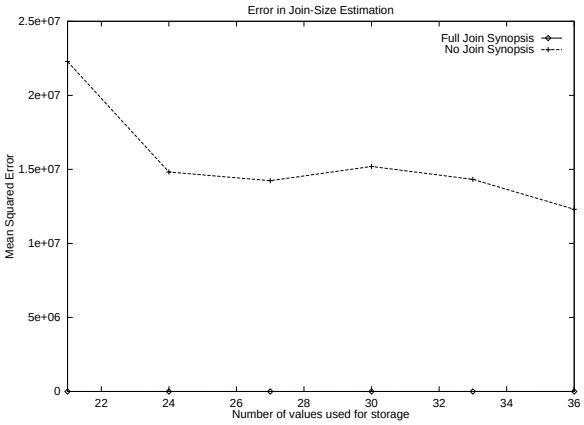

Figure 3.4: Error in join size estimation with and without join synopses

3.5 Experiments

In this section, I examine the accuracy on spline synopses for different estimation problems on synthetic and real-life datasets, comparing them to existing techniques. In order to make a fair comparison, I compare the estimation error when giving the same amount of storage space to each technique.

3.5.1 Experimental Setup

The Techniques: I compare spline synopses with the following existing histogram techniques: *equi-width* and *equi-depth* [PSC84], *MaxDiff(V,A)* [PIHS96], and *V-Optimal(V,F)* [JKM+98] histograms. The storage requirements are 3 values (number of distinct attribute values, average frequency and largest attribute value) for a one-dimensional histogram bucket.

For workloads consisting solely of range selections, I also examine the accuracy of *discrete transform techniques* based either on the Wavelet transform [MVW98], using Haar Wavelets, or the discrete cosine transform (DCT) [LKC99] (because both techniques do not approximate the attribute-value density, they are not suitable for projection and join approximation). To determine which Wavelet coefficients to keep for the first technique, I use the first thresholding method introduced in [MVW98], keeping the largest M coefficients for the experiments shown here (the other three thresholding methods show only insignificant changes in the approximation accuracy for the data sets used here). For the discrete cosine transform, the kept coefficients are selected by reciprocal sampling, which was shown to perform best among all filtering methods presented in [LKC99]. For both techniques we store 2 values for each coefficient, storing the coefficient's value and its index/position.

These techniques are compared with spline synopses using the *OPTIMAL, GREEDY-MERGE* and *GREEDY-SPLIT* partitioning techniques. As described in Section 3.3.1, the storage is 3 values per bucket for the frequency approximation (lowest value, two coefficients of the frequency-function) or density approximation (lowest value, number of unique values, Δ_i). For a fair comparison, in every experiment instances of each technique that consume the same amount of storage are compared to each other.

Datasets: The experiments with one-dimensional distributions are carried out with the following representative datasets: two real-life datasets provided by the KDD Data Archive of the University of California [KDD] and a synthetic one serving as a stress test for the different spline

techniques. The two real-life data sets are single attributes of the *Forest CoverType* data set (581012 tuples, 54 attributes, 75.2MB data size) of which I use (1) the attribute *elevation* (1978 unique values, medium distortion of frequency and density distribution) and (2) the attribute *horizontal_distance_to_hydrology* (189 unique values, small distortion of frequency and density distribution). Finally, I use a synthetic data set with randomly chosen frequencies (uniform distribution) and regular density distribution (1000 unique values). Because there is no correlation between adjacent frequency-values, this results in buckets with very small linear correlation r (see formula 3.2) and thus constitutes a stress test for the frequency approximation by linear splines. The datasets are visualized in Figure 3.5.

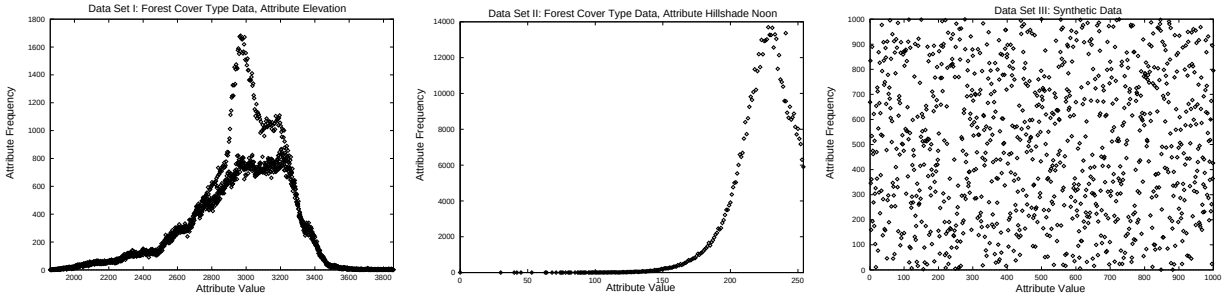

Figure 3.5: The one-dimensional datasets used in the experiments

Query Workload: First, an approximation of each data set is computed, which is then used for three different estimation tasks:

(a) Estimating the size of a self-join. A self join query is used so that the matching problem discussed in Section 3.4.3 does not affect the join-estimation, because every value has a join partner. Thus the accuracy of join estimation for self-joins depends solely on the attribute value frequency approximation. For any other type of join, techniques that do not use dedicated join synopses perform much worse than spline synopses (see Section 3.4.6). To enable a fair comparison between all techniques, the spline synopses constructed do not contain a join synopsis.

(b) Estimating the size $|\mathcal{V}|$ range-selection queries of the type `SELECT * FROM` R_1 `WHERE` $R_1.A_1 < v$, which are issued for all values $v \in \mathcal{V}$. The accuracy for this type of estimation depends on both the attribute value frequency approximation and attribute value density approximation.

(c) Estimating the size $|\mathcal{V}|$ projection queries of the type `SELECT DISTINCT * FROM` R_1 `WHERE` $R_1.A_1 < v$, which are issued for all values $v \in \mathcal{V}$. The accuracy of this type of estimation depends on attribute value density approximation.

In each case I measure the mean squared error

$$\textbf{MSE} := \frac{1}{|\mathcal{V}|} \sum_{i=1,\ldots,|\mathcal{V}|} (exact\_size_i - approx\_size_i)^2.$$

3.5.2 Results

The results of the experiments on the two real-life datasets are shown in Figures 3.6 – 3.8. The x-axis shows the size (specified through the number of values stored) given to the different

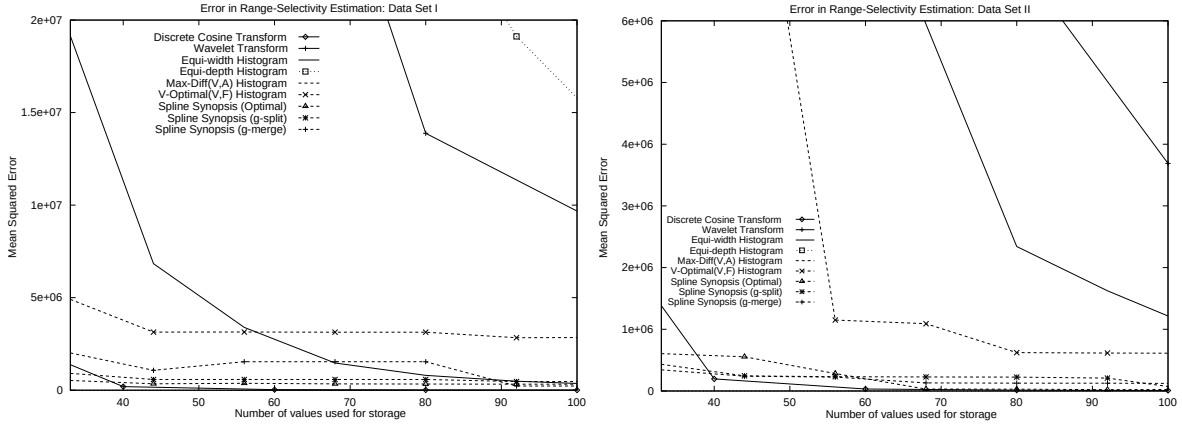

Figure 3.6: Accuracy of different techniques for range queries on two real-life datasets

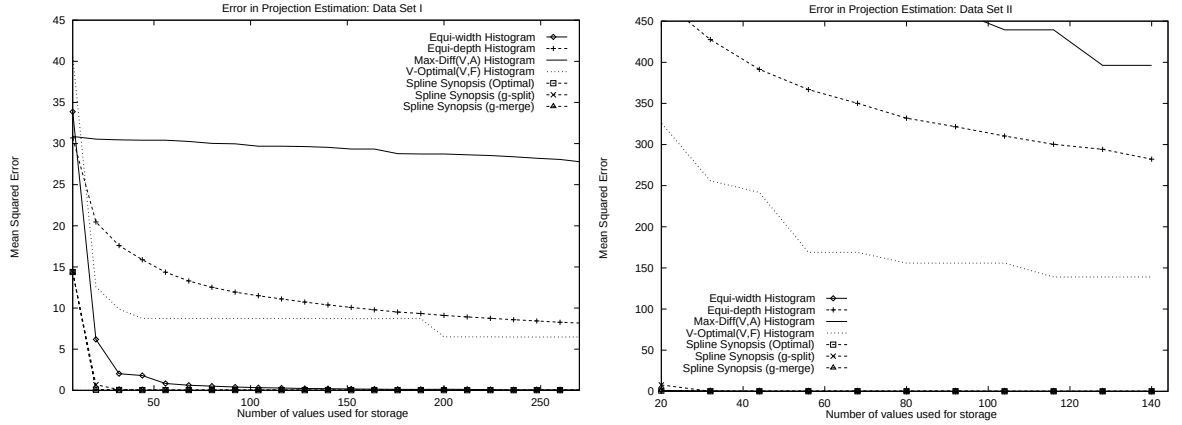

Figure 3.7: Accuracy of different techniques for projection queries on two real-life datasets

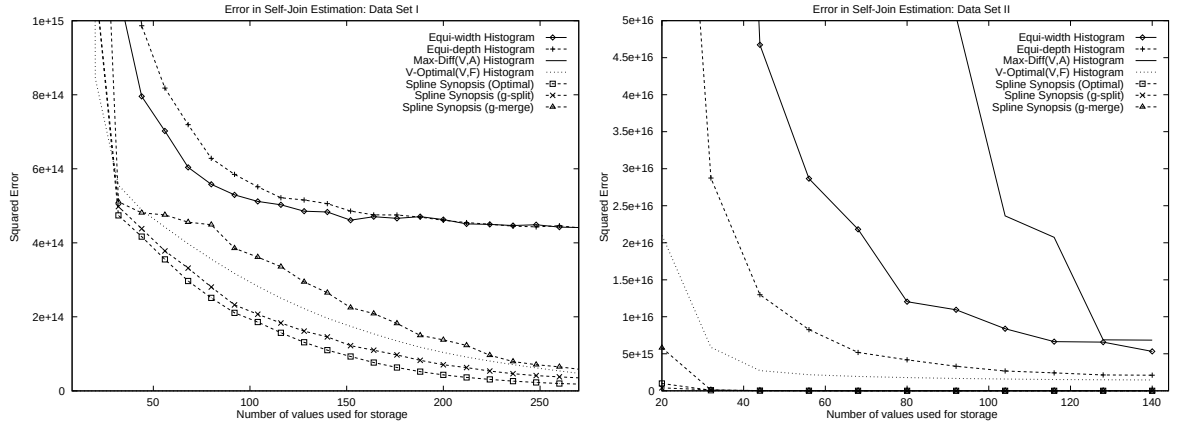

Figure 3.8: Accuracy of different techniques for self-join estimation on two real-life datasets

approximation techniques, which is varied to show how efficiently different techniques use additional memory. The y-axis of each graph shows the MSE for a given query type and size of the approximation.

For projection and self-join queries, the *OPTIMAL* spline techniques consistently outperform all other techniques, followed closely by the *GREEDY* spline variants; the only exception are *V-Optimal(V,F)* histograms, which outperform spline synopses constructed via *GREEDY-MERGE* for self-join estimation on the first dataset. Among the histogram techniques, *V-Optimal(V,F)* histograms perform best, for which the partitioning algorithm is of the same asymptotic complexity as the *OPTIMAL* spline partitioning. For range selections, the spline techniques are again are more accurate than histograms and the estimation by Wavelets; however, when 40 or more values are kept, the DCT technique performs slightly better. To give an intuition why spline synopses outperform the various histogram techniques, consider Figure 3.9. It shows the fitting of the attribute frequencies (with the original data shown as points for each (v_i, f_i)-combination) through spline synopses with 6 frequency-estimation buckets and a *MaxDiff(V,A)* histogram of 8 buckets. Even though the spline synopses uses less buckets, it can be clearly seen that the spline function fits the data much more closely than the average frequency stored in each histogram-bucket.

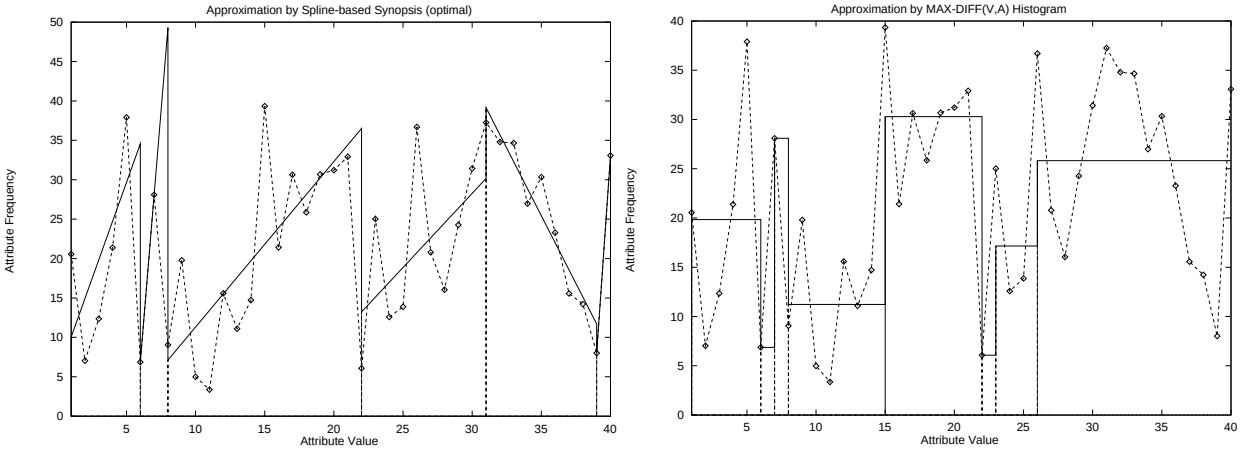

Figure 3.9: Illustration of the approximation quality

When comparing spline synopses to the DCT/Wavelet transform techniques, it is important to note that these techniques are geared specifically to range-selectivity estimation and cannot be used to estimate queries that depend on the attribute-value density of a dataset.

The results for estimating the synthetic "worst-case" data are shown in Figure 3.10 (I omit the results for projection estimation since all techniques capture the density domain with absolute accuracy). Again, the *Optimal* and *GREEDY-SPLIT* spline techniques outperform all competitors other than the DCT technique; since no linear correlation can be exploited, the DCT technique exhibits major gains in this particular experiment.

3.5.3 Running Times

To assess the CPU costs of the algorithms, I measured the running times of approximating the frequency and density distributions, compute their sizes and to compute integrate a synopsis of the self-join result for a single attribute and different sizes of n and M with uniformly distributed v_i and f_i values for execution on a single processor of a *SUN UltraSPARC 4000/5000* (168 MHz), shown in Table 3.1. I measured the CPU time used for partitioning of the frequency (*F-part*) and density domain (*V-part*) for all partitioning methods (*OPTIMAL, GREEDY-MERGE and GREEDY-SPLIT*).

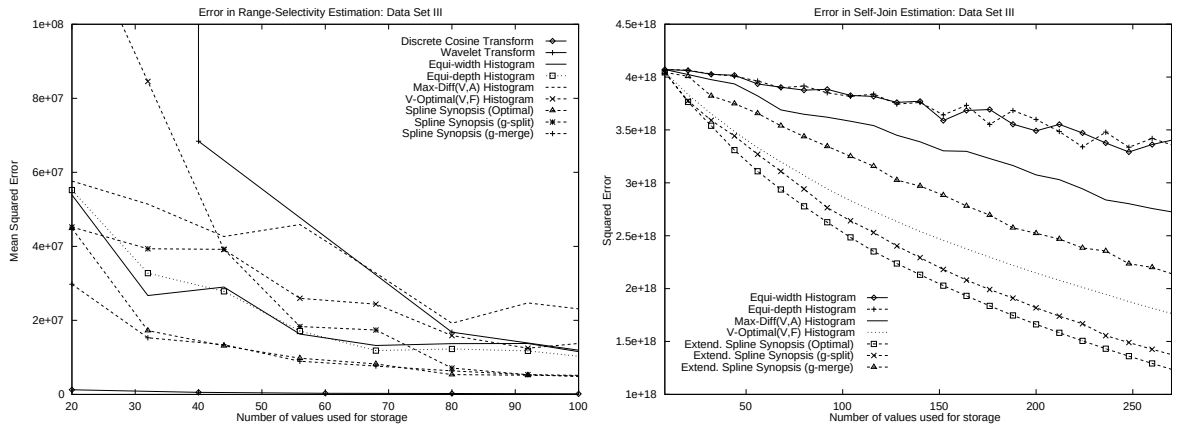

Figure 3.10: Accuracy of different techniques on synthetic "worst-case" data

Method	Step	$n =$	500		1000		4000	
		$M =$	10	50	10	50	10	50
Optimal	F-part.		0.38	1.99	1.98	10.27	51.16	264.63
	V-part.		0.38	1.00	1.97	5.93	41.35	133.43
G-Merge	F-part.		0.012	0.018	0.023	0.033	0.097	0.141
	V-part.		0.008	0.009	0.018	0.020	0.084	0.086
G-Split	F-part.		0.005	0.017	0.012	0.046	0.068	0.268
	V-part.		0.006	0.023	0.016	0.058	0.083	0.271
All	J-error		0.060	0.681	0.256	1.45	1.21	6.41
	M-part.		0.000031	0.00325	0.000031	0.00325	0.000031	0.00325

Table 3.1: Running times in seconds

Computing the join synopsis also entails computing the join-error (*J-error*) for all possible bucket combinations (see Sections 3.4.4 and 3.4.5). Finally, *M-part* gives the time used for determining the final sizes of the different approximations (see Sections 3.3.2 and 3.4.5).

