Foundations and Trends® in
sample
Vol. xx, No xx (xxxx) 1–275

DOI: xxxxxx

now
the essence of knowledge

Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches

Graham Cormode[1], Minos Garofalakis[2], Peter J. Haas[3] and Chris Jermaine[4]

[1] *AT&T Labs—Research, 180 Park Avenue, Florham Park, NJ, 07932, USA,* graham@research.att.com
[2] *Technical University of Crete, University Campus—Kounoupidiana, Chania, 73100, Greece,* minos@acm.org
[3] *IBM Almaden Research Center, 650 Harry Road, San Jose, CA, 95120-6099, USA,* peterh@almaden.ibm
[4] *Rice University, 6100 Main Street, Houston, TX, 77005, USA,* cmj4@cs.rice.edu

Abstract

Methods for approximate query processing are essential for dealing with massive data. They are often the only means of providing interactive response times when exploring massive datasets, and are also needed to handle high speed data streams. These methods proceed by computing a lossy, compact synopsis of the data, and then executing the query of interest against the synopsis rather than the entire data set. We describe basic principles and recent developments in approximate query processing. We focus on four key synopses: random samples, histograms, wavelets, and sketches. We consider issues such as accuracy, space and time efficiency, optimality, practicality, range of applicability, error bounds on query answers, and incremental maintenance. We also discuss the trade-offs between the different synopsis types.

Contents

1 Introduction **1**

1.1 The Need for Synopses 2
1.2 Survey Overview 4
1.3 Outline 5

2 Sampling **10**

2.1 Introduction 10
2.2 Some Simple Examples 11
2.3 Advantages and Drawbacks of Sampling 15
2.4 Mathematical Essentials of Sampling 20
2.5 Different Flavors of Sampling 30
2.6 Sampling and Database Queries 36
2.7 Obtaining Samples from Databases 52
2.8 Online Aggregation Via Sampling 61

3 Histograms **64**

3.1 Introduction 65

3.2 One-Dimensional Histograms: Overview 74
3.3 Estimation Schemes 79
3.4 Bucketing Schemes 88
3.5 Multi-Dimensional Histograms 105
3.6 Approximate Processing of General Queries 118
3.7 Additional Topics 126

4 Wavelets 135

4.1 Introduction 135
4.2 One-Dimensional Wavelets and Wavelet Synopses: Overview 136
4.3 Wavelet Synopses for Non-L_2 Error Metrics 145
4.4 Multi-Dimensional Wavelets 161
4.5 Approximate Processing of General SQL Queries 170
4.6 Additional Topics 173

5 Sketches 189

5.1 Introduction 189
5.2 Notation and Terminology 191
5.3 Frequency Based Sketches 198
5.4 Sketches for Distinct Value Queries 226
5.5 Other topics in sketching 242

6 Conclusions and Future Research Directions 248

6.1 Comparison Across Different Methods 248
6.2 Approximate Query Processing in Systems 251
6.3 Challenges and Future Directions in AQP 254

References 260

1

Introduction

A synopsis of a massive data set captures vital properties of the original data while typically occupying much less space. For example, suppose that our data consists of a large numeric time series. A simple summary allows us to compute the statistical variance of this series: we maintain the sum of all the values, the sum of the squares of the values, and the number of observations. Then the average is given by the ratio of the sum to the count, and the variance is ratio of the sum of squares to the count, less the average. An important property of this synopsis is that we can build it efficiently. Indeed, we can find the three summary values in a single pass through the data.

However, we may need to know more about the data than merely its variance: how many different values have been seen? How many times has the series exceeded a given threshold? What was the behavior in a given time period? To answer such queries, our three-value summary does not suffice, and synopses appropriate to each type of query are needed. In general, these synopses will not be as simple or easy to compute as the synopsis for variance. Indeed, for many of these questions, there is no synopsis that can provide the exact answer, as is the case for variance. The reason is that for some classes of queries, the query answers collectively describe the data in full, and so any synopsis would effectively have to store the entire data set.

To overcome this problem, we must relax our requirements. In many cases, the key objective is not obtaining the exact answer to a query, but rather receiving an accurate estimate of the answer. For example, in many settings, receiving an answer that is within 0.1% of the true result is adequate for our needs; it might suffice to know that the true answer is roughly $5 million without knowing that the exact answer is $5,001,482.76. Thus we can tolerate *approximation*, and there are many synopses that provide approximate answers. This small relaxation can make a big difference. Although for some queries it is impossible to provide a small synopsis that provides exact answers, there are many synopses that provide a very accurate approximation for these queries while using very little space.