4 Approximation of Multidimensional Correlation

> I could be bounded in a nutshell, and count myself king of infinite space, were it not that I have bad dreams.
>
> – Hamlet

This chapter describes how to extend spline synopses over a single attribute to represent information on the correlation between multiple attributes. Sections 4.1 and 4.2 give an overview of the problems connected with approximating multidimensional data. In Section 4.3 and overview of my novel approach is given. The approach is based on the use of a *space-filling curve* originally proposed by Sierpiński, the properties of which are described in detail in Section 4.4. Section 4.5 then describes how the resulting synopses are used for query estimation. Finally, Section 4.6 contains an experimental evaluation of the overall approach.

4.1 Multidimensional Partitioning and the "Curse of Dimensionality"

A straightforward approach to multidimensional approximation would be to extend our technique through the use of multi-dimensional buckets. Then a multi-dimensional attribute value distribution could be partitioned into buckets of this type, which would then store a compact representation (of constant size) of the contained values and frequencies.

Unfortunately, such an approach is problematic: because the space of possible partitionings increases exponentially with the dimensionality of the data, the problem of finding an optimal partitioning becomes NP-complete already for much simpler error functions than the ones used in spline synopsis construction [MPS99]. In addition, to achieve good approximation of the value density in one-dimensional synopses, it was only necessary that the values contained in a bucket were (approximately) evenly spaced in a single dimension. To achieve good approximation of value density in d dimensions (using constant-size buckets), it becomes necessary for the values to be evenly spaced in *all* d dimensions, which is only the case for high-dimensional distributions. To illustrate the problem, consider the approximation in multidimensional histograms. In each bucket, the number of distinct values present in each dimension is stored. For purposes of estimation it is then assumed that all possible combinations of these values are present, which is known as the (multi-dimensional) *uniform spread assumption*. This leads to consistent overestimation of the number of attribute values (see Figure 4.1) and underestimation of their frequencies, thereby making histograms very inaccurate for projection and join estimation in more than a single dimension. Note that this problem becomes exponentially worse with increase of the dimensionality of the histogram.

The specification of constant-sized buckets requires the formulation of rigid assumptions on the placement of attribute values within a single bucket, which typically do not hold for real-life data distributions. As a consequence, I use a different approach than the one outlined above.

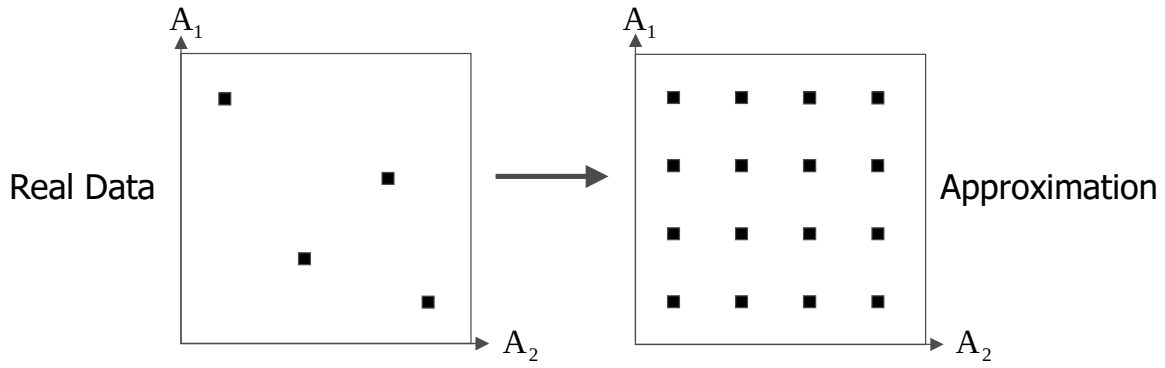

Figure 4.1: Approximation of multidimensional attribute value distributions in histograms

4.2 Preserving Correlation

As discussed in Section 2.4.1, size estimation for every query specifying filter conditions on more than one attribute of the same table requires that the estimation technique maintains the correlation between attribute values and their frequencies accurately. In addition, it is also important to maintain the correlation between values of different attributes as well.

To illustrate this, consider a relation $\mathcal{R}$ with the attributes ORDER-DATE and SHIP-DATE (each coded as an integer). These are strongly correlated, in the sense that the value SHIP-DATE is typically a fixed number of days larger than ORDER-DATE (or at least has very small variance for the difference between these dates). Such a data set (250 tuples, requiring 750 values of storage) is depicted in Figure 4.2: the values for ORDER-DATE were generated uniformly within the interval $[1, 500]$, the values for SHIP-DATE uniformly within $[Order\text{-}Date, Order\text{-}Date + 10]$.

Preserving this type of correlation is crucial for a large number of queries, as the conditions specifying the inclusion/exclusion of attribute values generally depend on the attribute-value distribution. For example a query checking for delayed deliveries:

$$\texttt{SELECT DISTINCT} * \texttt{FROM } R_1 \texttt{ WHERE } R_1.\text{SHIP-DATE} - R_1.\text{ORDER-DATE} > 20.$$

would return no tuples on the original data. Therefore, an estimation technique needs to be able to preserve the correlation in such a way that none or only very few tuples are returned when estimating this query.

Figure 4.3 shows the approximation of the data using 30 values of storage space using a 2-dimensional *MaxDiff(V,A)* histogram (constructed using the *MHIST* technique). It can be seen that the use of the *uniform spread assumption* within buckets limits the ability of histograms (or histogram-based techniques) to express the present correlation.

As a measure of the amount of correlation present between the attribute values of 2 attributes $R.A_1, R.A_2$, I use the *Spearman rank-order correlation coefficient* [PTVF96], which is defined as follows. Let L_i be the rank of the value of attribute A_1 in the i-th tuple, S_i be the rank of the value of attribute A_2 in the i-th tuple (in case of ties, we use the appropriate *midrank*), and $\bar{L}, \bar{S}$

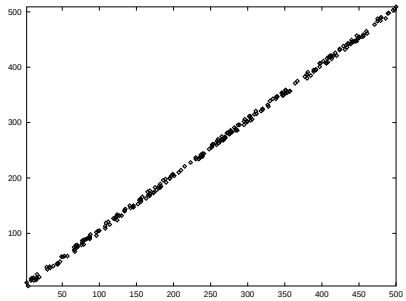

Figure 4.2: Original Data

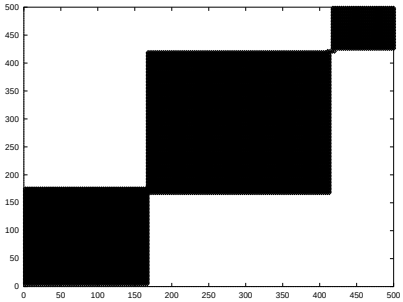

Figure 4.3: MHIST Histogram Approximation

the corresponding average rank. Then the *linear correlation coefficient* of the ranks is defined as

$$r_s = \frac{\sum_{i=1}^{|\mathcal{T}|}(L_i - \bar{L})(S_i - \bar{S})}{\sqrt{\sum_{i=1}^{|\mathcal{T}|}(L_i - \bar{L})^2}\sqrt{\sum_{i=1}^{|\mathcal{T}|}(S_i - \bar{S})^2}}. \tag{4.1}$$

The linear correlation coefficient of the example distributions is $r_s = 0.88$, the approximate distributions resulting from histogram estimation has $r_s = 0.99$, corresponding to the intuitive interpretation of Figures 4.2 and 4.3.

Most approximation techniques are geared towards preserving the correlation between attribute value and the corresponding frequency, but disregard correlation between attribute values.

4.3 Spline Synopses and Mapping Multidimensional Data

First, an injective mapping $\theta : \mathbb{R}^d \mapsto \mathbb{R}$ is applied to the attribute-value domain of the base data, reducing the d-dimensional distribution to a one-dimensional one, which is then approximated by the techniques introduced previously. We store the resulting approximation; when it is used to estimate a multi-dimensional query, the approximated data is mapped back to $\mathbb{R}^d$ via the inverse mapping $\psi := \theta^{-1}$.

This approach leverages the advantages of the spline techniques discussed previously (efficiency, accurate representation of both attribute frequency and attribute value density, fast computation) while avoiding the rigid requirements of multidimensional buckets (see Section 4.1), the computational overhead entailed by computing the optimal partitioning of multidimensional space and the lack of accurate value density estimation of transform-based approaches (see Sections 2.4.2 and 3.4.3). The only drawback of this approach is an increase in the overhead necessary for estimation of range conditions, because unlike in the one-dimensional case (where buckets correspond to intervals containing all values in them) all buckets and values have to be examined, as the bucket boundaries do not have a simple geometrical interpretation. This is discussed in detail in Section 4.5. Figure 4.4 shows the interaction between spline synopses and space-filling curves.

4.4 The Sierpiński Space-Filling Curve Construction

Definition 4.1 (Space-Filling Curves) :

A *Space Filling Curve* is a surjective, continuous mapping from the unit interval $I_1 = [0, 1]$ to the d-dimensional unit hyper-cube $I_d = [0, 1]^d$.

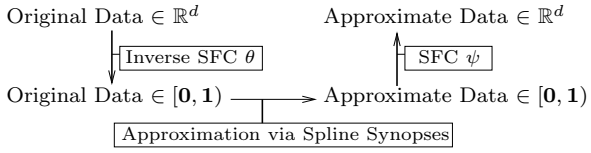

Figure 4.4: Combining Space-Filling Curves (SFC) and Spline Synopses

The first space-filling curve can be traced to Peano [Pea90]. In 1912 W. Sierpiński proposed a curve construction [Sie12] (Figure 4.7) which is the basis for the curve construction used in this thesis. While both the Peano and Sierpiński curves described the mapping $[0, 1] \mapsto [0, 1]^d$ for $d = 2$ only, [SR94] extends the construction by Sierpiński to arbitrary values of d. Space-filling curves have successfully been used in a number of applications [Man83], including high-dimensional indexing [KF94, LK01] and graphics compression [DCOM00]; the Sierpiński curve has also been used for heuristic solutions to the *Traveling Salesman Problem* with Euclidian distances [PB89, GS92] and partitioning problems in VLSI design [AK95].

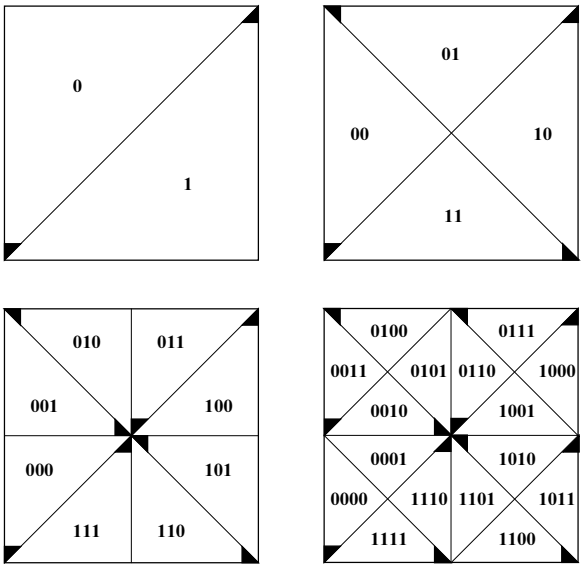

Figure 4.5: Successive partitionings of I_2 into 2^k identical triangles, each containing a k-digit binary label and a marked vertex for $k = 1, \ldots, 4$

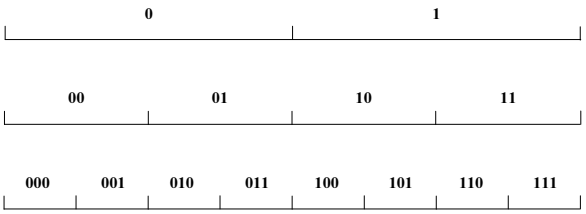

Figure 4.6: Successive partitionings of the unit interval into 2^k identical subintervals, each containing a k-digit binary label. The marked endpoint is always on the left. $k = 1, \ldots, 3$

The original Sierpiński curve can be constructed by successively partitioning I_2 as shown in Figure 4.5. Here, the k-th partition of I_2 consists of 2^k identical triangles, each labelled with the

binary representation of an integer in the range of $0, \ldots 2^k - 1$. One vertex in each triangle is marked to distinguish it from the others. The unit interval I_1 is similarly partitioned, as shown in Figure 4.6, into subintervals, each labelled and marked at one endpoint.

Each successive partition is obtained by bisecting every triangle or subinterval in the previous partition. A new label is created by appending an additional digit to the parent's label; the half containing the parent's marked vertex receives a zero, the other half receives a one. The space-filling curve maps each subinterval onto the subtriangle bearing an identical label and it maps the marked endpoint of the subinterval to the marked vertex of each triangle. The marked vertices are therefore visited in a sequence determined by their labels. The curve resulting from this construction is depicted in Figure 4.7. For the first two instances of the curve in the, the value corresponding to the position of the vertices in I_1 is marked along the curve.

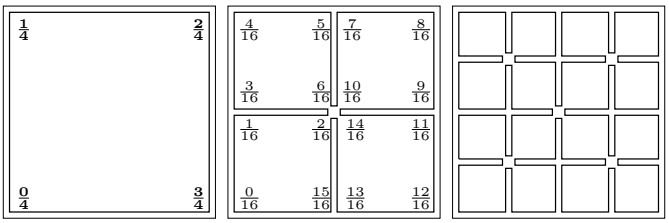

Figure 4.7: The Sierpiński Curve [Sie12] for $d = 2$ and $k = 2, 4, 6$ iterations of the recursive construction

The Sierpiński Curve extension to d dimensions of [SR94] is defined in terms of two *iterated function systems* (IFS) [Bar88]. The first system W generates a sequence of subdivisions on the hypercube. W defines a family of mappings $w_i : \mathbb{R}^d \mapsto \mathbb{R}^d$ as follows:

$$w_i(x_1, x_2, \ldots, x_d) = \begin{bmatrix} 1/2 - (1/2 - \delta_{1,i}) \cdot x_1 \\ 1/2 - (1/2 - \delta_{2,i}) \cdot x_2 \\ \vdots \\ 1/2 - (1/2 - \delta_{d,i}) \cdot x_d \end{bmatrix}$$

where $\delta_{j,i} \in \{0, 1\}, j = 1, \ldots, d, i = 1, \ldots 2^d - 1$. Each of the 2^d members of W is thus uniquely defined by the vector $D_{i,d} = (\delta_{1,i}, \delta_{2,i}, \ldots, \delta_{d,i})^T$. These vectors are now chosen in such a way that vector $D_{i,d}$ corresponds to the *Gray code* [Gra53] value for i of length d. The key idea here is the fact that the Gray codes for two consecutive values of i differ only by one position.

The second system G of 2^d iterated functions $g_i : [0, 1] \mapsto$ subsets of $[0, 1]$ is used to partition the unit interval. G is defined as follows:

$$\begin{aligned} g_i(t) &= b_d \cdot 2^{-2d} + 2^{-2d} \cdot (i - 1) + 2^{-d} \cdot t, \text{ (for } 0 \leq t \leq 1, i = 1, \ldots, 2^d - 1) \text{ and} \\ g_0(t) &= [t \leq (2^d - b_d) \cdot 2^{-d}] - (2^d - b_d) \cdot 2^{-2d} + 2^{-d} \cdot t \text{ (for } 0 \leq t \leq 1) \\ &\quad \text{with } b_1 = 1, b_k = 2^{k-1} - b_{k-1} + 1, k = 2, \ldots, d \end{aligned}$$

Let the unit interval $[0, 1]$ be denoted as I_1 and the unit hypercube as I_d. Now a one-to-one correspondence between the system of hypercubes and the sequence of subintervals is defined by matching the interval defined through $t \in g_{i_1} \circ g_{i_2} \circ \ldots g_{i_k}(I_1)$ with the sub-cube $t \in w_{i_1} \circ w_{i_2} \circ \ldots w_{i_k}(I_d)$. As the number of iterations of both iterated function systems approaches infinity, the space-filling curve is obtained (as the limit of this construction). We denote the resulting surjective mapping as $\psi : I_1 \mapsto I_d$ and its inverse as $\phi := \psi^{-1} : I_d \mapsto I_1$.

The overhead for computing the mapping via either ϕ or ψ of a single point depends on the dimensionality d of the space-filling curve and on the number of iterations t of the IFS. The value t itself is limited by the accuracy of the data type used to store the values v_i and their mapping. To be able to store all possible mapped values after t iterations of the curve construction, the underlying value type needs to be able to store $2^{t\cdot(d-1)}$ different values. Consequently, the size of t corresponds to the size (in bits) of this data type divided by $d-1$ and is thus a small constant. As a consequence, it becomes necessary for synopses over large number of attributes d, to use a different data type for the lowest value in a bucket (for both frequency and density buckets).

Since in any case t is constant, the overhead for computing the mapping of a single value is $O(d)$. When using ϕ to map values from their original domain to I_1 for approximation, the mapped values have to be sorted afterwards. Thus, for a data distribution $\mathcal{T}$, the overhead for approximation increases by $O(d\cdot|\mathcal{T}|)$ plus the overhead for sorting $|\mathcal{T}|$ values. In my experiments, the overhead for sorting generally dominated the mapping costs.

4.4.1 Properties of the Sierpiński Curve

The Sierpiński curve has a number of salient properties important for its use in connection with spline synopses:

- It contains no distortion, i.e., for arbitrary $x, x' \in \mathbb{R}$, the ranges $[x, x+\delta]$, $[x', x'+\delta]$ correspond to pieces of the curve of identical length.
- It is symmetric (unlike the *Hilbert or Peano curve*), i.e. invariant to rotations by multiples of 90 degrees in the direction of each dimensional axis. This means that independent of which dimension the different attributes are mapped to, the approximation quality can be expected to be similar.
- The mapping ψ preserves the *Lebesque measure* in the following sense: for every *Borel* set $A \subseteq I_d$

  $$\mu_1(\phi(A)) = \mu_d(A) \text{ [SR94]},$$

  when μ_1 and μ_d are the *Lebesque measure* in $\mathbb{R}$ and in $\mathbb{R}^d$ respectively. Intuitively, this can be interpreted as the fact that the Sierpiński curve fills I_d with uniform density everywhere.
- The resulting projection ψ preserves distances very well, as ψ is a *Lipschitz continuous* mapping of order $1/d$, in the following sense:

  $$\forall v_1, v_2 \in [0,1] : \|\psi(v_1) - \psi(v_2)\| \leq 2\sqrt{d+6}\ |v_1 - v_2|^{\frac{1}{d}} \text{ [SR94]} \tag{4.2}$$

  with $\|\cdot\|$ denoting the L_2 norm (*Euclidian Distance*) between the data points in $\mathbb{R}^d$. This is optimal in the sense that it can be shown that a surjection from I_1 to I_d that is Lipschitz continuous of order $\sigma > 1/d$ is not possible [Mil80].

The last property is crucial for the use of the Sierpiński curve for the estimation of (multidimensional) attribute value density. Consider the case of a d-dimensional value distribution $\mathcal{V} = \{v_1, \ldots, v_n\}$. Each value $v_i \in \mathbb{R}^d$ is then mapped to a value $v_i^{lin} \in [0,1]$, by normalizing each value (i.e., dividing the i-th component of v_i by the highest value in the corresponding domain, for $i = 1, \ldots, d$) and then applying ϕ. I then approximate the mapped values $\mathcal{V}^{lin} := \{v_1^{lin}, \ldots, v_n^{lin}\}$ as $\hat{\mathcal{V}}^{lin} := \{\hat{v}_1^{lin}, \ldots \hat{v}_n^{lin}\}$, minimizing $\sum_{i=1}^{|\mathcal{T}|}(v_i^{lin} - \hat{v}_i^{lin})^2$. In order to use the resulting approximation for query estimation, the $\hat{v}_i^{lin}$ are mapped back via ψ at query processing time and the normalization reversed (i.e., the i-th component of $\psi(v_i^{lin})$ is multiplied by highest value in the

corresponding domain, for $i = 1, \ldots, d$). Because of Equation 4.2, the final approximation error for a value v_i (i.e., $\|\psi(v_i^{lin}) - \psi(\hat{v}_i^{lin})\|$) is limited by the upper bound of $2\sqrt{d+6}\ |v_i^{lin} - \hat{v}_i^{lin}|^{\frac{1}{d}}$.

Therefore, the process of constructing a synopsis over d-dimensional minimizes an upper bound for the overall approximation error

$$\sum_{i=0}^{|\mathcal{T}|}(f_i - \hat{f}_i)^2 + \frac{v\_domain}{f\_domain} \cdot (\|v_i - \hat{v}_i\|)^2. \tag{4.3}$$

Unlike in the case of one-dimensional synopses, where the overall approximation error is minimized, the synopsis construction is only optimal with regard to the (mean squared) error in I_1, but not with regard to the error in $\mathbb{R}^d$. However the notion of minimizing the upper bound for Equation 4.3 and minimizing the equation itself coincide for all practical purposes when the large data distribution to be estimated becomes large, as the different mappings of a distance in I_1 to different distances in I_d distances average out. In this sense (and the interests of a simple notation), I refer to the spline synopsis construction in higher dimensions as *minimizing* Equation 4.3.

Because our approach means that no rigid assumptions on the placement of attribute values within a bucket are used, the resulting estimation preserves correlation between values of different attributes much better. Consider the example data shown in Figure 4.2. The approximation of this data using a spline synopses requiring 30 values storage is shown in Figure 4.8. Not only does this estimation resemble the shape of the original data much more than the histogram approximation (Figure 4.3), it also has a linear correlation coefficient $r_s = 0.83$ much closer to the original one.

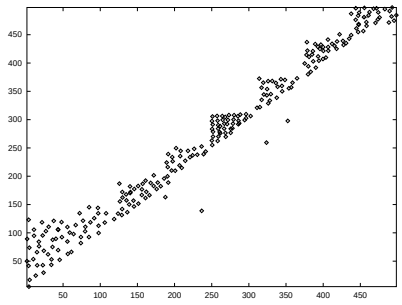

Figure 4.8: Spline Synopsis Approximation

4.5 Using Multidimensional Spline Synopses for Query Estimation

The key difference between multidimensional and one-dimensional spline synopses is that the geometrical interpretation (in I_d) of a bucket's boundaries are fractals themselves (not d-dimensional ranges as with multidimensional histograms). Therefore, information on the bucket boundaries cannot be leveraged efficiently to eliminate buckets from consideration for range-selection queries[1]. Instead, all attribute values have to be materialized and mapped back using ψ before filtering, resulting in $O(d \cdot |\mathcal{T}|)$ overhead, as discussed in Section 4.4. Still, as d is generally a

[1] However, the process of transforming a d-dimensional range query into a set of one-dimensional range-queries, that may then be evaluated using all optimizations introduced in Section 3.4 is straightforward. Initially, all intersections between the curve and the range-boundaries are computed. As the curve itself is a fractal,

small value (with the possible exception of materialized views in large decision-support systems, relations used in databases generally have less than 15 attributes), this approach is still sufficiently fast to be used in query optimization. Otherwise, the overall approach remains the same as for one-dimensional synopses (Section 3.4).

4.6 Experiments

The Techniques: I compare spline synopses with two multidimensional *MaxDiff(V,A)* histograms computed using the *MHIST-2* partitioning algorithm, which were found to be the most accurate histogram technique for multidimensional data [PI97]. In addition, for workloads consisting solely of range selections, I also examine the accuracy of *discrete transform techniques* based either on the Wavelet transform [MVW98], using Haar Wavelets, or the discrete cosine transform (DCT) [LKC99].

Datasets: The experiments on multidimensional distributions were carried out on two data sets: (1) 1970 PUMS Census data, of which we use the attributes *migplac5* and *chborn* (661 unique attribute-value pairs, 70817 tuples), provided by the KDD Archive [KDD], and (2) a synthetic stress test similar to the one used before, with 400 unique value pairs, 10000 tuples, and randomly chosen density distribution and attribute-value frequencies.

Query Workload: First, an approximation of each data set is computed, which is then used for three different estimation tasks:

(a) Estimating the size $|\mathcal{V}|$ range-selection queries of the type `SELECT * FROM` R_1 `WHERE` $R_1.A_1 < v^1$ `AND` $R_1.A_2 < v^2$, which are issued for all values $(v^1, v^2) \in \mathcal{V}$. The accuracy for this type of estimation depends on both the attribute value frequency approximation and the attribute value density approximation.

(b) Estimating the size $|\mathcal{V}|$ projection queries of the type `SELECT DISTINCT * FROM` R_1 `WHERE` $R_1.A_1 < v^1$ `AND` $R_1.A_2 < v^2$, which are issued for all values $(v^1, v^2) \in \mathcal{V}$. The accuracy of this type of estimation depends on attribute value density approximation.

In each case I measure the mean squared error

$$\textbf{MSE} := \frac{1}{n} \sum_{i=1,\ldots,n} (exact\_size_i - approx\_size_i)^2,$$

with n being the number of queries.

4.6.1 Results

The results of the two experiments on multi-dimensional data are shown in Figures 4.9 and 4.10. For range-selectivity estimation the histogram-based techniques consistently outperformed the multidimensional *MaxDiff(V,A)* histograms. Wavelets turned out to be superior to splines for

the number of such intersections is infinite in theory, however, because of the limited accuracy of the data types used to store the mapped values, the curve is only computed for a limited number t of iterations of the curve construction (see Section 4.4), resulting in a limited number of intersections (for a detailed treatment of this topic with regards to the *Hilbert curve*, see [MJFS01]). Now the intervals along the curve connecting two of these intersections (without another intersection in between) encompass values that satisfy the range condition. Furthermore, each of these ranges corresponds to a one-dimensional range in I_1. Thus, the result of the d-dimensional range query can be estimated as the union of the results for all these one-dimensional range queries.

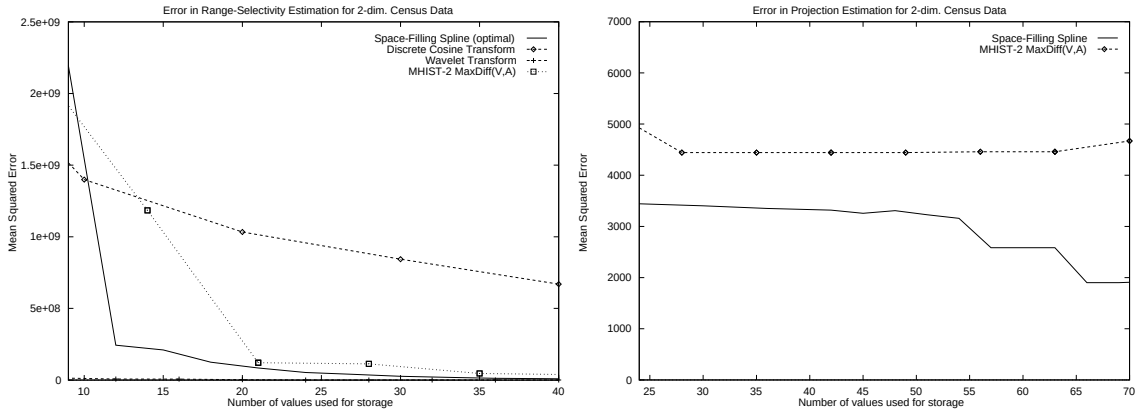

Figure 4.9: Accuracy of different techniques for multidimensional CENSUS data

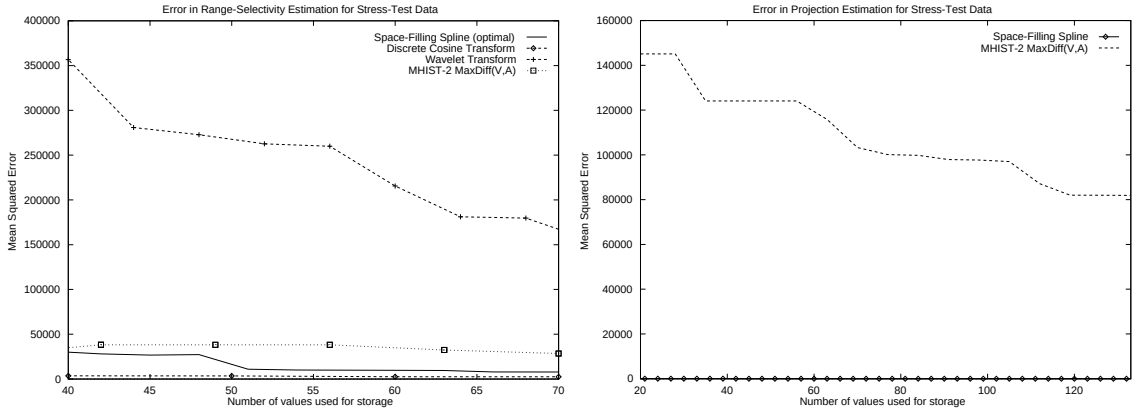

Figure 4.10: Accuracy of different techniques for multidimensional synthetic data

range queries for one experiment and the DCT based technique was the winner in the other experiment. However, the winning technique in one experiment performed very badly in the other one.

The performance of both transform-based techniques depended very much on the sparseness of the data. With approximating data via Haar Wavelets, as used in the experiment, very sparse data results in a large number of coefficients of value 0, which can therefore be dropped without increasing the estimation error. Consequently, the Wavelet technique results in very accurate approximation for sparse data. The opposite is true in dense domains, however; here all other techniques perform better. The two datasets are examples for this behavior. While the Census data contains 661 unique value pairs in a domain of $715 \cdot 13 = 9295$ value pairs, the synthetic data contains 400 value pairs in a domain of $40 \cdot 40 = 1600$ value pairs.

In both experiments spline synopses show the 2nd best estimation accuracy, offering performance competitive with the winning technique.

Regarding projection estimation, multi-dimensional histograms were consistently outperformed by the spline techniques by several orders of magnitude.

5 Physical Design for Data Synopses

> According to an optimist, the glass is half-full,
> according to a pessimist, the glass is half-empty,
> according to an engineer, the glass has twice the necessary size.
>
> – Unknown

5.1 Introduction

In this chapter, I develop a novel framework for and algorithmic solution to the physical design problem for data synopses, covering the entire class of SPJ (i.e., select-project-join) queries. In contrast to the work in [CN00], the approach goes beyond the binary decisions on building vs. not building a certain synopsis by addressing the fundamentally important issue of how much memory each synopsis should be given. This is especially important when the role of statistics management goes beyond choosing good query execution plans, and synopses also serve to predict absolute run-times and result sizes, which in turn is highly relevant in data mining or Web source mediation environments. I characterize the exact solution for the optimal choice of synopses for a given workload. For complexity reasons, I derive various problem-specific heuristics that make my approach reasonably efficient and thus practical.

The chapter is organized as follows. First, it is necessary to introduce additional notation in Section 5.2. In Section 5.3 the underlying optimization problem and the used error model are defined. Section 5.4 then describes how to determine the optimal set of synopses exactly, using two assumptions which (in earlier experiments) have been found to hold for nearly all datasets and which lead to a compact formulation of the necessary computations. For large instances of the optimization problem, I discuss several heuristics to remove potential bottlenecks. In Section 5.7 it is then shown how to combine the various building blocks of the framework into a unified algorithm. Section 5.8 contains an empirical validation of the approach in form of several experiments conducted with the *TPC-H* decision support benchmark.

5.2 Notation

I consider a set of queries $\mathcal{Q} := \{Q_1, \ldots, Q_{|\mathcal{Q}|}\}$. Queries can be "approximately answered" by a set of synopses $\mathcal{S} := \{S_1, \ldots, S_l\}$ corresponding to the data distributions $\mathcal{T}_1, \ldots, \mathcal{T}_l$. Because in our context there is never more than one synopsis for a given set of attributes I also write S_A with A being the set of attributes captured by the synopsis, i.e., $S_{\{R_1.A_2, R_1.A_3\}}$ denotes the synopsis over the two attributes A_2 and A_3 of relation R_1. Analogously, we use the notation $\mathcal{T}_{\{R_1.A_2, R_1.A_3\}}$ to describe the corresponding joint data distribution in the full data set. The size of a synopsis S_A (in terms of the number of values necessary to store S_A) is denoted by $Size(S_A)$.

A simple (range) selection or projection query can be answered using the data distribution of the queried relation over the attributes involved in the range selection. A join query can be

processed by examining the joint data distribution of the joining relations. Thus it is possible to assign to each query Q_i on relation R_p the *minimum* set of attributes $Min(Q_i) \subseteq \bigcup_{R_p \in \mathcal{R}} Att(R_p)$, whose joint data distribution must be examined to answer the query. For example consider the query q_1

$$\texttt{SELECT } R_1.A_1 \texttt{ WHERE } R_1.A_2 > 100.$$

This query can be answered by examining the joint data distribution of relation R_1 over the attributes $R_1.A_1$ and $R_1.A_2$, thus $Min(q_1) = \{R_1.A_1, R_1.A_2\}$.

When only the size of a result is of interest (for example in the context of query optimization), it is sufficient to query the attributes that determine the number of tuples in the result; assuming that no duplicate elimination is performed, in this case the minimum set becomes $Min(q_1) = \{R_1.A_2\}$. Consequently, the set $Min(Q_i)$ contains the information which synopses need to be built in order to answer query Q_i while observing all correlations between the relevant attributes.

As join synopses have been shown to be vital to accurate estimation of joins for nearly all real-life datasets (see Sections 3.4.3 and 3.4.6), I adopt the following approach for all queries in $\mathcal{Q}$ involving joins. For each query Q_i involving one or more joins, I add a "virtual relation" R' to $\mathcal{R}$ representing the joint data distribution of the top node in the corresponding join tree (i.e. the complete n-way join if the join tree has n leaves). A query involving a join could thus be modelled by introducing join synopsis over the relevant attributes from the joining relations; consider query q_2:

$$\texttt{SELECT } R_1.A_1 \texttt{ FROM } R_1, R_2, R_3 \texttt{ WHERE } R_1.A_2 = R_2.A_3 \texttt{ AND } R_2.A_4 = R_3.A_5$$

Here I introduce $R' := R_1 \overset{A_2=A_3}{\bowtie} R_2 \overset{A_4=A_5}{\bowtie} R_3$. Then $Min(q_2) = \{R'.A_1\}$. Note that it is only necessary to introduce this R' if is not present already. If q_2 appears more than once in the workload (either solitary or as a sub-query), R' is still only introduced once. This approach corresponds to introducing a full join synopsis (see Section 3.4.4) for every unique join present in the workload.

5.3 Framework

The *physical design problem for data synopses* is defined as follows:

Definition 5.1 (Physical Design Problem for Data Synopses) :
When given a number of datasets $\mathcal{R} := \{R_1, \ldots, R_n\}$, a workload consisting of SPJ (select-project-join) queries $\mathcal{Q} := \{Q_1, \ldots, Q_k\}$ and a limited amount of available memory M, the *physical design problem for data synopses* is to determine the best combination $\mathcal{S}$ of synopses that can be stored using up to M memory, so that the estimation error over all queries is minimized.

This corresponds to the natural goal of maximizing accuracy for query optimization, approximate query answering, or minimizing the query time when supporting top-k queries [CG99]. If the synopses in $\mathcal{S}$ are only used for query optimization, additional pruning of $\mathcal{S}$ may be beneficial, as synopses that do not influence plan selection can be omitted (see [CN00]); in addition, maximizing the estimation accuracy for the final set of selected synopses (in this case, by tuning their sizes) can further improve query optimization.