1.1 The Need for Synopses

The use of synopses is essential for managing the massive data that arises in modern information management scenarios. When handling large data sets, from gigabytes to petabytes in size, it is often impractical to operate on them in full. Instead, it is much more convenient to build a synopsis, and then use this synopsis to analyze the data. This approach captures a variety of use-cases:

- A search engine collects logs of every search made, amounting to billions of queries every day. It would be too slow, and energy-intensive, to look for trends and patterns on the full data. Instead, it is preferable to use a synopsis that is guaranteed to preserve most of the as-yet undiscovered patterns in the data.
- A team of analysts for a retail chain would like to study the impact of different promotions and pricing strategies on sales of different items. It is not cost-effective to give each analyst the resources needed to study the national sales data in full, but by working with synopses of the data, each analyst can perform their explorations on their own laptops.
- A large cellphone provider wants to track the health of its network by studying statistics of calls made in different regions, on hardware from different manufacturers, under different levels of contention, and so on. The volume of information is too large to retain in a database, but instead the provider can build a synopsis

of the data as it is observed live, and then use the synopsis off line for further analysis.

These examples expose a variety of settings. The full data may reside in a traditional data warehouse, where it is indexed and accessible, but is too costly to work on in full. In other cases, the data is stored as flat-files in a distributed file system; or it may never be stored in full, but be accessible only as it is observed in a streaming fashion. Sometimes synopsis construction is a one-time process, and sometimes we need to update the synopsis as the base data is modified or as accuracy requirement change. In all cases though, being able to construct a high quality synopsis enables much faster and more scalable data analysis.

From the 1990's through today, there has been an increasing demand for systems to query more and more data at ever faster speeds. Enterprise data requirements have been estimated [174] to grow at 60% per year through at least 2011, reaching 1,800 exabytes. On the other hand, users—weaned on Internet browsers, sophisticated analytics and simulation software with advanced GUIs, and computer games—have come to expect real-time or near-real-time answers to their queries. Indeed, it has been increasingly realized that extracting knowledge from data is usually an interactive process, with a user issuing a query, seeing the result, and using the result to formulate the next query, in an iterative fashion. Of course, parallel processing techniques can also help address these problems, but may not suffice on their own. Many queries, for example, are not embarrassingly parallel. Moreover, methods based purely on parallelism can be expensive. Indeed, under evolving models for cloud computing, specifically "platform as a service" fee models, users will pay costs that directly reflect the computing resources that they use. In this setting, use of Approximate Query Processing (AQP) techniques can lead to significant cost savings. Similarly, recent work [15] has pointed out that approximate processing techniques can lead to energy savings and greener computing. Thus AQP techniques are essential for providing, in a cost-effective manner, interactive response times for exploratory queries over massive data.

Exacerbating the pressures on data management systems is the increasing need to query streaming data, such as real time financial data or sensor feeds. Here the flood of high speed data can easily overwhelm the often limited CPU

and memory capacities of a stream processor unless AQP methods are used. Moreover, for purposes of network monitoring and many other applications, approximate answers suffice when trying to detect general patterns in the data, such as a denial-of-service attack. AQP techniques are thus well suited to streaming and network applications.

1.2 Survey Overview

In this survey we describe basic principles and recent developments in building approximate synopses (that is, lossy, compressed representations) of massive data. Such synopses enable approximate query processing, in which the user's query is executed against the synopsis instead of the original data. We focus on the four main families of synopses: random samples, histograms, wavelets, and sketches.

A *random sample* comprises a "representative" subset of the data values of interest, obtained via a stochastic mechanism. Samples can be quick to obtain, and can be used to approximately answer a wide range of queries.

A *histogram* summarizes a data set by grouping the data values into subsets, or "buckets," and then, for each bucket, computing a small set of summary statistics that can be used to approximately reconstruct the data in the bucket. Histograms have been extensively studied and have been incorporated into the query optimizers of virtually all commercial relational DBMSs.

Wavelet-based synopses were originally developed in the context of image and signal processing. The data set is viewed as a set of M elements in a vector—i.e., as a function defined on the set $\{0,1,2,\ldots,M-1\}$—and the wavelet transform of this function is found as a weighted sum of wavelet "basis functions." The weights, or coefficients, can then be "thresholded", e.g., by eliminating coefficients that are close to zero in magnitude. The remaining small set of coefficients serves as the synopsis. Wavelets are good at capturing features of the data set at various scales.

Sketch summaries are particularly well suited to streaming data. Linear sketches, for example, view a numerical data set as a vector or matrix, and multiply the data by a fixed matrix. Such sketches are massively parallelizable. They can accommodate streams of transactions in which data is both inserted and removed. Sketches have also been used successfully to estimate the answer to COUNT DISTINCT queries, a notoriously hard problem.

Many questions arise when evaluating or using synopses.

- What is the class of queries that can be approximately answered?
- What is the approximation accuracy for a synopsis of a given size?
- What are the space and time requirements for constructing a synopsis of a given size, as well as the time required to approximately answer the query?
- How should one choose synopsis parameters such as the number of histogram buckets or the wavelet thresholding value? Is there an optimal, i.e., most accurate, synopsis of a given size?
- When using a synopsis to approximately answer a query, is it possible to obtain error bounds on the approximate query answer?
- Can the synopsis be incrementally maintained in an efficient manner?
- Which type of synopsis is best for a given problem?

We explore these issues in subsequent chapters.

1.3 Outline

It is possible to read the discussion of each type of synopsis in isolation, to understand a particular summarization approach. We have tried to use common notation and terminology across all chapters, in order to facilitate comparison of the different synopses. In more detail, the topics covered by the different chapters are given below.

Sampling

Random samples are perhaps the most fundamental synopses for approximate query processing, and the most widely implemented. The simplicity of the idea—executing the desired query against a small representative subset of the data—belies centuries of research across many fields, with decades of effort in the database community alone. Many different methods of extracting and maintaining samples of data have been proposed, along with multiple ways to build an estimator for a given query. This chapter introduces the mathematical foundations for sampling, in terms of accuracy and precision, and discusses the key sampling schemes: Bernoulli sampling, stratified sampling, and simple random sampling with and without replacement.

For simple queries, such as basic SUM and AVERAGE queries, it is straightforward to build unbiased estimators from samples. The more general case—an arbitrary SQL query with nested subqueries—is more daunting, but can sometimes be solved quite naturally in a procedural way.

For small tables, drawing a sample can be done straightforwardly. For larger relations, which may not fit conveniently in memory, or may not even be stored on disk in full, more advanced techniques are needed to make the sampling process scalable. For disk-resident data, sampling methods that operate at the granularity of a block rather than a tuple may be preferred. Existing indices can also be leveraged to help the sampling. For large streams of data, considerable effort has been put into maintaining a uniform sample as new items arrive or existing items are deleted. Finally, "online aggregation" algorithms enhance interactive exploration of massive data sets by exploiting the fact that an imprecise sampling-based estimate of a query result can be incrementally improved simply by collecting more samples.

Histograms

The histogram is a fundamental object for summarizing the frequency distribution of an attribute or combination of attributes. The most basic histograms are based on a fixed division of the domain (equi-width), or using quantiles (equi-depth), and simply keep statistics on the number of items from the input which fall in each such bucket. But many more complex methods have been designed, which aim to provide the most accurate summary possible within a limited space budget. Schemes differ in how the buckets are chosen, what statistics are stored, how estimates are extracted, and what classes of query are supported. They are quantified based on the space and time requirements used to build them, and the resulting accuracy guarantees that they provide.

The one-dimensional case is at the heart of histogram construction, since higher dimensions are typically handled via extensions of one-dimensional ideas. Beyond equi-width and equi-depth, end biased and high biased, maxdiff and other generalizations have been proposed. For a variety of approximation-error metrics, dynamic programming (DP) methods can be used to find histograms—notably the "v-optimal histograms"—that minimize the error, subject to an upper bound on the allowable histogram size. Approximate methods can be used when the quadratic cost of the DP is not practical.

Many other constructions, both optimal and heuristic, are described, such as lattice histograms, STHoles, and maxdiff histograms. The extension of these methods to higher dimensions adds complexity. Even the two-dimensional case presents challenges in how to define the space of possible bucketings. The cost of these methods also rises exponentially with the dimensionality of the data, inspiring new approaches that combine sets of low-dimensional histograms with high-level statistical models.

Histograms most naturally answer range-sum queries—for example, "compute total sales between July and September for adults from age 25 through 40"—and their variations. They can also be used to approximate more general classes of queries, such as aggregations over joins. Various negative theoretical and empirical results indicate that one should not expect histograms to give accurate answers to arbitrary queries. Nevertheless, due to their conceptual simplicity, histograms can be effectively used for a broad variety of estimation tasks, including set-valued queries, real-valued data, and aggregate queries over predicates more complex than simple ranges.

Wavelets

The wavelet synopsis is conceptually close to the histogram summary. The central difference is that, whereas histograms primarily produce buckets that are subsets of the original data-attribute domain, wavelet representations transform the data and seek to represent the most significant features in a wavelet (i.e., "frequency") domain, and can capture combinations of high and low frequency information. The most widely discussed wavelet transformation is the Haar-wavelet transform (HWT), which can, in general, be constructed in time linear in the size of the underlying data array. Picking the B largest HWT coefficients results in a synopsis that provides the optimal L_2 (sum-squared) error for the reconstructed data. Extending from one-dimensional to multi-dimensional data, as with histograms, provides more definitional challenges. There are multiple plausible choices here, as well as algorithmic challenges in efficiently building the wavelet decomposition.