In the following, it is assumed that each query Q_i is mapped to exactly one synopsis $S_j \in \mathcal{S}$ which captures all attributes that are relevant for Q_i, i.e., attributes on which filter conditions are

defined as well as attributes that appear in the `SELECT` clause of the query (see Section 2.2). This is no limitation, as it is always possible to decompose a more complex query into subqueries such that the above condition holds for each subquery, when full join synopses are used for each join in the workload. In fact, an SPJ query would often be the result of decomposing a complex SQL query (e.g., to produce an intermediate result for a group-by and aggregation decision-support query). The subqueries that matter in this context are those for which I wish to estimate the result or result size. In commercial query engines and in most of the prior work on data synopses, these subqueries were limited to simple range selections. This approach improves the state of the art in that we consider entire SPJ queries as the building blocks for data synopses.

5.3.1 Storage of Multiple Synopses

In the discussed scenario, the database system has to store multiple synopses over a number of different attribute combinations. Therefore, in addition to the buckets stored by each synopsis itself, the following bookkeeping information needs to be stored to construct a directory of all synopses:

(1) the relation approximated by the synopsis,
(2) the combination of attributes approximated by the synopsis (coded as a bit vector),
(3) the total number of buckets,
(4) the number of buckets used for frequency approximation,
(5) the highest attribute value stored in the synopsis/relation, and
(6) a pointer to the synopsis itself.

Consequently, each synopsis requires additional 6 values storage for this bookkeeping information. As discussed in Section 4.4, the values used for (2) and (5) may be represented by a different (i.e., larger) data type than the other values, when the number of attributes stored in the corresponding synopsis is very large. This will later be important in connection for the proof of Theorem 5.

5.3.2 The Error Model

The overall goal is to minimize the estimation error over all queries $Q_j \in \mathcal{Q}$. First consider a scenario in which all queries only depend on a single synopsis S_0 over the data distribution $\mathcal{T} = \{(v_1, f_1), (v_2, f_2), \ldots, (v_n, f_n)\}$. I define the error for a given query $Q_j \in \mathcal{Q}$ by characterizing how well the query result $Result(Q_j) \subseteq \mathcal{T}$ is approximated. Then I define the error over all queries in $\mathcal{Q}$ with respect to a data distribution $\mathcal{T}$ as

$$Error(\mathcal{Q}, S_0) := \sum_{Q_j \in \mathcal{Q}} \Big(\sum_{i \in \{k | v_k \in Result(Q_j)\}} (f_i - \hat{f}_i)^2 + r \cdot (\|v_i - \hat{v}_i\|)^2 \Big) \tag{5.1}$$

with the factor r determining the importance of frequency-approximation relative to the importance of density approximation. Thus, if the weights w_i are defined as $w_i := |\{Q' \mid v_i \in Result(Q'), Q' \in \mathcal{Q}\}|$, and the factor[1] v_domain / f_domain is used for r, the sum of the errors for each query posed to synopsis S_0 can be written as:

$$Error(\mathcal{Q}, S_0) := \sum_{i=1}^{|\mathcal{T}|} w_i \cdot (f_i - \hat{f}_i)^2 + \tfrac{v\_domain}{f\_domain} \cdot w_i \cdot (\|v_i - \hat{v}_i\|)^2, \tag{5.2}$$

[1] The choice of the value r does not influence the algorithmic properties of the approach, as any value for r can easily be incorporated into the reconciliation of frequency and density synopses described in Section 3.3.2.

with the appropriate values for *f_domain* and *d_domain*. Except for the weights w_i, this is the error function (Equation 4.3) *minimized* (in the sense described in Section 4.4) by spline synopses. Since the weights w_i can be easily incorporated into the spline construction process (see Section 3.3.5), minimizing the query error in the case of a single distribution has become a problem of constructing the optimal spline synopsis.

This is a slight simplification as it ignores approximation errors with regard to the boundary conditions of a query: when using a synopsis for answering a query some attribute values $\hat{v}_i$ may be included in the approximate answer even though the corresponding v_i would not be in the query result. Likewise, some attribute values may be erroneously excluded. However, incorporating these effects into our model would result in an optimization problem of intractable complexity.

In a scenario with multiple synopses $\mathcal{S} := \{S_1, \ldots, S_l\}$, each query Q_j is answered (depending on $Min(Q_j)$) by a synopsis in $\mathcal{S}$. To assign each queried attribute combination to exactly one synopsis, a mapping function

$$map : \bigcup 2^{\{Att(R)|R\in\mathcal{R}\}} \mapsto \{1, \ldots, l\} \tag{5.3}$$

is used. I will describe how to automatically obtain this mapping in Section 5.4.2.

Note that this model assumes that queries over the same attribute combination are always mapped to the same synopsis (otherwise it would be necessary to store additional information on the mapping of specific queries, which would in turn compete for the memory available for synopses). Thus, the error over a set of synopses $\mathcal{S} := \{S_1, \ldots, S_l\}$ is defined as:

$$Error(\mathcal{Q}, \mathcal{S}) = \sum_{i=1}^{l} (Error(\{Q_j \in \mathcal{Q} \mid map(Min(Q_j)) = i\}, S_i).$$

Since the error of each synopsis S_i is dependent on the memory size $Size(S_i)$ of the synopsis, this is more accurately stated as:

$$Error(\mathcal{Q}, \mathcal{S}) = \min_{(Size(S_1),\ldots,Size(S_l))\in\mathbb{N}^l} \sum_{i=1}^{l} (Error(\{Q_j \in \mathcal{Q} \mid map(Min(Q_j)) = i\}, S_i)) \tag{5.4}$$

under the constraint that $\sum_{i=1}^{l} Size(S_l)$ is equal to the memory size M available for all synopses together. Thus the problem of optimizing the estimation error for the entirety of queries in the workload can be seen as a problem of selecting the optimal set of synopses and choosing their sizes.

5.4 Synopsis Selection and Memory Allocation

To illustrate the issues involved in the selection of the optimal set of synopses, consider a workload $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$, $\mathcal{Q}_1$ containing no_1 queries Q' (i.e., queries of type Q' whose fraction in the entire workload is proportional to no_1) with $Min(Q') = \{\{R_1.A_1\}\}$ and $\mathcal{Q}_2$ containing no_2 queries Q'' with $Min(Q'') = \{\{R_1.A_2\}\}$. Then these can be answered by either

(a) two synopses $S_{\{R_1.A_1\}}$ and $S_{\{R_1.A_2\}}$ over each single attribute

(b) one synopsis $S_{\{R_1.A_1,R_1.A_2\}}$ over the joint data distribution of $R_1.A_1$ and $R_1.A_2$.

Therefore, to compute the optimal error for the overall available memory M we have to evaluate

$$Error(\mathcal{Q}) \quad := \quad \min \begin{cases} \overbrace{Error(\mathcal{Q}, S_{\{R_1.A_1,R_1.A_2\}})}^{\text{Error for combination (b)}}, & \text{(with } Size(S_{\{R_1.A_1,R_1.A_2\}}) = M\text{)} \\ \underbrace{\min_{\substack{Size(S_{\{R_1.A_1\}})\in\mathbb{N} \\ Size(S_{\{R_1.A_2\}})\in\mathbb{N}}} \big(Error(\mathcal{Q}_1, S_{\{R_1.A_1\}}) + Error(\mathcal{Q}_2, S_{\{R_1.A_2\}})\big)}_{\text{Error for combination (a)}} & \text{(with } Size(S_{\{R_1.A_1\}}) + Size(S_{\{R_1.A_2\}}) = M\text{)} \end{cases}$$

and keep track of the resulting synopses and memory partitioning. So the problem of computing the optimal set of synopses (and the corresponding memory allocation) becomes a two-step process:

(1) Computing $Error(\mathcal{Q}', S_x)$ for all candidate synopses S_x and all possible combinations of queries $\mathcal{Q}' \subseteq \mathcal{Q}$ that may be mapped to S_x and the maximum amount of Memory M to be used for S_x. This requires $O(M \cdot |\mathcal{T}_x|^2)$ steps (for the OPTIMAL partitioning, see Section 3.1.3) for each pair of S_x and $\mathcal{Q}'$ and also generates the values of $Error(\mathcal{Q}', S_x)$ for all values of $Size(S_x) \leq M$.

(2) Selecting the optimal set of synopses from a set of candidates computing an optimal memory partitioning such that the weighted sum over all synopsis errors (weighted by the number of times each synopsis is queried) becomes minimal for the synopses included in the optimal solution. Since the weights change according to the combinations of synopses in the solution, this is a different and more difficult problem than finding the optimal combination of synopses for different relations. It will be shown in Section 5.4.2, the problem of synopsis selection and memory partitioning are closely related and thus solved together.

In the following (Sections 5.4.1 and 5.4.2), I will show how to solve the above problem for a single dataset $R \in \mathcal{R}$. The sub-solutions for all datasets in $\mathcal{R}$ can then be combined to solve the overall problem (Section 5.4.3). Since the resulting solutions are computationally expensive, I will describe in Section 5.6 how to reduce the computational cost through the use of heuristics.

The key idea of my approach is to use two properties that hold for over the plethora of query (size) estimation techniques to prune the search space of all possible combinations of synopses and their sizes significantly. While both properties were found to hold for all combinations of queries and data sets tested in this thesis, only one of them can be shown to hold for every query/data combination possible. For the other property datasets can be constructed, for which it does not hold; therefore it should be seen as an efficient heuristic.

5.4.1 Pruning the Search Space

Note that in the example of Section 5.4 the combinations $\mathcal{S}' = \{S_{\{R_1.A_1\}}, S_{\{R_1.A_1,R_1.A_2\}}\}$ or $\mathcal{S}'' = \{S_{\{R_1.A_2\}}, S_{\{R_1.A_1,R_1.A_2\}}\}$ were never considered. This is due to a simple property of spline synopses, which also generally holds for both histograms and Wavelet-based approximations:

Observation 5.1 "Pruning Property" :

When answering queries over the set of attributes a, a synopsis S_x, over the set of attributes x with $a \subseteq x$ will yield more accurate answers than a synopsis S_y if $x \subset y$ and both synopses are of identical size.

While artificial data distributions can be constructed that do not obey the above observation, the pruning property was found to hold in all experiments on real-life datasets. The intuition behind it is the fact that by including more attributes in a synopsis, the number of unique attribute-value combinations v_i in the corresponding data distribution increases as well (in this respect, the synopsis selection problem is similar to the one of index selection), making it harder to capture all attribute values/frequencies with acceptable accuracy.

In the above case, it means that $S_{\{R_1.A_1\}}$ answers queries posed to $R_1.A_1$ better than $S_{\{R_1.A_1,R_1.A_2\}}$ (using the same memory). Similarly $S_{\{R_1.A_2\}}$ is an improvement over $S_{\{R_1.A_1,R_1.A_2\}}$ for queries posed to $R_1.A_2$. Thus the combination $\mathcal{S} = \{S_{\{R_1.A_1\}}, S_{\{R_1.A_2\}}\}$ generally outperforms $\mathcal{S}'$ or $\mathcal{S}''$.

Using the above observation, it becomes possible to characterize the set of candidate synopses in a compact manner. Consider a single relation R. Then the sets of attribute combinations of R queried is

$$Syn(R, \mathcal{Q}) := \{Min(Q_i) \mid Q_i \in \mathcal{Q}\}$$

Now the set of all candidate synopses for R can be defined as:

$$Cand(R, \mathcal{Q}) := \{S_x \mid x = \bigcup z, z \subseteq Syn(R, \mathcal{Q})\}$$

This means that the set of all candidate synopses forms a lattice (see Figure 5.1) over the attribute-combinations queried; e.g. if the set of all queried attribute combinations is

$$Syn(R, \mathcal{Q}) = \{\{R.A_1\}, \{R.A_2\}, \{R.A_2, R.A_3\}, \{R.A_3\}\}$$

then the set of candidate synopses becomes

$$Cand(R, \mathcal{Q}) = \{\{R.A_1\}, \{R.A_2\}, \{R.A_3\}, \{R.A_1, R.A_2\}, \{R.A_1, R.A_3\}, \{R.A_2, R.A_3\}, \{R.A_1, R.A_2, R.A_3\}\}.$$

The intuition behind the definition of $Cand(R, \mathcal{Q})$ is the following: if a Synopsis S_y is in

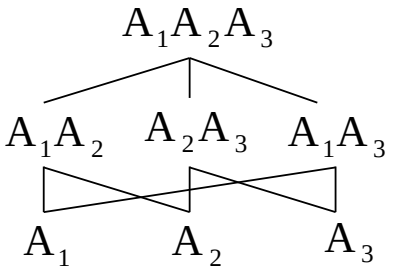

Figure 5.1: The lattice of candidate synopses

$Cand(R, \mathcal{Q})$, it must be considered, for it is the most efficient way to answer a subset of queries of R using only one synopsis (all other synopses capable of answering the same subset would be less efficient, due to the pruning property). Conversely, if $S_y \notin Cand(R, \mathcal{Q})$ then y must be of the form $y = cand \cup nocand$ with $cand \in \{\bigcup z, z \subseteq Syn(R, \mathcal{Q})\}, nocand \subseteq Att(R)$, and $\forall x \in nocand : (cand \cup x) \notin \{\bigcup z, z \subseteq Syn(R, \mathcal{Q})\}$. But then S_{cand} answers the same set of queries as S_y and does so more efficiently, since $cand \subset y$.

In addition to the pruning property, I utilize a second property of spline synopses to prune the search space even further.

Observation 5.2 "Merge Property" :

For a set of queries $\mathcal{Q}$ each querying the same combination of attributes A, the error for answering the queries using one synopsis S over A with M memory is smaller than the error using two synopses S_1, S_2 over A, which together use memory M.

The intuition for this property is the following: By joining the synopses S_1 and S_2, the estimation for the (potentially) overlapping regions in S_1 and S_2 is improved, as additional memory is invested in its estimation. It is a trivial consequence that the merge property also holds for combinations of more than two synopses over A. In contrast to the pruning property, it is possible to prove that the merge property always holds.

Theorem 5. *For a set of queries $\mathcal{Q}$ over the same attribute combination A, the error for answering these queries by a single synopsis S_{union} is less to or equal to the error for $\mathcal{Q}$ with two synopses S_1, S_2, which together require the same amount of memory as S_{union}.*

Proof: In Section 5.3.2 it was shown that the error for a set of queries corresponds to the error of the fitting-problem posed in Equation 5.2. Therefore, if I can show that the value for Equation 5.2 is smaller for S_{union} than for S_1 and S_2, the overall statement follows.

This will be done in the following manner: Based on a given S_1, S_2 and $\mathcal{Q}$, I will construct a corresponding S_{union} in terms of the bucket-partitioning used for fitting attribute frequencies and values. I will show that the overall error is less or equal for S_{union}; now, since our partitioning method for constructing spline synopses traverses the space of all possible partitionings, either S_{union} or a synopsis with better overall error will result from running the spline synopses construction on $\mathcal{Q}$.

First, it is necessary to consider the storage requirements for spline synopses: buckets for both frequency- and attribute-value approximation store 3 values (see Section 3.3.1); in addition, a total of 6 values storage for bookkeeping information is required (see Section 5.3.1). Therefore, each synopsis requires $6 + 3 \cdot$ (number of buckets) of storage. Because no further further bookkeeping information is stored, it is necessary to require that each synopsis approximates all values between the lowest and highest value stored[2]. Otherwise it would not be possible to decide whether at run-time whether a query can be approximated by a synopsis or not, for the incoming queries may not be part of the workload the synopses were constructed over.

I also need to introduce the following notation: Define F_Err^i as the respective error for the problem of fitting the frequencies (i.e. $\sum_{i=0}^{\mathcal{T}} w_i \cdot (f_i - \hat{f}_i)^2$) in Synopsis S_i. Define the constraint error $F\_Err^i_{[low,high)}$ as $\sum_{i=low}^{high-1} w_i \cdot (f_i - \hat{f}_i)^2$ for S_i. Define D_Err^i and $D\_Err^i_{[low,high)}$ analogously, only with regard to fitting of the attribute-values.

Define $I_i = [v_{start_i}, v_{end_i})$ as this interval of the values approximated by S_i. If I_1 and I_2 overlap, it has to be decided to which synopses to map the (parts of) queries in $\mathcal{Q}$ querying $I_1 \cap I_2$. As it is not possible to map every single query to a synopsis individually without introducing additional information, each query (part) querying the interval $I_1 \cap I_2$ is mapped to the synopsis with the lower overall error in the area of intersection.

Without loss of generality I assume that $v_{start_1} \leq v_{start_2}$. For purposes of approximation, each interval I_j is partitioned into buckets; I define a partitioning into k buckets as a $k-tuple$ $P = (low_1^j, \ldots, low_k^j)$ corresponding to the buckets $[v_{low_1^j}, v_{low_2^j}), \ldots, [v_{low_k^j}, v_{end_j})$.

Now, given synopses S_1 using b_1 buckets (of which b_1^f are used for frequency-approximation) and S_2, using b_2 buckets (b_2^f defined analogously), I construct S_{union} with at most $b_1 + b_2 + 2$

[2]For synopses over more than one attribute, it suffices to require that all values between the highest and lowest value in I_1 after the mapping by the space-filling curve are stored.

buckets, requiring the same amount of memory (as the bookkeeping overhead is only incurred once, not twice, which allows to store the two additional buckets[3]).
Regarding the construction, I have to differentiate 3 scenarios:

$\mathbf{I_1 \cap I_2 = \emptyset}$: In this case, I can construct S_{union} on the union of the buckets in S_1 and S_2, as there is no overlap between them. In order to fulfill the requirement of synopses approximating all values in the interval $[v_{start_1}, v_{end_2})$, I need to add two additional buckets (one for frequency, the other for attribute-value approximation) covering $[v_{end_1}, v_{start_2})$. Since these aren't queried, however, this doesn't influence the resulting error. Therefore $Error(\mathcal{Q}, S_{union}) = F\_Err^1 + F\_Err^2 + D\_Err^1 + D\_Err^2 = Error(Q_1, S_1) + Error(Q_2, S_2)$.