The core approximate query processing task for wavelet summaries is to estimate the answer to range sums. More general SPJ (select, project, join) queries can also be directly applied on relation summaries, to generate a summary of the resulting relation. This is made possible through an appropriately-

defined approximate query processing algebra that operates entirely in the domain of wavelet coefficients.

Recent research into wavelet representations has focused on error guarantees beyond L_2. These include L_1 (sum of errors) or L_∞ (maximum error), as well as relative-error versions of these measures. A fundamental choice here is whether to restrict the possible coefficient values to those arising under the basic wavelet transform, or to allow other (unrestricted) coefficient values, specifically chosen to reduce the target error metric. The construction of such (restricted or unrestricted) wavelet synopses optimized for non-L_2 error metrics is a challenging problem.

Sketches

Sketch techniques have undergone extensive development over the past few years. They are especially appropriate for streaming data, in which large quantities of data flow by and the sketch summary must continually be updated quickly and compactly. Sketches, as presented here, are designed so that the update caused by each new piece of data is largely independent of the current state of the summary. This design choice makes them faster to process, and also easy to parallelize.

"Frequency based sketches" are concerned with summarizing the observed frequency distribution of a data set. From these sketches, accurate estimations of individual frequencies can be extracted. This leads to algorithms for finding approximate "heavy hitters"—items that account for a large fraction of the frequency mass—and quantiles such as the median and its generalizations. The same sketches can also be also used to estimate the sizes of (equi)joins between relations, self-join sizes and range queries. Such sketch summaries can be used as primitives within more complex mining operations, and to extract wavelet and histogram representations of streaming data.

A different style of sketch construction leads to sketches for "distinct-value" queries that count the number of distinct values in a given multiset. As mentioned above, using a sample to estimate the answer to a COUNT DISTINCT query may give highly inaccurate results. In contract, sketching methods that make a pass over the entire data set can provide guaranteed accuracy. Once built, these sketches estimate not only the cardinality of a given attribute or combination of attributes, but also the cardinality of various

operations performed on them, such as set operations (union and difference), and selections based on arbitrary predicates.

In the final chapter, we compare the different synopsis methods. We also discuss the use of AQP within research systems, and discuss challenges and future directions.

In our discussion, we often use terminology and examples that arise in classical database systems, such as SQL queries over relational databases. These artifacts partially reflect the original context of the results that we survey, and provide a convenient vocabulary for the various data and access models that are relevant to AQP. We emphasize that the techniques discussed here can be applied much more generally. Indeed, one of the key motivations behind this survey is the hope that these techniques—and their extensions—will become a fundamental component of tomorrow's information management systems.

2

Sampling

2.1 Introduction

The use of random samples as synopses for approximate query processing has a twenty year history in the database research literature, with the earliest major-venue database sampling paper being published in 1984 [252]. This arguably makes sampling the longest studied method for approximate query processing. Sampling as a research topic in probability theory and statistics has an even longer history. Laplace and others famously used sampling to estimate the population of the French empire in the 18th century [28]. The study of *survey sampling*—sampling from a finite population such as a database—blossomed and matured as a sub-field of statistics in the first half of the 20th century, due most notably to the pioneering work of Jerzy Neyman. See Hansen's 1987 retrospective on survey sampling [160] for an interesting historical perspective on the field of survey sampling from a statistical point of view.

With such a long history, the breadth and variety of the sampling-based estimation methodologies that are applicable to approximate query processing are staggering. Given the variety of the work, this chapter will not attempt to cover and summarize most or even a majority fraction of the work from the

research literature in depth. Rather, the goal of the chapter is to serve as a tutorial on the application of sampling to approximate query processing in an information-management environment; references to papers in the research literature will be given as appropriate.

The basic topics covered by the chapter, in order, are as follows:

(1) A basic definition and simple examples of sampling for approximate query processing.
(2) A treatment of the advantages and disadvantages of approximate query processing using sampling.
(3) An introduction to the basic probability and statistics concepts that underlie data sampling, including example derivations of bias and variance, as well as a catalog of the different types of sampling schemes that are applicable to approximate query processing.
(4) A discussion of how sampling can be applied to estimate the answer to most of the common SQL query constructs.
(5) A discussion of how random samples can actually be drawn from large data sets that are stored on disk, and how sampling in this context differs from traditional survey sampling.
(6) A short discussion of online estimation via sampling.