$\mathbf{I_1 \cap I_2 \neq \emptyset, I_1 \cap I_2 \neq I_1, I_1 \cap I_2 \neq I_2}$: In this case, I first construct the frequency approximation of S_{union} using at most $b_1^f + b_2^f$ buckets. Here I have to differentiate 2 cases:

(a) $F\_Err^1_{I_1 \cap I_2} \leq F\_Err^2_{I_1 \cap I_2}$ (*): Denote the partition for the frequency-approximation in S_1 by $P_1^f = (low_1^1, \ldots, low^1_{b_1^f})$, for S_2 $P_1^f = (low_1^2, \ldots, low^2_{b_2^f})$. Let k be the smallest index such that $v_{low_k^2} > v_{end_1}$. If $b_2^f > 1$, such a k must exist and we construct the partitioning for frequency-approximation in S_{union} as $P = (low_1^1, \ldots, low^1_{b_1^f}, end_1, low_k^2, \ldots, low^2_{b_2^f})$. Otherwise (if $b_2^f = 1$), the partitioning constructed is $P = (low_1^1, \ldots, low^1_{b_1^f}, end_1)$.

In both cases, the corresponding frequency-approximation is S_{union} has at most $b_1^f + b_2^f$ buckets. The corresponding error for fitting the frequency-values is $F\_Err^{union} = F\_Err^1 + F\_Err^1_{I_1 \cap I_2} + F\_Err^2_{[end_1, end_2)} \leq F\_Err^1 + F\_Err^2_{I_1 \cap I_2} + F\_Err^2_{[end_1, end_2)} = F\_Err^1 + F\_Err^2$ (because of (*)).

(b) $F\_Err^1_{I_1 \cap I_2} > F\_Err^2_{I_1 \cap I_2}$: In this case, the proof is analogous, with k defined as the largest index, such that $v_{low_k^1} \leq v_{start_2}$ and $P = (low_1^1, \ldots, low_k^1, low_1^2, \ldots, low^2_{b_2^f})$ (if $b_1^f = 1$ then $P = (low_1^1, low_1^2, \ldots, low^2_{b_2^f})$). In this case $F\_Err^{union} = F\_Err^2 + F\_Err^2_{I_1 \cap I_2} + F\_Err^1_{[start_1, start_2)} \leq F\_Err^2 + F\_Err^1_{I_1 \cap I_2} + F\_Err^2_{[start_1, start_2)} = F\_Err^1 + F\_Err^2$.

The attribute-value approximation of S_{union} can now be constructed similarly, yielding a synopses with $b_1 + b_2$ buckets, with $Error(\mathcal{Q}, S_{union}) \leq F\_Err^1 + F\_Err^2 + D\_Err^1 + D\_Err^2 = Error(Q_1, S_1) + Error(Q_2, S_2)$.

$\mathbf{I_1 \cap I_2 = I_2}$: Again I have to separate two cases:

(a) $F\_Err^1_{[start_1, end_1)} + D\_Err^1_{[start_1, end_1)} \leq F\_Err^2_{[start_1, end_1)} + D\_Err^2_{[start_1, end_1)}$: in this case we simply include all buckets in S_1 into S_{union}. Then $Error(\mathcal{Q}, \mathcal{S}) = F\_Err^1 + D\_Err^1 + F\_Err^1_{[start_1, end_1)} + D\_Err^1_{[start_1, end_1)} \leq F\_Err^1 + D\_Err^1 + F\_Err^2_{[start_2, end_2)} + D\_Err^2_{[start_2, end_2)} = Error(Q_1, S_1) + Error(Q_2, S_2)$.

(b) Otherwise I again have to construct a new partitioning for both frequency and density-approximation such that $F\_Err^{union} \leq F\_Err^1 + F\_Err^2$ and $D\_Err^{union} \leq D\_Err^1 + D\_Err^2$, using at most $b_1 + b_2 + 2$ buckets. For the frequency-approximation in S_{union} this is done in the following way: denote the partition for the frequency-approximation in S_1 by $P_1^f = (low_1^1, \ldots, low^1_{b_1^f})$, for S_2 by $P_1^f = (low_1^2, \ldots, low^2_{b_2^f})$. Now define k as the number of buckets in S_1 such that $v_{low_i} < v_{start_2}$, define k' as the number of buckets

[3]Note this still holds for synopses over a large number d of attributes, that use a larger data type for the lowest value in a bucket, as in this case the bookkeeping for a synopsis stores 2 values of this type itself.

in S_1 such that $v_{low_i} \geq v_{end_2}$. Then I construct $P = (low_1^1, \ldots, low_k^1, low_1^2, \ldots, low_{b_2^f}^2, end_2, low_{k'}^1, \ldots, low_{b_1^f}^1)$. This results in $k+1+b_2^f+(b_1^f-k'+1)$ buckets, which is at most $b_1^f+b_2^f+1$, since $k'-k \geq 1$. The resulting error for approximation of value frequencies is $F^{union} = F\_Err^1_{[start_1,start_2)} + F\_Err^2$. The construction for the attribute-value approximation is similar. □

Because both of these properties typically apply also to approximation by histograms or wavelets, the following results also apply to the physical design problem when these techniques are used as base synopses[4]. It is necessary to use a different error measure, however, as it is not possible to match pairs of values and frequencies in the original data and the approximation, because the size of a data distribution $|\mathcal{T}_x|$ is not invariant when approximated by either of these techniques. Instead, I suggest the use of an error measure that tracks the difference between the actual and the approximated data distribution, and computes its own matching, such as the *MAC* distance [IP99] or the error-metric introduced for join-errors in Section 3.4.4.

5.4.2 Selecting the Synopses for a Single Relation *R*

In the following I will describe, for a given set $\mathcal{Q}$ of queries over a single relation R, how to compute the optimal combination $\mathcal{S}$ of synopses, their sizes, and the corresponding mapping of queries, such that all queries can be answered and the overall error becomes minimal.

The problem resembles the issue of selecting the optimal set of (multicolumn) indices for a given workload, as the trade-off is similar: by selecting a synopsis or an index over a larger number of attributes, the number of queries this synopsis or index supports is increased, while at the same time its cost also rises. However, the synopsis selection problem is unique, and substantially more difficult, for a number of reasons:

(1) *The size of the synopsis for a given attribute combination is not fixed.* Therefore, a synopsis that provides poor estimations at size m might be quite useful at size $m', m' > m$. So not only all potential combinations of synopses have to evaluated, but also all possible combinations of their sizes.

(2) *A standard B-tree index over attributes $A_1, \ldots, A_h$ can be used to support queries over any prefix $A_1, \ldots, A_{h'}, h' \leq h$;* thus the presence of such an index makes indexes over prefixes superfluous. In contrast, a synopsis over the set of attributes $\mathcal{A} = \{A_1, \ldots, A_h\}$ can answer queries over all subsets of $\mathcal{A}$, but because the approximation quality deteriorates with the inclusion of additional attributes (see Section 5.4.1), additional synopses over subsets of $\mathcal{A}$ may still be beneficial.

(3) *The maintenance costs of data synopses are only a second-order issue*, as synopses do not have to be constantly kept up-to-date, but can be re-computed once their accuracy deteriorates too far.

Therefore, exact or heuristic optimization methods for index selection, such as the formulation as a 0-1 integer linear program [CFM95] or *index merging* [CN99], cannot be applied in this context.

As shown before, the optimal combination of synopses $\mathcal{S}_{opt}$ can consist of synopses over single attribute combinations from $Syn(R, \mathcal{Q})$ that are optimal for a particular query in $\mathcal{Q}$, as well as synopses for the joint attribute combinations of multiple members of $Syn(R, \mathcal{Q})$, which are not

[4] Because both techniques generally only optimize the approximation of the attribute value frequencies, this holds only with regard to the corresponding frequency error $\sum_{i=0}^{|\mathcal{T}|}((f_i - \hat{f}_i)^2$.

optimal for any single query but more efficient than other combinations of synopses (using the same amount of memory) capable of answering the same queries. Now I want to capture this notion algorithmically, describing a method to construct a set of synopses for a given workload/data combination. I first introduce the necessary notation:

$Opt\_Syn_{\mathcal{A},M}$ is defined as the combination of synopses for answering all queries over the attribute combinations in $\mathcal{A} \subseteq Syn(R, \mathcal{Q})$ using memory M as constructed below.

$Opt\_Err_{A,M}$ describes the overall error resulting from $Opt\_Syn_{\mathcal{A},M}$.

Now consider the problem of computing the optimal combination of synopses $Opt\_Syn_{\mathcal{A},M}$ for given $\mathcal{A}$ and M. $Opt\_Syn_{\mathcal{A},M}$ has one of the following forms:

(a) $\mathbf{Opt\_Syn}_{\mathcal{A},\mathbf{M}} = \{\mathbf{S}_{\bigcup \mathcal{A}}\}$ with $Size(S_{\bigcup \mathcal{A}}) = M$ (one synopsis for all queries over the attribute combinations in $\mathcal{A}$).

(b) $\mathbf{Opt\_Syn}_{\mathcal{A},\mathbf{M}} = \mathbf{Opt\_Syn}_{\mathcal{A}',\mathbf{m}'} \cup \mathbf{Opt\_Syn}_{\mathcal{A}-\mathcal{A}',\mathbf{M}-\mathbf{m}'}$
(a combination of the optimal synopses for answering two disjoint subsets of $\mathcal{A}$ with $\mathcal{A}' \neq \emptyset$). Because of the merge property, only decompositions for which $Opt\_Syn_{\mathcal{A}',m'} \cap Opt\_Syn_{\mathcal{A}-\mathcal{A}',M-m'} = \emptyset$ have to be considered.

Which combination is optimal depends on the error resulting from each alternative:

In case (a) $Opt\_Err_{\mathcal{A},M} = Error(\underbrace{\{Q' \mid Min(Q') = \bigcup \mathcal{A}\}}_{\text{The set of queries answered by } S_{\bigcup \mathcal{A}}}, \{S_{\bigcup \mathcal{A}}\})$ with $Size(S_{\bigcup \mathcal{A}}) = M$.

In case (b) $Opt\_Err_{\mathcal{A},M} = \min_{m' \in \{1,\ldots,M-1\}} Opt\_Err_{\mathcal{A}',m'} + Opt\_Err_{\mathcal{A}-\mathcal{A}',M-m'}$

Therefore, it is possible to compute the optimal set of synopses for $\mathcal{A}$ by computing the minimal error for cases (a) and (b) and choosing the memory partitioning that minimizes the corresponding error. Note that by computing the optimal combination of synopses in the above manner, a mapping that dictates which attribute combinations from $Syn(R, \mathcal{Q})$ are mapped to which synopses is also computed implicitly: because of the above decomposition, $\mathcal{S} := Opt\_Err_{\bigcup Syn(R,\mathcal{Q}),M}$ is of the form $\mathcal{S} = \{S_{\bigcup \mathcal{A}_1}, \ldots, S_{\bigcup \mathcal{A}_l}\}$ with each $a \in Syn(R, \mathcal{Q})$ being a member of exactly one $\mathcal{A}_1, \ldots, \mathcal{A}_l$. While more complex models are possible, in which queries over the same attribute combination are mapped to different members of $\mathcal{S}$, this would mean that additional information, from which the correct mapping for each single query could be derived at run-time, would have to be stored (creating contention for memory with the actual data synopses).

Using the above definitions, the final set of synopses kept for R using memory M is $Opt\_Syn_{\bigcup Syn(R,\mathcal{Q}),M}$, the corresponding error being $Opt\_Err_{\bigcup Syn(R,\mathcal{Q}),M}$. However, it is still necessary to prove that the optimal solution can indeed be obtained based on the decompositions described above:

Theorem 6. *The set $Opt\_Syn_{\mathcal{A},M}$ constructed in the above manner is the optimal combination of synopses for answering all queries in $\mathcal{Q}$ over the attribute combinations in $\mathcal{A}$, when the pruning and merge properties hold.*

Proof: I show that $\mathcal{S} := Opt\_Syn_{\mathcal{A},M}$ using the above construction implies that $\mathcal{S}$ is the optimal combination of synopses answering all queries over the attribute combinations in $\mathcal{A}$ using memory M. This is proven by induction over $|\mathcal{A}|$:

$|\mathcal{A}| = 1$: Then $Opt\_Syn_{\mathcal{A},M} = \{S_{\mathcal{A}}\}$ (no partitioning involving multiple synopses possible because of the merge property), and because of the pruning property $S_{\mathcal{A}}$ is the best way to answer queries over $\mathcal{A}$.

$|\mathcal{A}| \rightarrow |\mathcal{A}| + 1$: Now I assume that all $Opt\_Syn_{\mathcal{A},M}$ for $|\mathcal{A}| \leq h$ are indeed optimal and try to show the optimality for $Opt\_Syn_{\mathcal{A},M}$ with $|\mathcal{A}| = \text{h+1}$. This is shown by contradiction:

Assumption: There exists a solution $\mathcal{S}_{opt} = \{S_{x_1}, \ldots, S_{x_t}\}$ (with $Size(S_{x_i}) = m_i, i = 1, \ldots, t$) such that the resulting overall error Err_{opt} over all queries is indeed smaller than $Opt\_Err_{\mathcal{A},M}$ with $\mathcal{S}_{opt} \neq Opt\_Syn_{\mathcal{A},M}$.

- **(Case 1) $|\mathcal{S}_{\mathbf{opt}}| = 1$:** Then $\mathcal{S}_{opt} = \{S_{\mathcal{A}'}\}$, with $Size(S_{\mathcal{A}'}) = M$. Since $\mathcal{S}_{opt}$ has a smaller Error than $Opt\_Err_{\mathcal{A},M}$, $\mathcal{S}_{opt} \neq \{S_{\mathcal{A}}$ (as S_A is a possible synopsis combination for $Opt\_Syn_{\mathcal{A},M}$ and thus $Error(\mathcal{Q}, \{S_{\mathcal{A}}) \geq Opt\_Err_{\mathcal{A},M} > Err_{opt}$). However, since $\mathcal{S}_{opt}$ must be able to answer all queries over $\mathcal{A}$, $\mathcal{A} \subset \mathcal{A}'$ holds. Then it follows from the pruning property that $Opt\_Syn_{\mathcal{A},M}$ results in better accuracy than $\mathcal{S}_{opt}$, contradicting the previous assumption.
- **(Case 2) $|\mathcal{S}_{\mathbf{opt}}| > 1$:** Because of the merge property, I assume that all queries to the same attribute combination $a \in \mathcal{A}$ are mapped to the same synopsis. Should this not be the case, it is possible replace $\mathcal{S}_{opt}$ by $\mathcal{S}'_{opt}$, for which all synopses over the same attribute have been merged, resulting in a smaller error. If it is now possible to contradict the assumption for $\mathcal{S}'_{opt}$, it is thereby contradicted for $\mathcal{S}_{opt}$, too.
  $\mathcal{S}_{opt}$ can now be written as $\mathcal{S}_{opt} = \mathcal{S}_1 \cup \mathcal{S}_2, \mathcal{S}_1 \neq \emptyset, \mathcal{S}_1 \neq \mathcal{S}, \mathcal{S}_2 := \mathcal{S} - \mathcal{S}_1$ with $\mathcal{S}_1 = \{S_{x_1^1}, \ldots, S_{x_p^1}\}, \mathcal{S}_2 = \{S_{x_1^2}, \ldots, S_{x_q^2}\}, p \leq h, q \leq h$. Because all queries over the same attribute combination are mapped to the same synopsis, both $\mathcal{S}_1$ and $\mathcal{S}_2$ each answer queries over the attribute combinations in disjoint subsets $\mathcal{A}_1, \mathcal{A}_2$ of $\mathcal{A}$ with $\mathcal{A}_1 \cup \mathcal{A}_2 = \mathcal{A}$. Then it follows from the induction hypothesis that $Opt\_Syn_{\mathcal{A}_1, \sum_{i=0}^{p} Size(S_{x_i^1})}$ results in a smaller error than $\mathcal{S}_1$ for queries over attribute combinations in $\mathcal{A}_1$, and $Opt\_Syn_{\mathcal{A}_1, \sum_{i=0}^{q} Size(S_{x_i^1})}$ results in a smaller error than $\mathcal{S}_2$ for queries over attribute combinations in $\mathcal{A}_2$. It follows that the error for $Opt_{\mathcal{A},M}$ is less than the one caused by $\mathcal{S}_{opt}$, contradicting the assumption.

5.4.3 Selecting the Synopses for all Relations

The error over all relations for total memory size M can now be written as

$$Error(\mathcal{Q}, \mathcal{S}) = \min_{(M_1, \ldots, M_{|\mathcal{R}|}) \in \mathbb{N}^{|\mathcal{R}|}} \sum_{i=1}^{|\mathcal{R}|} Opt\_Err_{\bigcup Syn(R_i, \mathcal{Q}), M_i} \tag{5.5}$$

under the constraint that $\sum_{i=1}^{|\mathcal{R}|} M_i = M$. This is equivalent to the initial definition in Equation 5.4. Expression 5.5 can be solved by dynamic programming using $O(M^2 \cdot |\mathcal{R}|)$ operations. By keeping track of the memory partitioning $(M_1, \ldots, M_{|\mathcal{R}|})$, it is then possible to determine the optimal set of synopses $\mathcal{S} := \bigcup_{i=1,\ldots,|\mathcal{R}|} Opt\_Syn_{\bigcup Syn(R_i, \mathcal{Q}), M_i}$.

5.5 Enumeration of all synopses combinations

To enumerate all possible combinations of synopses and mappings for a given relation R and a set of queries $\mathcal{Q}$ over R, the following algorithm (Algorithm 5) is used. Each single combination is encoded the following way:

Definition 5.2 (Combination of synopses and *map*) :
A single *combination* of a candidate set of synopses, a corresponding mapping-function map and a set of (yet) unmapped queries is described as a 3-tuple $(synopses, map, free\_queries)$, with *synopses* being the set containing the attribute combinations over which synopses are to be constructed. *map* corresponds to the mapping function defined in Section 5.3.2; i.e. for every queried attribute combination $a \in Syn(R, \mathcal{Q})$ *map* contains the pair (a, a') meaning queries on a are mapped to the synopsis $S_{a'}$. Attribute combinations not yet mapped are stored in the set $free\_queries$. The 3 components of a combination x are referred to as $x.synopses, x.map$ and $x.free\_queries$, respectively.