2.2 Some Simple Examples

The basic idea behind sampling is quite simple. Given a data set—which we sometimes call a "population"—and a query with an unknown (possibly multi-attribute) numerical result that we wish to estimate, a small number of elements are first selected at random from the data set. Then a few statistics are computed over the sample, such as the sample mean and variance. Finally, these statistics are used to estimate the value of the query result and provide bounds on the accuracy of the estimate.

For an example of the process, imagine that we wish to answer the query

```
SELECT SUM (R.a)
FROM R
```

and that data set `R` comprises the ten different `R.a` values

$$\langle 3, 4, 5, 6, 9, 10, 12, 13, 15, 19 \rangle.$$

A sample of this data of size $n = 5$ can be obtained by simulating the rolling of a ten-sided die five times using an appropriate *pseudorandom number generator* (PRNG). A PRNG is an algorithm which takes an input a *seed*—that is, a string of bits—and then performs a number of numerical operations on the seed to obtain a new, seemingly random bit string that has no obvious statistical correlation with the input string. By `div`-ing or `mod`-ing the output string, a number with the appropriate range can be obtained—in our case, from 1 to 10. See [119] for a discussion of pseudorandom number generation).

In our example sampling process, rolling the number j on the ith trial implies that the ith item chosen at random from the population is the jth item in the population. This straightforward sampling scheme is known as *simple random sampling with replacement* because the same element may be sampled multiple times, so conceptually, an element is returned to the population for possible re-selection whenever it is selected. Say that we roll the die five times and obtain the following sequence of numbers:

$$\langle 6,3,5,3,9 \rangle$$

Then the associated sample of `R.a` values is:

$$\langle 10,5,9,5,15 \rangle$$

Using the resulting sample, it is then quite easy to estimate the answer to the `SUM` query. We compute the sum of the sample values, which is 44, and then scale up this sum by a factor of 2, to compensate for the fact that we have only seen roughly $5/10 = 1/2$ of the values in the population. (We say "roughly" because we are sampling with replacement.)

There is another, very useful way to derive the foregoing estimator. Denote by t_i the value of the ith item in the population and by X_i the random variable that represents the number of times that this ith item appears in the sample. For instance, $X_6 = 1$ and $X_3 = 2$ in our example since the sixth item ($t_6 = 10$) appears once and the third item ($t_3 = 5$) appears twice. The *expected value* of X_i is defined as $E[X_i] = \sum_{j=0}^{5} j \times Pr[X_i = j]$, and has the following interpretation: if we drew many samples of size 5 and recorded the frequency of item i in each sample, then, with probability 1, the average of these item-i frequencies would approach the number $E[X_i]$ as the number of samples

increased. That is, $E[X_i]$ represents the value of X_i we expect to see "on average." Now consider the estimator

$$Y = \sum_{i=1}^{N} \frac{X_i t_i}{E[X_i]} = \sum_{i \in \text{sample}} \frac{X_i t_i}{E[X_i]}, \tag{2.1}$$

where $N = 10$ is the population size and "sample" denotes the set of distinct items appearing in the sample (the set comprising items 6, 3, 5, and 9 in our example). The second equality holds because $X_i = 0$ for any item i that does not appear in the sample, so that such an item does not contribute to the sum. Because $E[X+Y] = E[X] + E[Y]$ and $E[cX] = cE[X]$ for any random variables X, Y and constant c, we have

$$E[Y] = E\left[\sum_{i=1}^{N} \frac{X_i t_i}{E[X_i]}\right] = \sum_{i=1}^{N} E\left[\frac{X_i t_i}{E[X_i]}\right] = \sum_{i=1}^{N} \frac{E[X_i] t_i}{E[X_i]} = \sum_{i=1}^{N} t_i.$$

Observe that the rightmost term is the true query answer Q, and so the estimator Y is *unbiased* in that $E[Y] = Q$. That is, Y is equal to the true query result on average, which is a desirable property of a sampling-based estimator. The estimator Y is an example of a *Horvitz-Thompson* (HT) estimator (see Sarndal, Section 2.8 [268]; the Horvitz-Thomson estimator was first described in a 1952 paper [171]). To see that Y corresponds to the first "scale-up" estimator that we described, note that at each sampling step, the ith item is included with probability $p = 1/10$. Since there are $n = 5$ sampling steps, it is intuitive[1] that the expected number of times that item i appears in the sample is $E[X_i] = np = 5/10 = 1/2$. Since $E[X_i]$ is the same for each item i, we can see that Y is computed by simply multiplying the sum $\sum_{i=1}^{N} X_i t_i$ by $1/E[X_i] = 2$. As discussed previously, this latter sum is simply the sum over all of the sample items, so that Y indeed corresponds to the scale-up estimator.