In order to compute all synopses combinations, the algorithm iterates over all members of the set of candidate synopses $Cand(R, \mathcal{Q})$ and for each member S_x computes all possible synopsis combinations capable of answering all queries in $\mathcal{Q}$ over the attributes contained in x. As described in Section 5.4.1, $Cand(R, \mathcal{Q})$ forms a lattice, which is traversed bottom-up; i.e. if $y \subset x$, then S_y is traversed before S_x. To this end, the set of candidate synopses is enumerated as $Cand(R, \mathcal{Q}) = S_{c_1}, \ldots, S_{c_{|Cand(R,\mathcal{Q})|}}$ (with c_i denoting attribute combinations) in such a way that $\forall l = 1 \ldots |Cand(R, \mathcal{Q})| - 1 : |c_i| \leq |c_{i+1}|$. The outer loop of the algorithm then traverses the synopses in this order. In the inner loop, the algorithm computes all combinations for synopses over a set of attributes c_i, which are stored as $Candidates(c_i)$. After iterating over all candidates, all possible synopses combinations for all queries on R are stored in $Candidates(c^{top})$ with c^{top} being the *top* element of the lattice (i.e. $\exists S_{c_i} \in Cand(R, \mathcal{Q}) : \forall S_{c'} \in Cand(R, \mathcal{Q}) : c' \subset c_i$).

If the pruning of candidate synopses described in Section 5.6 is employed, the lattice structure may be destroyed, for some of its elements are missing. In this case, the candidates are still traversed in the same way, however, it is necessary to execute lines 6-33 one more time, substituting the set of all elements that are the top-elements of a limited subset of Candidates (i.e. $\{c_i \in Cand(R, \mathcal{Q} : \nexists c' \in Cand(R, \mathcal{Q}) : c_i \subset c'\}$) for $Predecessors(c_1)$ in line 6 and 8.

5.6 Reducing the Computational Overhead

So computing the optimal set of synopses $\mathcal{S}$ for a given query $\mathcal{Q}$ can be broken down into three steps:

(1) Computing the error for each candidate synopsis and each combination of queries.

(2) Computing the optimal set of synopses for each relation $R_i \in \mathcal{R}$ and all possible values M_i for the allocated memory.

(3) Solving Equation 5.5 to determine how much memory to dedicate to each relation's synopses.

Only step 3 is of low computational overhead, while steps 1 and 2 are expensive even for small instances of $Syn(R, \mathcal{Q})$. This means that while the described method scales up well with a rising number of relations, it still becomes necessary to ensure the tractability of the computation of $Opt\_Syn_{\bigcup Syn(R,\mathcal{Q}),M}$ for relations in which many different attribute combinations are being queried (recall that the potential application domains include data mining tasks on datasets that may have dozens of attributes). I address potential bottlenecks for both step 1 and 2 for a single relation R in the following sections.

```
1: for i = 0, ..., |Cand(R, Q)| do
2:   if c_i ∈ Syn(R, Q) then
3:     Temp_Candidates.insert({S_{c_i}}, {(c_i, c_i)}, ∅)
4:     Temp_Candidates.insert(∅, ∅, {c_i})
5:   end if
6:   Temp_Candidates.insert(∅, ∅, ∅)
7:   for all p' ∈ Predecessors(c_1) do
8:     for all x ∈ Temp_Candidates do
9:       for all y ∈ Candidates(p') do
10:        if (x.synopses ∩ y.synopses = ∅ ∧ x.free_queries ∩ y.free_queries = ∅) ∧ ∄c_j, c_k, c_q ∈
           Cand(R_i, Q : ((c_j, c_k) ∈ x.map ∧ (c_j, c_q) ∈ x.map ∧ c_k ≠ c_q) then
11:          if c_i ∈ Syn(R, Q) then
12:            Temp_Cand_2.insert(x.synopses ∪ y.synopses ∪ {S_{c_i}}, x.map ∪ y.map ∪ {(c_k, c_i) |
               c_k ∈ (x.free_queries ∪ y.free_queries)} ∪ (c_i, c_i), ∅)
13:            Temp_Cand_2.insert(x.synopses ∪ y.synopses, x.map ∪ y.map,
               x.free_synopses ∪ y.free_synopses ∪ {c_i})
14:          else
15:            Temp_Cand_2.insert(x.synopses ∪ y.synopses ∪ {S_{c_i}}, x.map ∪ y.map ∪ {(c_k, c_i) |
               c_k ∈ (x.free_queries ∪ y.free_queries)}, ∅)
16:            Temp_Cand_2.insert(x.synopses ∪ y.synopses, x.map ∪ y.map,
               x.free_synopses ∪ y.free_synopses)
17:          end if
18:        end if
19:      end for
20:    end for
21:    Temp_Candidates = Temp_Cand_2
22:    Temp_Cand_2.clear()
23:  end for
24:  for all cand ∈ Temp_Candidates do
25:    if |cand.map| ≥ |{a | a ∈ Cand(R, Q) ∧ a ⊆ c_i ∧ ∄a' ∈ Cand(R, Q) : a' ⊈ c_i ∧ a ⊆ a'}|
       then
26:      Temp_Candidates(c_i).delete(cand) // Not all subsumed queries mapped
27:    end if
28:    Candidates(c_i) = Temp_Candidates
29:    Temp_Candidates.clear()
30:  end for
31: end for
```

Algorithm 5: Enumeration of all synopsis combinations

Computation of the *Error* Values

Before computing $Opt\_Syn_{\bigcup Syn(R,\mathcal{Q}),M}$ for a given relation R, it is necessary to compute $Error(\mathcal{Q}', S_x)$ for $Size(S_x)$ ranging from 1 to M, for all $S_x \in Cand(R, \mathcal{Q})$ and all combinations of queries $\mathcal{Q}' \subseteq \mathcal{Q}$ that can be answered by S_x (i.e., all combinations of queries corresponding to disjoint attribute combinations $\subseteq x$). Computing each $Error(\mathcal{Q}', S_x)$ means constructing the corresponding synopsis with memory M and requires $O(|\mathcal{T}_x|^2 \cdot M + M^2)$ operations (because the construction process computes $Error(\mathcal{Q}', S_x)$ for all $Size(S_x) < M$ at no additional cost, its

necessary to construct the synopsis only once). It follows that this potential bottleneck can be addressed by either reducing the number of combinations of x and $\mathcal{Q}'$ considered or by reducing the complexity of computing each $Error(\mathcal{Q}', S_x)$, i.e., of constructing the corresponding synopsis.

Reducing the number of $x, \mathcal{Q}'$ combinations: It is possible to significantly reduce the number of combinations under consideration by assuming that all future queries are equally likely to access a given (multidimensional) attribute-value region in the synopsis. This would mean that instead of minimizing the weighted error in Equation 5.2, all weights would be uniformly set to $w_i := 1$. So the workload information in $\mathcal{Q}$ is then used to obtain how often certain attribute combinations are queried, however, no information on query locality is used with regard to attribute values. This makes it possible to maintain large traces of previously executed queries without using significant memory, for now only the number of queries to each synopsis have to be tracked.

It is important, that this simplification is only made when computing the *Error* values used for determining the combination and size of the synopses stored. Locality information (if collected) can still be used when computing the final synopses themselves. Under this assumption, $Error(\mathcal{Q}', S_x)$ has to be computed only once for each x. $\mathcal{Q}'$ then corresponds to the set of all possible queries on S_x. Thus the number of different synopses to compute would be given by $|Cand(R, \mathcal{Q})|$ (I will discuss methods for reducing $|Cand(R, \mathcal{Q})|$ itself in Section 5.6). Also $Error(\mathcal{Q}', S_x)$ with $Size(S_x) = M$ can now be characterized as

$$\begin{aligned} Error(\mathcal{Q}', S_x) &= |\mathcal{Q}'| \cdot \text{ error for approximating } \mathcal{T}_x \text{ in } M \text{ memory.} \\ &= |\mathcal{Q}'| \cdot \sum_{i=1}^{|\mathcal{T}|} (f_i - \hat{f}_i)^2 + r \cdot (\|v_i - \hat{v}_i\|)^2 \end{aligned}$$

This approach also allows the *Error* values to be computed lazily (i.e., whenever the underlying database system, data-mining or mediation platform has free resources) and stored until the corresponding datasets change significantly. I refer to this heuristic as Uniform-Locality.

Reducing the overhead of synopsis construction: The overhead caused by this part of the algorithm can be reduced significantly, by (initially) not using the *OPTIMAL* algorithm when generating synopses, but using the faster and less accurate *GREEDY-MERGE* and *GREEDY-SPLIT* heuristics. The important, empirically observed, feature here is that the loss in accuracy is relatively uniform for all candidate synopses, resulting in *Error* values that lead to a synopsis set S' very close to the optimal set S. A similar approach is possible using histograms; an approximate algorithm for partitioning *V-Optimal* histograms can be found in [JKM+98].

After $\mathcal{S}$ has been selected, the final synopses in $\mathcal{S}$ can be (re-)computed using the optimal algorithm; the resulting overall running time is significantly shorter than before, since generally $|\bigcup_{i=1,\ldots,n} Cand(R_i, \mathcal{Q})| \gg |\mathcal{S}|$. We refer to this heuristic as Greedy-Construction.

Reducing $|Cand(R, \mathcal{Q})|$

With the modifications of the previous subsection, the overhead of both step 1 & 2 in the construction of the (near-)optimal synopses is determined by the size of the set of candidate synopses $Cand(R, \mathcal{Q})$:

In step 1 we have to construct $|Cand(R, \mathcal{Q})|$ synopses to obtain the resulting *Error* values, and

in step 2, reducing the cardinality of $Cand(R, \mathcal{Q})$ is crucial to limit the number of different combinations of attributes $\mathcal{A}', \mathcal{A} - \mathcal{A}'$ examined in sub-case (b) of the construction of *Opt_Syn*.

Therefore, I consider merely a small number of "promising" attribute combinations for synopses in $Cand(R, \mathcal{Q})$. The key to finding such promising attribute combinations is to be able to "guess" the resulting *Error* values with reasonable accuracy without having to actually compute them. I found during our experiments with multidimensional spline synopses, that $|\mathcal{T}_x|$ is a good indicator regarding the size of the resulting approximation error, i.e., if $|\mathcal{T}_x| < |\mathcal{T}_y|$ then typically $Error(\mathcal{Q}', S_x) < Error(\mathcal{Q}', S_y)$ when $Size(S_x) = Size(S_y)$. Intuitively, utilizing this rule means assuming that all data distributions are "equally difficult" to approximate and thus the resulting approximation error depends on the size of the data distribution.

Based on these considerations, the set of "promising" synopses $\widehat{Cand}(R, \mathcal{Q})$ is chosen in the following way. Initially, for each synopsis $S_a \in Syn(R, \mathcal{Q})$, S_a is added to $\widehat{Cand}(R, \mathcal{Q})$, as it is the best way to answer queries over the attribute combination a only. For the remaining synopses $S_a \in Cand(R, \mathcal{Q})$ then a rating of each synopsis $val(x) := |\mathcal{T}_x|/|\mathcal{Q}'|$ is computed, with $\mathcal{Q}'$ being the number of queries in $\mathcal{Q}$ for which S_a can be used. Then, only a small number of the remaining synopses S_y corresponding to the lowest values for $val(y)$ is included in $\widehat{Cand}(R, \mathcal{Q})$. In the following, I will refer to this heuristic as Small-Cand.

5.7 Putting the Pieces Together

Solving the *physical design problem* for data synopses can be characterized as a 4-step process:

(a) **Collection of workload information:** Initially, it is necessary to acquire the necessary information about the access behavior of the workload, which can be done automatically by the data manager that processes the queries. The details of this information depend on the heuristics employed for solving the physical design problem: when assuming that all data points of a table are equally likely to be accessed (Small-Cand, see Section 5.6), it is only necessary to collect the access frequency of each attribute combination present in the workload. Otherwise, it is also necessary to store the access frequencies for attribute value combinations; however, this should only be done for tables that are both very important (e.g., resource-intensive) and exhibit significant locality in their access behavior.

(b) **Enumeration of all possible synopsis combinations:** As described in Section 5.4.3 the synopses selection problem can be solved for each relation independently; from the resulting sub-solutions the overall combination can then be obtained by solving Equation 5.5. To obtain the sub-solution for each relation $R_i \in \mathcal{R}$, first all possible synopsis combinations of $Opt\_Syn_{\mathcal{A},M}$ for R_i are computed. This is done by traversing the lattice of the attribute combinations in $Cand(R_i, \mathcal{Q})$ in the order of the sizes $|\mathcal{T}_x|$ of the data distributions at each node. For each node we compute all synopsis combinations possible from its attribute combinations A_i and all subsets of A_i corresponding to nodes in the lattice (as well as potential mappings from queries to synopses).

(c) **Minimization of the error values** As described in Section 5.4.2, each of the combinations of synopses and mappings corresponds to an *Opt_Err* expression, which is defined in the form of a minimization problem. In order to determine the best synopsis combination, it is necessary to compute the corresponding values for *Opt_Err*. This is done by constructing the corresponding synopses and evaluating the error for the resulting data distributions. The minimum *Opt_Err* expression corresponds to the optimal synopsis combination.

By combining the *enumeration* and *minimization* steps, it is furthermore possible to avoid solving identical minimization-problems more than once. Each time a new (sub-) combi-

nation of synopses/mapping is created, the corresponding minimization problem is solved immediately. Because each new combination is either created by joining two previously know combinations together, plus at most one additional synopsis, the corresponding minimization problem can be solved using the solutions for the two joining synopses in at most $O(M)$ steps.

(d) **Construction of the final synopses** The overall optimal solution can now be obtained from the sub-solutions for each relation by minimizing Equation 5.5.

5.7.1 Running Times

The computational overhead of our techniques is caused by (a) the computation of the candidate synopses, (b) the solving of the resulting minimization problems, and (c) the enumeration of all possible minimization problems. The running times for (a) and (b) are discussed in Section 3.5.3, including the cost of memory reconciliation. The loss in estimation quality connected with *GREEDY* synopsis construction is discussed in Section 3.5.2. In order to assess the cost of (c), enumerating all minimization problems, I measured the time the algorithm uses to construct all possible synopsis combinations for a given set of attributes $\mathcal{A}$ for which all possible subsets were queried (i.e. $2^{|\mathcal{A}|}$ different types of queries and thus the same worst-case number of potential synopses). The running times for this worst-case stress test are shown in Table 5.1. Obviously, even though the space of all combinations grows exponentially with the size of $\mathcal{A}$, the enumeration is still reasonably efficient for up to 20 attributes, which covers the range of query-relevant attributes in most tables (including join views) in relational databases.

# Attributes	Running time (sec.)	# Attributes	Running time (sec.)
4	0,009 sec.	12	1,23 sec.
8	0,049 sec.	16	93,93 sec.

Table 5.1: Running times for the enumeration of all synopses on a *SUN UltraSPARC 4000* (168 Mhz)

5.8 Experiments

To validate the approach and to demonstrate its accuracy, I have implemented the techniques and applied them to a scenario based on the *TPC-H* decision support benchmark. In this scenario, I compared the synopses selection techniques introduced in this paper against several simpler heuristics. Because I am not aware of other approaches to the given problem, these heuristics are not intended to represent opponents. Rather, some represent assumptions commonly used in connection with synopses selection in commercial database systems. Others are used to examine how much approximation accuracy is affected by simpler approaches to either synopses selection or memory allocation.

5.8.1 Base Experiment

I used a subset of the queries of *TPC-H*, chosen to be large enough to make the synopses-selection problem non-trivial yet small enough to facilitate understanding of the resulting physical design. The queries selected were Q_1, Q_6, Q_{13}, Q_{15} and Q_{17}, referring to the LINEITEM, PART, ORDERS,

and CUSTOMER tables[5]. Table 5.2 shows the query-relevant attribute sets, the *minimum sets* $Min(Q_i)$, for the above five queries.

Query	Min-Set
Q_1	L.SHIPDATE
Q_6	L.SHIPDATE, L.DISCOUNT, L.QUANTITY
Q_{13}	J_1.EXTENDED_PRICE, J_1.CLERK, J_1.DISCOUNT, J_1.RETURN_FLAG
Q_{15}	L.EXTENDED_PRICE, L.SHIPDATE, L.DISCOUNT
Q_{17}	J_2.CONTAINER, J_2.DISCOUNT, J_2.QUANTITY, J_2.BRAND

Table 5.2: The *Mininmum Sets* for the used queries

I chose the minimum sets in Table 5.2 according to a result-size approximation scenario, i.e., I only selected those attributes that are necessary to estimate the *number* of tuples in the query results (for queries which have an aggregation as the last operator, the result-size was estimated before the aggregation). This results in five multidimensional data distributions. Three of these are projections of the LINEITEM table, referred to as L onto subsets of its attributes (which all overlap so that there are a number of different, potentially suitable combinations of synopses). The other two data distributions to be approximated are join synopses J_1 := LINEITEM $\bowtie$ ORDERS and J_2 := LINEITEM $\bowtie$ PART. For our experiments, I used a scaled-down version of the *TPC-H* data with scale factor SF= 1/100) and SF $*$ 500 KBytes memory available for all synopses together).

The physical design technique presented in this paper was compared to six heuristic competitors that were generated from the following two option sets for synopses selection and memory allocation.

Synopses selection:

Single A single-dimensional synopsis was allocated for each attribute that appears at least once in the minimum sets of the five queries. While this heuristics cannot be expected to perform comparably to the more sophisticated allocation schema, I included it since most commercial database systems still use one-dimensional synopses/histograms only. So this heuristic gives an idea of the loss in accuracy when ignoring multi-attribute correlation.