For many sampling schemes, such as simple random sampling without replacement, an item can appear at most once in a sample ($X_i = 0$ or 1). In this case $E[X_i] = 1 \times Pr[X_i = 1] = p_i$, where p_i is the probability that item i is included in the sample. Thus, an HT estimator of a sum is typically expressed as the sum of the item values in the sample, each divided by the item's prob-

[1] Technically, X_i has a binomial distribution, so that $Pr[X_i = j] = \binom{n}{j} p^j (1-p)^{n-j}$, and a standard calculation shows that $E[X_i] = np$.

ability of inclusion:

$$Y = \sum_{i \in \text{sample}} \frac{t_i}{p_i}. \tag{2.2}$$

Sums of functions of item values, such as $\sum_{i=1}^{N} t_i^2$ can be estimated in an analogous manner, e.g., $Y = \sum_{i \in \text{sample}} t_i^2 / p_i$. An unbiased sampling-based estimator of the average value in the data is Y/N, that is, we divide the HT estimator for the sum by the number of items in the data. As discussed in the sequel, other quantities of interest can be expressed as functions of one or more sums, and hence can be estimated (in an approximately unbiased manner) as corresponding functions of HT estimators of the sums. Thus the HT sampling and estimation framework is quite general in that, for many sampling schemes, many estimators for sum-related queries can be represented as an HT estimator or a variant of an HT estimator.

Error bounds (usually probabilistic in nature) for our `SUM`-query estimator Y defined in (2.1) can be obtained in many ways; see Section 2.4.2 for an in-depth discussion. One classical approach to error bounds is via the *Central Limit Theorem* (CLT) [96]. First we compute the sample variance of the sampled item values $\langle 10, 5, 9, 5, 15 \rangle$, which is

$$\frac{1}{4}\left((10-8.8)^2 + (5-8.8)^2 + (9-8.8)^2 + (5-8.8)^2 + (15-8.8)^2\right) = 17.2,$$

since the average of the five sampled item values is 8.8. (We divide by $n-1=4$ rather than $n=5$ to compensate for sampling bias.) This serves as an estimate for the variance of the population. According to the CLT, the quantity Y/N (which estimates the population mean) will be approximately normally distributed with a variance of (approximately) $17.2/5$, which implies that Y itself is approximately normally distributed, with a variance of $17.2 \times (100/5) = 344$, where 5 is the size of the sample and 100 is the square of the size of the dataset. This is only an approximation of the variance of Y because 17.2 is only an approximation of the population variance. Furthermore, slightly more than 95% of the mass of the normal distribution is within two standard deviations of the mean of the distribution (where the "standard deviation" is the square root of the variance). Then, since $\sqrt{344} = 18.55$, there is (approximately) a 95% chance that our estimate is within ± 37.10 of the correct answer. This range can be used to provide an end user with some idea of the accuracy of the sampling-based estimate.

As another example of an HT estimator, this time for a slightly more complicated query, suppose that we wish to estimate the sum $\texttt{R.a} + 4 \times \texttt{S.b}$ over a cross product of two data sets R and S. We can first independently sample 10% of the tuples from R and 10% of the samples S, and then take a cross product of the two samples, thereby obtaining a random sample U of all of the items in $\texttt{R} \times \texttt{S}$. The probability that a given element $(\texttt{r}, \texttt{s}) \in \texttt{R} \times \texttt{S}$ is included in the sample U is simply $10\% \times 10\% = 1\%$, which is the joint probability that r is sampled from R and s is sampled from S. The HT estimator of the query result is then $Y = (1/0.01) \sum_{(\texttt{r},\texttt{s}) \in U} (\texttt{r.a} + 4 \times \texttt{s.b})$. That is, we simply compute the summation over the sample and then scale up the result by a factor of 100 to compensate for the sampling.[2]

The above examples illustrate perhaps the simplest applications of random sampling to approximate query processing; more complex examples are given in the sequel. Unless specified otherwise, we will focus on the use of sampling for approximate answering of aggregation queries, that is, queries in which relational operations are applied to a set of base relations to produce an intermediate relation, and then the tuples of this relation are fed into a function—such as SUM, AVERAGE, and so on—that computes a number (or small set of numbers) that comprise the final query result. Sampling may also be applied to a query that returns a set of tuples; the goal here is to produce a representative sample of these tuples. Such samples are useful for purposes of auditing, exploratory data analysis, statistical quality control, and privacy enforcement [243].

2.3 Advantages and Drawbacks of Sampling

Now that we have illustrated the use of sampling with a simple example, we will next discuss various scenarios in which sampling is, or is not, useful. Advantages of sampling include:

(1) Simplicity. Conceptually, it is very simple to understand the idea of drawing items at random from a dataset, then scaling up the result of a query over the sample to guess the result of applying

[2] Note that the samples obtained by this process are not, in fact independent samples from the cross product, which introduces some complications in the analysis of the accuracy of the estimator Y. See Section 2.6.1 for more details.

the query to the whole dataset.

(2) Pervasiveness. Sampling is widely supported by database systems, and support for sampling is part of the current SQL standard (SQL-2008).

(3) Extensive theory. Almost 100 years of prior research in survey sampling can be applied directly to sampling massive data. Classic results such as the CLT can be used to assess the error of estimates obtained via sampling, to determine how many samples to obtain to achieve a desired accuracy, and to develop special-purpose sampling strategies that obtain high accuracy for difficult queries.

(4) Immediacy. Sampling is unique in that it need not rely on a pre-constructed model that has been built offline, before query processing has begun. Since (by definition) a sample contains a small subset of the data—perhaps only a few hundred tuples—it can often be constructed after a query has been issued, without incurring a delay that is long enough to be perceptible to an end-user.

(5) Adaptivity. Unlike "one-and-done" approximation methods that rely on a pre-constructed model, sampling permits *online* approximation: if a small sample does not provide enough accuracy for a specific query, then more tuples can be sampled to provide for more accuracy while the user waits, in an online fashion. In contrast, if a data structure such as an AMS sketch [7] does not provide enough accuracy, the user has no option for incrementally improving the synopsis.

(6) Flexibility. A "sample" is a very general-purpose data structure and as such, the same sample can be used to answer a wide variety of arbitrary queries. The generality results from the fact that sampling commutes with common query operations such as selection, projection, and grouping. In the context of aggregation queries, sampling also commutes, in an appropriate sense, with the cross product and join operations; see Section 2.6.1. That is, it is often possible to replace the input relations to an aggregation query by samples from those relations and use the query result to obtain well behaved estimators of the true query result. ("Well behaved" estimators are computable from the informa-

tion at hand, are unbiased or approximately unbiased, improve in accuracy as the sample size is increased, and come with assessments of estimator accuracy.) This is true no matter what form the underlying boolean selection or join predicate takes—equi-joins, greater-than predicates, not-equal-to-predicates, arbitrary and complicated CNF expressions—all commute with sampling and so all work seamlessly with sampling. It is also true whether the underlying attributes are numerical or categorical.

(7) Insensitivity to dimension. The accuracy of sampling-based estimates is usually independent of the number of attributes in the data. That is, unlike other methods such as wavelets and histograms, there are no combinatorial difficulties induced by the "dimensionality" of the data.

(8) Ease of implementation. Because sampling commutes with many of the common query operations, it is possible to use a database engine *itself* to evaluate a query over a sample. That is, a database query can be sped up by first sampling the database, then feeding the sample into the database engine where the *original query is run without modification*, and then adjusting the final answer—for example, by scaling it upwards—to take the sampling into account. This means that sampling can be used as an approximation method in a database environment with very little modification to the source code of the database system. This is precisely the idea, for example, behind the AQUA project which proposed storing a sample of a database inside another database instance, and using the same database engine to process incoming queries using the sample [3].

However, sampling has its own particular drawbacks as well. Most notably:

(1) Because sampling relies on having a reasonable chance of selecting some of the data objects that are relevant for answering a query, sampling can be unsuitable for approximating the answer to queries that depend only upon a few tuples from the dataset (that is, those queries that are highly selective). For example, if only ten

tuples out of one million contribute to a query answer, then a 1% sample of the data is unlikely to select any of them, and the resulting estimate can be poor. This is one reason why, despite all of sampling's benefits, statistical synopses such as histograms are far more widely used as tools for selectivity estimation in commercial database systems.

(2) The fact that samples are generally used for approximate query processing by simply applying an incoming query to the sample is sometimes seen as a drawback, and not as a benefit. Evaluating a query over a large, 5% sample of a database may take 5% of the time that it takes to evaluate the query over the entire dataset. A 20× speedup may be significant, but other, more compact synopses such as histograms can provide much faster estimates. This is another reason that sampling is not widely used for selectivity estimation—it is widely thought that providing an estimate via a sample is much more expensive than estimation via other methods.

(3) Sampling is generally sensitive to skew and to outliers. Reconsider our example from the previous section. If our dataset instead contained the ten items:

$$\langle 3,4,5,6,9,10,12,13,15,10^{99}\rangle$$

Then any estimate for the final sum will be quite inaccurate. Indeed, a sample that happens to miss 10^{99} will radically underestimate the final sum, and a $(p \times 100)\%$ sample that happens to hit 10^{99} will overestimate the sum by approximately a factor of $1/p$.