Table One multidimensional synopsis is allocated for each table, and this synopsis covers all attributes of the table that appear in the minimum sets. This heuristic results in a large single synopsis reflecting all correlations between attributes. However, because of the merge-property, its accuracy may be significantly less than synopses using subsets of attributes.

and **Memory allocation:**

Uniform Each synopsis is given the same size. Again, this assumption can be found in commercial database systems.

Tuples The size of a synopsis is proportional to the size of the table that it refers to, measured in the number of tuples that reside in the table (where a join result is viewed as a table, too) multiplied with the number of attributes covered by the synopsis.

[5]The non-numerical values present in a TPC-H database are coded as numbers. For example, P.BRAND consists of a constant text string and two integers in the range $[1, 5]$. We only store the 25 possible number combinations.

Values The size of a synopsis is proportional to the size of the unique value combinations among the attributes over which the synopsis is built.

The synopsis-selection technique of this paper is referred to as *Opt_Syn*, the corresponding memory reconciliation as *Opt_Size*. To illustrate the importance of memory reconciliation for our overall approach, we also combined our synopsis-selection with the *Uniform*, *Tuples* and *Values*-based memory allocation; i.e., the optimal set of synopses was first generated and the sizes of these synopses were then computed using the above heuristics. For each set of synopses I executed 1000 instances of each query (using different, uniformly distributed, inputs for the query parameters, as specified in the benchmark) and used the available synopses to estimate the result sizes. I measured the average relative error of the result sizes:

$$\textbf{Relative\_Error} := \frac{1}{n}\sum_{i=1,\ldots,n}\frac{|(exact\_size(i) - estimated\_size(i))|}{exact\_size(i)}$$

with n being the number of instances of all queries together. All queries occur with the same frequency in all experiments. The results of the first experiment are shown in the first three columns of Table 5.3. In this set of experiments, the technique employed for synopses selection

Selection	Memory	Original data	Skewed data	Query locality
Single	Uniform	1.98	7.98	10.19
	Tuples	1.98	7.42	9.72
	Values	1.92	7.62	9.34
Table	Uniform	1.46	3.14	4.96
	Tuples	1.47	3.17	5.11
	Values	1.43	3.47	5.01
Opt_Syn	Uniform	1.05	1.14	1.04
	Values	1.04	1.01	1.27
	Tuples	1.03	1.08	1.17
	Opt_Size	1.04	0.83	0.85

Table 5.3: Experiment I, II and III : The *relative_error* for the original *TPC-H* data, skewed data and data with strong locality

had significant impact on the resulting approximation accuracy, whereas the way memory is allocated only results in negligible changes to the overall error.

5.8.2 Skewed and Correlated Data

As described in Section 4.2, for purposes of approximation it is crucial to preserve the correlation contained in the data. Unfortunately, the original TPC-H data is generated using uniformly random distributions for each attribute, resulting in almost completely uncorrelated data[6], which is not a good benchmark for data approximation techniques. Therefore, I ran a second set of experiments using the same schema, but with skewed and correlated data. This more realistic kind of data was generated the following way:

[6]The exceptions being O.Totalprice (correlated with L.Tax, L.Discout, L.Extendedprice), L.Shipdate (correlated with O.Orderdate), L.Commitdate (correlated with O.Orderdate) and L.Receiptdate (correlated with L.Shipdate).

Skew in attribute-value frequencies. The attribute-value frequencies were generated so that the frequency of the attribute values was Zipf-like distributed; i.e., the frequency of the i-th most frequent value is proportional to $(1/i)^\theta$ where θ is a control parameter for the degree of skew. In this experiment $\theta = 1$ was used.

Correlation between attributes Here the generated data was permuted in order to obtain the desired correlation. After creating the data according to the TPC-H specification, then (randomly chosen) permutations were performed on the values of selected attributes in order to create specific correlations between pairs of attributes. The correlation itself was specified in terms of the linear correlation coefficient r_s as defined in Equation 4.1. For each of the following pairs of attributes data with a linear correlation coefficient $r_s \in [0.725, 0.775]$ was created: (L.SHIPDATE, L.QUANTITY), (J_2.BRAND, J_2.CONTAINER), (P.PARTKEY, P.BRAND).
The results for this experiment are shown in the fourth column of Table 5.3. Again, the choice of the synopses-selection technique was most important with regards to the resulting approximation error: the *Opt_Syn* technique developed in this paper reduced the error by a factor of 7 and 3 compared to the Single and Table heuristics, respectively. In addition, with *Opt_Syn* for synopses selection, the use of our memory reconciliation technique *Opt_Size* resulted in noticeable further improvement. So for this more realistic dataset, the combination of *Opt_Syn* and *Opt_Size* outperformed all competitors by a significant margin.

5.8.3 Query Locality

The above experiments were repeated using a workload that exhibited significant locality, again using the data exhibiting significant skew and correlation. For this experiment, the input parameters were generated for the TPC-H queries using a Zipf-like distribution ($\theta = 0.25$), first executing 1000 queries of each type to obtain the weights w_i (see equation 5.2) then used to construct the synopses. Subsequently, I ran another 1000 queries (with different parameters generated by the same probability distribution) for which the error was measured. The results for this experiment are shown in the fifth column of Table 5.3. The trends from the previous experiment can be observed here as well: the synopses-selection technique clearly outperforms the simpler approaches, with the estimation accuracy further improving when memory reconciliation is used.

6 Conclusion

When in Rome, burn it.

- GCU Arbitrary, "The state of the art"

In this thesis, I have developed a new framework for providing query result (size) estimation based on spline synopses. Unlike most previous approaches, spline synopses provide a unified approach for a number of different estimation problems, such as accurate estimation for (combinations of) different operators, representation of the correlation present between both the values of different attributes in one relation and also multidimensional attribute values and their frequencies, and the physical design problem for data synopses. Spline synopses have been designed based on the set of design choices identified through the study of the design space for data synopses in Chapter 2. They have been shown to cause low overhead, and with regards to estimation accuracy consistently outperform the best known histogram techniques and offer estimation accuracy to specialized techniques geared towards minimizing the estimation error for a single operator only.

While a number of important questions has been answered in this work, it has also raised new issues that may lead to interesting areas of research. In the following, I will briefly discuss some of these.

Further Pruning through Principal Component Analysis In order to achieve further reduction on the number of synopsis combinations considered in the solution of the physical design problem for data synopses, I hope to be able to characterize very weak and very strong correlation between values of different attributes in compact form, and then prune synopses that contain the joint data distribution over these attributes. Future work will include exploring the use of *Principal Component Analysis* as a dimensionality reduction technique to exploit situations when two or more attributes are highly correlated or, in the extreme form, functionally dependent.

Context-based Space-Filling Curves and Data Rearrangements The mapping of multidimensional data to a linear domain by the Sierpiński curve is limited in the sense that many data sets that show a very simple structure in their original domain, do not exhibit this structure after mapping and thus become much harder to approximate. For example, consider a data distributions that have a value distribution corresponding to a grid. A solution to this problem would the the construction of *context-based* space-filling curves [DCOM00], that adapt themselves to the given data and are then used in place of the Sierpiński curve.

The tradeoff involved here is the fact that now the shape of the context-based curve has to be stored in addition to the approximation, thus making this approach only feasible, when the resulting increase in approximation accuracy is significant. In [DCOM00], where the context-based curves are used for data compression, the approach did not result in improved compression; however, the formalism used to describe curves was able to express a very large number of different curves, thereby resulting in prohibitive storage overhead for the curve description itself. In the

scenario of data synopses, the context-based curves would only be employed, when the multi-dimensional distribution of attribute values exhibits a significant amount of structure, which can the be leveraged in the curve construction. Suitable techniques for the detection of structure (most importantly self-similarity) and variable-sized encoding of curve descriptions have been developed in the context of iterated function systems. Combining context-based curves with the techniques of this thesis may result in a hybrid approach that represents value distribution partially through linear fitting and partially through encoding of structure information through a suitable curve.

A closely related area is the study of how data can be rearranged to improve the efficiency of compression or approximation. While this combination may result in very difficult problems (for example, concerning compression through run-length encoding, the problem of finding a data rearrangement that minimizes the number of runs is NP-complete [OR86]), the fact that the simple process of sorting the data by value frequencies results in a class of histograms [IC93] optimal for many estimation problems, may lead to a hybrid approach similar to the one discussed above for spline synopses where the data is initially rearranged, then approximated via spline synopses and after estimation the reordering reversed.

7 Summary of this Thesis

Query estimation is vital for a number of applications in the context of database systems, most importantly query estimation. Techniques used in this scenario must be able to estimate queries made up of arbitrary combinations of selection, projection, and join operators, use the space available for synopses storage efficiently (for more memory may then be used for the page cache and workspace buffers), must be able to represent the correlation present between both values of different attributes and between attribute values and their frequencies, and provide both query result and size estimation.

To this end, I have proposed a technique called spline synopses, that is based on independent and separate approximations of the distributions of attribute values and their frequencies. This makes it possible to assign more memory to the more difficult or important estimation problem, which is crucial for overall accuracy, for both types of approximation are relevant for query estimation. For both approximation types, efficient algorithms have been developed that compute the optimal approximation for each domain and also find the best combination of both types of approximations. To ensure accurate estimation of join queries, an additional class of join synopses has been introduced, which was shown to limit the worst-case join error and improved the join estimation for experimental data by several orders of magnitude.

Through the use of the Sierpiński curve to map multi-dimensional data to a linearly ordered domain (where it is approximated) and back (at estimation time), it is possible to extend spline synopses to represent the correlation between multiple attributes, while still being able to leverage their other advantages. The only disadvantage of this approach is a slight increase in the overhead of query estimation.

I have motivated and defined the physical design problem for data synopses, and developed an algorithmic approach to its solution. Based on two general observations on the properties of data synopses it is possible to prune the space of synopsis combinations that need to be considered by the algorithm significantly. As these properties hold for a wide class of data synopsis techniques it is further possible to extend the algorithm (within the limits of the base synopses employed) to other techniques such as histograms. For large instances of the design problem, heuristics exist that alleviate computational bottlenecks.

For all of the studied subproblems, my algorithms were shown to have small enough overhead to be used in practice and to be efficient with regard to the tradeoff between used memory and estimation accuracy. Regarding estimation accuracy, spline synopses consistently outperform the best known histogram techniques, while offering accuracy competitive with specialized techniques geared towards range-selectivity estimation only. Regarding the physical design problem for data synopses, it has been shown that the proposed algorithm significantly increases the estimation accuracy compared to combinations of various heuristics. This is the first comprehensive solution

to the physical design problem for data synopses.

In summary, spline synopses satisfy all requirements on query estimation techniques and outperform all previously proposed techniques, thereby increasing the flexibility, efficiency, and practicability of query estimation in database systems.

8 Zusammenfassung der Arbeit

Das akkurate Schätzen der Größe von Datenbankanfragen ist von entscheidender Bedeutung für diverse Anwendungen im Bereich von Datenbanksystemen, insbesondere bei der Auswahl des besten Ausführungsplans einer Anfrage. Techniken, die hierfür eingesetzt werden, müssen in der Lage sein, die Resultate von Anfragen, die aus beliebigen Kombinationen der Operatoren *Selektion*, *Projektion* und *Join* bestehen, schätzen zu können. Weiterhin müssen sie den Speicher, der für Daten Synopsen reserviert ist, effizient nutzen (so dass mehr Speicher für den Seitencache und Ausführungspuffer bleibt), in der Lage sein, die Korrelationen zwischen den Werten verschiedener Attribute innerhalb einer Relation und zwischen verschiedenen Attributwerten und deren Frequenz zu repräsentieren, und akkurate Abschätzungen sowohl für das Resultat einer Anfrage als auch dessen Größe liefern.

Zu diesem Ziel wurden *Spline Synopsen* vorgestellt, eine Technik, die auf der separaten und unabhängigen Approximation der Verteilung der Attributwerte eines Datensatzes und der Verteilung der Häufigkeiten dieser Attributwerte basiert. Dieser Ansatz ermöglicht es, anders als bei bisherigen Techniken, mehr Speicher für das wichtigerer bzw. schwerere Approximations-Problem zu verwenden. Da beide Arten der Approximation relevant für die Abschätzung von Anfragen sind, ist dies essentiell, um hohe Genauigkeit der Abschätzungen zu erzielen. In beiden Fällen war es möglich, effiziente Algorithmen zur Bestimmung der optimalen Approximation für eine gegebene gegebenen Verteilung zu formulieren. Um ferner die genaue Abschätzung von Join-Operatoren zur gewährleisten, wurden zusätzliche *Join Synopsen* eingeführt, welche den Schätzfehler für Join-Anfragen sowohl im Bezug auf dessen Obergrenze, als auch in Experimenten mit synthetischen Datenverteilungen effektiv minimieren.

Durch die Benutzung der Sierpiński Kurve war es ferner möglich, mehr-dimensionale Datenverteilungen in einen linear geordneten Raum (wo diese dann approximiert werden) und die resultierende Approximation (zum Zeitpunkt der Abschätzung) wieder in den ursprünglichen Raum abzubilden. Durch diesen Ansatz konnten Spline Synopsen für die Approximation mehrdimensionaler Daten eingesetzt werden, ohne ihre sonstigen Eigenschaften zu verlieren. Der einzige Nachteil unserer Methode im Vergleich zur Approximation ein-dimensionaler Daten ist zusätzlicher Rechenaufwand bei der Abschätzung einer Anfrage.

Das Problem, die optimale Kombination von Synopsen für eine gegebene Kombination aus Daten, Anfragen und verfügbarem Speicher zu bestimmen, wurde zunächst motiviert, und ein algorithmischer Ansatz zu dessen Lösung entwickelt. Aufgrund von zwei allgemeinen Eigenschaften von Daten Synopsen war es möglich, den Suchraum der Kombinationen von Synopsen, welche hierfür betrachtet werden müssen, stark einzuschränken. Da diese Eigenschaften nicht nur für Spline Synopsen, sondern für eine Vielzahl von Techniken gelten, ist es möglich, den algorithmischen Ansatz (im Rahmen der Möglichkeiten der genutzten Daten Synopsen) auf diese Techniken, wie z.B. Histogramme, auszuweiten. Für große Instanzen des Problems wurden ferner Heuristiken

vorgeschlagen, welche gezielt den Aufwand für rechenintensive Teilprobleme verringern.

Für alle untersuchten Bereiche konnte gezeigt werden, dass die resultierenden Algorithmen effizient genug sind, um in der Praxis Verwendung zu finden und zudem den verfügbaren Speicher effektiv nutzen. Im Bezug auf die Schätzgenauigkeit übertreffen Spline Synopsen durchgehend die besten bekannten Histogramm-basierten Ansätze und bieten mit auf einzelne Operatoren spezialisierten Techniken vergleichbare Resultate. Ferner erhöht die Auswahl der Synopsen durch den vorgestellten Algorithmus die Schätzgenauigkeit gegenüber der Synopsenauswahl durch Kombinationen diverser Heuristiken drastisch. Unsere Arbeit ist die erste umfassende Studie des Auswahlproblems für Datensynopsen.

Zusammenfassend ist festzuhalten, dass Spline Synopsen alle Anforderungen an Anfrage-Abschätzung in Datenbanksystemen erfüllen, und in Bezug auf Genauigkeit, Flexibilität und Effizienz die bisher bekannten Techniken überbieten.

Bibliography

[AC99] Ashraf Aboulnaga and Surajit Chaudhuri. Self-tuning Histograms: Building Histograms Without Looking at Data. In *Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, Philadephia, Pennsylvania*, pages 181–192. ACM Press, 1999.

[AGPR99] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. Join Synopses for Approximate Query Answering. In *Proceedings of the ACM SIGMOD Conference*, pages 275–286. ACM Press, 1999.

[AK95] Charles J. Alpert and Andrew B. Kahng. Multi-Way Partitioning Via Geometric Embeddings, Orderings and Dynamic Programming. *IEEE Transactions on CAD*, 14(11):1342–1358, 1995.

[Ant93] Gennady Antoshenkov. Dynamic Query Optimization in Rdb/VMS. In *Proceedings of the Ninth International Conference on Data Engineering, April 19-23, Vienna, Austria*, pages 538–547, 1993.

[APR99] Swarup Acharya, Viswanath Poosala, and Sridhar Ramaswamy. Selectivity Estimation in Spatial Databases. In *Proceedings ACM SIGMOD Conference*, pages 13–24. ACM Press, 1999.

[AZ96] Gennady Antoshenkov and Mohamed Ziauddin. Query Processing and Optimization in Oracle Rdb. *VLDB Journal*, 5(4):229–237, 1996.

[Bar88] Michael F. Barnsley. *Fractals Everywhere*. Academic Press, New York, 1988.

[BCG01] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. STHoles: A Multidiemsnional Workload-Aware Histogram. In *Proceedings of the ACM SIGMOD Conference*, Santa Barbara, California, 2001.

[BDF+97] Daniel Barbará, William DuMochel, Christos Faloutsos, Peter J. Haas, Joseph M. Hellerstein, Yannis Ioannidis, H.V. Jagadish, Theodore Johnson, Raymond Ng, Viswanath Poosala, Kenneth A. Ross, and Kenneth C. Sevcik. The New Jersey Data Reduction Report. *IEEE D.E. Bulletin*, 1997.

[BGM02] Nicolas Bruno, Luis Gravano, and Amélie Marian. Evaluating Top-k Queries over Web-Accessible Databases. In *Proceedings of the 18th ICDE Conference*, Santa Jose, California, 2002.

[BKS99] Bjorn Blohsfeld, Dieter Korus, and Bernhard Seeger. A Comparison of Selectivity Estimators for Range Queries on Metric Attributes. In *Proceedings of the ACM SIGMOD Conference*, pages 239–250, 1999.

[CCMN00] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. Towards Estimation Error for Distinc Values. In *Proceedings of the ACM PODS Conference*, pages 268–276, 2000.

[CFM95] Alberto Caprara, Matteo Fischetti, and Dario Maio. Exact and approximate algorithms for the index selection problem in physical database design. *TKDE*, 7(6):955–967, 1995.

[CG94] Richard L. Cole and Goetz Graefe. Optimization of Dynamic Query Evaluation Plans. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, USA*, pages 150–160, 1994.

[CG99] Surajit Chaudhuri and Luis Gravano. Evaluating Top-k Selection Queries. In *Proceedings of the 25th VLDB Conference*, pages 399–410, 1999.

[CGRS00] Kaushik Chakrabarti, Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Approximate query processing using wavelets. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 111–122, 2000.

[Cha98] Surajit Chaudhuri. An Overview of Query Optimization in Relational Systems. In *Proceedings of ACM PODS Conference*, pages 34–43, 1998.

[Chr83] Stavros Christodoulakis. Estimating Block Transfers and Join Sizes. In *Proceedings of the ACM SIGMOD Conference*, pages 40–54. ACM Press, 1983.

[Chr84] S. Christodoulakis. Implications of certain Assumptions in Database Performance Evaluation. In *ACM TODS*, pages 163–186, June 1984.

[CHS01] Rada Chirkova, Alon Y. Halevy, and Dan Suciu. A Formal Perspective on the View Selection Problem. In *Proc. of the 27th Int. Conference on Very Large Databases*, 2001.

[CMN98] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Random Sampling for Histogram Construction: How much is enough? In *Proceedings of ACM SIGMOD Conference*, pages 436–447, 1998.

[CMN99] Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. On Random Sampling over Joins. In *Proceedings of the ACM SIGMOD Conference*, pages 263–274, 1999.

[CN99] Surajit Chaudhuri and Vivek R. Narasayya. Index Merging. In *Proceedings of the 15th International Conference on Data Engineering, 23-26 March 1999, Sydney, Austrialia*, pages 296–303, 1999.

[CN00] Surajit Chaudhuri and Vivek R. Narasayya. Automating Statistics management for Query Optimizers. In *Proceedings of the IEEE Conference on Data Engineering*, pages 339–348, 2000.

[CR94] Chungmin Melvin Chen and Nick Roussoploulos. Adaptive Selectivity Estimation Using Query Feedback. In *Proceedings of the ACM SIGMOD Conference*, pages 161–172, May 1994.

[CWBC95] Sunil Choenni, Henk Wagterveld, Henk M. Blanken, and Thiel Chang. TOPYDE: a Tool for Physical Database Design. In *6. Conference on Database and Expert Systems Applications*, pages 502–511, London,UK, 1995.

[DaJB93] David DeWitt and Jeffrey Naughton an J. Burger. Nested Loops revisited. In *International Conference on Parallel and Distributed Information Systems*, January 1993.

[dB78] Carl de Boor. *A practical guide to splines*. Springer-Verlag, 1978.

[DCOM00] Revital Dafner, Daniel Cohen-Or, and Yossi Matias. Context-based Space Filling Curves. In *Eurographics 2000*, volume 19, 2000.

[DGR01] Amol Deshpande, Minos Garofalakis, and Rajeev Rastogi. Independence is Good: Dependency-Based Histogram Synopses for High-Dimensional Data. In *Proceedings of the ACM SIGMOD Conference*, 2001.

[Die93] Paul Dierckx. *Curve and Surface Fitting with Splines*. Monographs on Numerical Analysis. Oxford Science Publications, 1993.

[DIR00] Donko Donjerkovic, Yannis Ioannidis, and Raghu Ramakrishnan. Dynamic Histograms: Capturing Evolving Data Sets. In *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA*, page 86, 200.

[DR99] Donko Donjerkovic and Raghu Ramakrishnan. Probabilistic Optimization of Top N Queries. In *Proceedings of 25th International Conference on Very Large Data Bases, Edinburgh, Scotland, UK*, pages 411–422, 1999.

[FST88] Sheldon J. Finkelstein, Mario Schkolnick, and Paolo Tiberio. Physical Database Design for Relational Databases. *TODS*, 13(1):91–128, 1988.

[GAB+98] Phillip B. Gibbons, S. Acharya, Y. Bartal, Y. Matias, S. Muthukrishnan, V. Poosala, S. Ramaswamy, and T. Suel. Aqua: System and techniques for approximate query answering. Technical report, Bell Labs, 1998.

[GD87] Goetz Graefe and David J. DeWitt. The EXODUS Optimizer Generator. In *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, California, May 27-29, 1987*, pages 160–172. ACM Press, 1987.

[GGMS96] Sumit Ganguly, Philip B. Gibbons, Yossi Matias, and Avi Silberschatz. Bifocal Sampling for skew-resistant Join-Size Estimation. In *Proc. of the ACM SIGMOD Conf*, 1996.

[GKTD00] Dimitrios Gunopulos, George Kollios, Vassilis J. Tsotras, and Carlotta Domeniconi. Approximating Multi-Dimensional Aggregate Range Queries Over Real Attributes. In *Proceedings of the ACM SIGMOD Conference*, 2000.

[GLR00] Venkatesh Ganti, Mong-Li Lee, and Raghu Ramakrishnan. Icicles: Self-tuning samples for approximate query answering. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 176–187, 2000.

[GLSW93] Peter Gassner, Guy M. Lohman, K. Bernhard Schiefer, and Yun Wang. Query Optimization in the IBM DB2 Family. *Data Engineering Bulletin*, 16(4):4–18, 1993.

[GM93] Goetz Graefe and William J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *Proceedings of the Ninth International Conference on Data Engineering, April 19-23, Vienna, Austria*, pages 209–218. IEEE Computer Society, 1993.

[GM98] Philip B. Gibbons and Yossi Matias. New Sampling-Based Summary Statistics for Improving Approximate Query Answers. In *Proceedings of the ACM SIGMOD Conference*, 1998.

[GM99] Phillip B. Gibbons and Yossi Matias. Synopsis Data Structures for Massive Data Sets. In *Symposium on Discrete Algorithms*, 1999.

[GMP97] Phillip B. Gibbons, Yossi Matias, and Viswanath Poosala. Fast Incremental Maintenance of Approximate Histograms. In *Proceedings of the 23rd International Conference on Very Large Databases*, 1997.

[Gra53] F. Gray. Pulse Code Communications. US Patent 2632058, March 17 1953.

[GS92] Jun Gao and J. Michael Steele. General Spacefilling Curve Heuristics and Limit Theory for the Traveling Salesman Problem. *Journal of Complexity*, 10:230–245, 1992.

[GTK01] Lise Getoor, Ben Taskar, and Daphne Koller. Selectivity Estimation using Probabilistic Models. In *Proceedings of the ACM SIGMOD Conference*, 2001.

[Haa96] Peter J. Haas. Selectivity and Cost Estimation for Joins Based on Random Sampling. *Journal of Computer and System Sciences*, pages 550–569, 1996.

[HNSS95] P.J. Haas, J.F. Naughton, S. Seshardi, and L. Stokes. Sampling-based Estimation of the Number of Distinct Values of an Attribute. In *Proc. of the 21st International Conference on Very Large Databases*, 1995.

[HS92] Peter J. Haas and Arun N. Swami. Sequential Sampling Procedures for Query Size Estimation. In *Proceedings of the ACM SIGMOD Conference*, pages 341–350, 1992.

[IC93] Yannis Ioannidis and Stavros Christodoulakis. Optimal Histograms for limiting Worst-Case Error Propagation in the Size of Join Results. In *ACM TODS*, 1993.

[INSS97] Yannis E. Ioannidis, Raymond T. Ng, Kyuseok Shim, and Timos K. Sellis. Parametric Query Optimization. *VLDB Journal*, 6(2):132–151, 1997.

[Ioa93] Yannis Ioannidis. Univeratility of Serial Histograms. In *Proceedings of the 19th VLDB Conference*, pages 256–267, December 1993.

[IP95] Yannis Ioannidis and Viswanath Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. In *Proceedings of the ACM SIGMOD Conference*, pages 233–244, May 1995.

[IP99] Yannis E. Ioannidis and Viswanath Poosala. Histogram-Based Approximation of Set-Valued Query-Answers. In *Proceedings of 25th International Conference on Very Large Data Bases*, pages 174–185, 1999.

[Ive62] Kenneth A. Iverson. *A Programming Language*. Wiley, 1962.

[JJOT01] H.V. Jagadish, Hui Jin, Beng Chin Ooi, and Kian-Lee Tan. Global Optimization of Histograms. In *Proceedings of the ACM SIGMOD Conference*. ACM Press, 2001.

[JKM+98] H. V. Jagadish, N. Koudas, S. Mutukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal Histograms with Quality Guarantees. In *Proceedings 24th International Conference on Very Large Databases*, pages 275–286, 1998.

[JNS99] H. V. Jagadish, Raymond T. Ng, and Divesh Srivastava. Substring Selectivity Estimation. In *Proceedings of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, Pennsylvania*, pages 249–260. ACM Press, 1999.

[Jup78] D.L.B. Jupp. Approximation to data by splines with free knots. *SIAM Journal on Numerical Analysis*, 15:328–343, 1978.

[KD98] Navin Kabra and David J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *Proceedings of the ACM SIGMOD Conference*, 1998.

[KDD] The UCI KDD Archive (http://kdd.ics.uci.edu).

[KF94] Ibrahim Kamel and Christos Faloutsos. Hilbert R-Tree: An Improved R-Tree using Fractals. In *Proceedings of the 20th International Conference on Very Large Databases*, pages 500–509, 1994.

[KW99] A.C. König and G. Weikum. Combining Histograms and Parametric Curve Fitting for Feedback-Driven Query Result-size Estimation. In *25th International Conference on Very Large Databases*, 1999.

[KW00] A.C. König and G. Weikum. Auto-Tuned Spline Synopses for Database Statistics Management. In *10th International Conference on the Management of Data, Pune, India (COMAD)*, 2000.

[KW02] A.C. König and G. Weikum. A Framework for the Physical Design Problem on Data Synopses. In *Advances in Database Technology - EDBT 2002*, Prague, Czech Republic, 2002.

[Lim90] J.S. Lim. *Two Dimensional Signal and Image Processing*. Prentice Hall, 1990.

[LK01] Jonathan K. Lawder and Peter J. H. King. Querying Multi-dimensional Data Indexed Using the Hilbert Space-filling Curve. *SIGMOD Record*, 30(1):19–24, 2001.

[LKC99] Ju-Hong Lee, Deok-Hwan Kim, and Chin-Wan Chung. Multi-dimensional Selectivity Estimation Using Compressed Histogram Information. In *Proceedings of the ACM SIGMOD Conference*, pages 205–214, 1999.

[LS95] Yibei Ling and Wei Sun. An Evaluation of Sampling-Based Size Estimation Methods for Selections in Database Systems. In *Proceedings of the ICDE Conference*, 1995.

[Lyn88] Clifford A. Lynch. Selectivity Estimation and Query Optimization in Large Databases with Highly Skewed Distributions of Column Values. In *Proc. on the 14th International Conference on Very Large Databases*, pages 240–251, 1988.

[Man83] Benoit B. Mandelbrot. *The Fractal Geometry of Nature*. Freeman, New York, 1983.

[MD88] M. Muralikrishna and David J. Dewitt. Equi-depth histogramms for estimating selectivity factors for multi-dimensional queries. In *Proc. of the ACM SIGMOD Conf.*, pages 28–36, 1988.

[Mil80] S.C. Milne. Peano Curves and the Smoothness of Functions. In *Advances in Mathematics 35*, pages 129 – 157, 1980.

[MJFS01] Bongki Moon, H.V. Jagadish, Christos Faloutsos, and Joel H. Saltz. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, 2001.

[MNU97] K. Mehlhorn, St. Näher, and Ch. Uhrig. LEDA: Library of Efficient Datatypes and Algorithms. available via ftp:mpi-sb.mpg.de, 1997.

[MPS99] S. Muthukrishnan, Viswanath Poosala, and Thorsten Suel. On Rectangular Partitionings in Two Dimensions. In *Proceedings of International Conference on Database Theory*, pages 236–256, 1999.

[MVW98] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. Wavelet-Based Histograms for Selectivity Estimation. In *Proceedings of the ACM SIGMOD Conference*, pages 448–459. ACM Press, 1998.

[MVW00] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. Dynamic Maintenance of Wavelet-Based Histograms. In *Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, Cairo, Egypt*, pages 101–110. Morgan Kaufmann, 2000.

[Nel95] Randolph Nelson. *Probability, Stochastic Processes and Queueing Theory*. Springer-Verlag, 1995.

[NL00] Felix Naumann and Ulf Leser. Cooperative Query Answering with Density Scores. In *10th International Conference on the Management of Data, Pune, India*, pages 33–44, 2000.

[NP00] Joong Chae Na and Kunsoo Park. Data Compression with Truncated Suffix Trees. In *Data Compression Conference*, page 565, Snowbird, Utah, USA, 2000.

[O'N94] Patrick O'Neil. *Database – Principles, Programming, Performance*. Morgan Kaufman, 1994.

[OR86] Frank Olken and Doron Rotem. Rearranging Data to Maximize the Efficiency of Compression. In *Proceedings of the 5th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Cambridge, Massachusetts*, pages 78–90, 1986.

[PB89] Loren K. Platzman and John J. Bartholdi, III. Spacefilling Curves and the Planar Travelling Salesman Problem. *Journal of the ACM*, pages 719–737, Oct 1989.

[Pea90] G. Peano. Sur une courbe qui remplit toute une aire plane. *Math. Ann.*, 36:157–160, 1890.

[PI96] Viswanath Poosala and Yannis E. Ioannidis. Estimation of Query-Result Distribution and its Application in Parallel-Join Load Balancing. In *Proceedings of 22th International Conference on Very Large Data Bases, Mumbai (Bombay), India*, pages 448–459. Morgan Kaufmann, 1996.

[PI97] Viswanath Pooosala and Yannis E. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *Proceedings of the ACM SIGMOD Conference*, Athens, Greece, 1997.

[PIHS96] V. Poosala, Y.E. Ioannidis, P.J. Haas, and E.J. Shekita. Improved Histograms for Selectivity Estimation or Range Predicates. In *Proceedings of the ACM SIGMOD Conference*, pages 294–305. ACM Press, 1996.

[Poo97] Viswanath Poosala. *Histogram-based Estimation Techniques in Database Systems*. PhD thesis, University of Wisconsin-Madison, 1997.

[PSC84] Gregory Piatetsky-Shapiro and Charles Connel. Accurate Estimation of the Number of Tuples Satisfying a Condition. In *Proceedings of the ACM SIGMOD Conference, Boston, Massachusetts*, pages 256–276, 1984.

[PTVF96] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P Flannery. *Numerical Receipes in C*. Cambridge University Press, 1996.

[PYK+97] Jignesh M. Patel, Jie-Bing Yu, Navin Kabra, Kristin Tufte, Biswadeep Nag, Josef Burger, Nancy E. Hall, Karthikeyan Ramasamy, Roger Lueder, Curt Ellman, Jim Kupsch, Shelly Guo, David J. DeWitt, and Jeffrey F. Naughton. Building a Scaleable Geo-Spatial DBMS: Technology, Implementation, and Evaluation. In *Proceedings ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, USA*, pages 336–347, 1997.

[RS91] Steve Rozen and Dennis Shasha. A Framework for Automating Physical Database Design. In *17th International Conference on Very Large Data Bases, September 3-6, 1991, Barcelona, Catalonia, Spain*, pages 401–411, 1991.

[RY90] K.R. Rao and P. Yip. *Discrete Cosine Transform Algorithms, Advantages, Applications*. Academic Press, 1990.

[SAC+93] P. Selinger, M. Astrahan, D. Chamberlain, R. Lorie, and T. Price. Access Path Selection in Relational Database Managment Systems. In *Proc. ACM SIGMOD Conf.*, pages 23–34, 1993.

[Sco92] David W. Scott. *Multivariante Density Estimation*. Wiley, 1992.

[Sie12] Waclaw Sierpiński. Sur une nouvelle courbe contine qui remplit toute une air plane. *Bull. Acad. Sci. de Cracovie (Sci. math. en nat., Serie A)*, pages 462–478, 1912.

[SLMK01] Michael Stillger, Guy Lohman, Volker Markl, and Mokhtar Kandil. LEO - DB2's Learning Optimizer. In *Proceedings of the 27th Conference on Very Large Databases, Rome, Italy*, pages 19–28, 2001.

[SLRD93] Wei Sun, Yibei Ling, Naphtali Rishe, and Yi Deng. An instant and accurate Size Estimation Method for Joins and Selections in an Retrival-Intensive Environment. In *Proceedings of the ACM SIGMOD Conference*, pages 79–88, 1993.

[Sni92] Moshe Sniedovich. *Dynamic Programming*. Marcel Dekker, Inc., 1992.

[SR94] Eva Skubalska-Rafajlowicz. The Closed Curve Filling Multidimensional Cube, Technical Report no. 46/94. ICT Technical University of Wroclaw, 1994.

[SRS96] E.J. Stollnitz, T.D. Rose, and D.H. Salesin. *Wavelets for Computer Graphics – Theory and Applications*. Morgan Kaufman Publishers, Inc., San Francisco, CA, 1996.

[Ukk93] Esko Ukkonen. Approximate String-Matching over Suffix Trees. In *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching*, pages 228–242, Padova, Italy, 1993.

[Vit85] Jeffrey Scott Vitter. Random Sampling with a Reservoir. In *ACM Proceedings on Mathematical Software, Volume 11*, pages 37–57. ACM Press, 1985.

[VW99] Jeffrey Scott Vitter and Min Wang. Approximate Computation of Multidimensional Aggregates of Sparse Data using Wavelets. In *Proceedings 1999 ACM SIGMOD International Conference on Management of Data, June 1-3, Philadephia, Pennsylvania, USA*, pages 193–204. ACM Press, 1999.

[VWI98] Jeffrey Scott Vitter, Min Wang, and Bala Iyer. Data Cube Approximation and Histograms via Waveletts. In *Proceedings of the 7th International Conference on Information and Knowledge Managment*, pages 96–104, November 1998.