(4) There are important classes of aggregation queries for which the basic Horvitz-Thompson setup of Section 2.2 breaks down, in which case sampling-based estimation usually becomes very challenging. One way that an HT estimator can run into trouble is when the inclusion probability p_i of an item into the sample—see equation (2.2)—depends on the (unknown) distribution of values in the data population. In this case, the inclusion probabilities are unknown at estimation time so the HT formula cannot be used directly. (More generally, the quantities $E[X_i]$ as in equation (2.1) may be unknown.) For example, consider the problem of evaluat-

ing the sum of R.a over relation R, subject to the constraint that we wish to only consider those tuples whose R.b value does not appear anywhere in attribute S.c from relation S. This is effectively a NOT IN query of the from:

```
SELECT SUM (R.a)
FROM R
WHERE R.b NOT IN (SELECT S.c FROM S)
```

Suppose that one independently samples both R and S (say, without replacement) and then attempts to estimate the query result using an HT estimator in a manner similar to the final example in Section 2.2. The probability that a given tuple r is included in U—the sample set of items from R whose R.a values are to be summed—depends on whether r fails to find a match in the sample from S, and this latter probability depends on the frequency distribution of attribute S.c, which is unknown. Thus it is difficult to characterize the sample set U. Indeed, some r from R can actually appear in the result if one samples R and S first and then runs the query over the samples, even when r does not appear in the true result set—simply because r has no mate in the sample from S does not mean that it has no mate in all of S. Thus, it is hard to use sampling to estimate the answer to this query. NOT IN queries are not the only example of this. Others that are difficult are those having the SQL DISTINCT keyword (in other words, duplicate removal), as well as those with the IN or EXISTS keywords, those containing antijoins and outer joins, and most queries with set-based (as opposed to bag-based) semantics. Another way that the HT approach can fail is when the aggregate of interest cannot be expressed in terms of sums or functions of sums. MAX and MIN queries are typical examples. Sections 2.6.2 and 2.6.3 contain some current research results related to these hard sampling and estimation problems.

2.4 Mathematical Essentials of Sampling

2.4.1 A Mathematical Model For Sampling

Before covering specific work relevant to sampling for approximate query processing, it is instructive to describe some of the basic statistical principles underlying sampling, introduce the reader to some of the common terminologies, and demonstrate how those principles might apply to the simple sampling example from the previous section. Virtually all of the ideas in the database literature and in the statistics literature are derived directly from the basic ideas introduced here.

As described previously, a *sample* is a set of data elements selected via some random process. The sampling process can be modeled statistically as follows. As in Section 2.2, denote by t_j the jth item or tuple in the dataset and by X_j the random variable that controls the number of copies of the item that are included in the sample. Intuitively, we view a random variable as a mathematical object that encapsulates the idea of "random chance". It can be thought of as a non-deterministic machine, where "pressing a button" on the machine causes a random value to be produced; this is known as a "trial" over the variable. In our case, a trial over X_j produces a non-negative integer value x_j. The behavior of the various X_j's defines the behavior of the underlying sampling process. If x_j is zero, then t_j is not sampled. Otherwise, t_j is sampled x_j times. By changing the sampling process (with replacement, without replacement, biased, unbiased, etc.) we change the statistical properties of the various X_j's, and change the statistical properties of the sampling process. For example, if so-called "Bernoulli" sampling is desired (see Section 2.5.3), then we use a pseudorandom number generator to simulate flipping a biased coin once for each and every data item. The item is included in the sample if the coin comes up heads; otherwise, it is not. In this case, each X_j takes the value one if the coin comes up heads, and takes the value zero otherwise.

Given the various X_j's which control the sampling process, the next thing that is needed to apply sampling to the problem of approximate query processing is an *estimator* that can be used to guess the answer to the query. An estimator Y can be thought of as nothing more than a function $\mathscr{F}$ that has been parameterized on all of the data elements and all of the random variables

that control the sampling process:

$$Y = \mathscr{F}\big(\langle (t_1, X_1), (t_2, X_2), \ldots, (t_n, X_n)\rangle\big)$$

In general, $\mathscr{F}$ will satisfy the constraint that:

$$\mathscr{F}\big(\langle \ldots, (t_{j-1}, X_{j-1}), (t_j, 0), (t_{j+1}, X_{j+1}), \ldots\rangle\big) = \mathscr{F}\big(\langle \ldots, (t_{j-1}, X_{j-1}), (t_{j+1}, X_{j+1}), \ldots\rangle\big)$$

In other words, $\mathscr{F}$ cannot "look at" the tuple t_j if the associated $x_j = 0$. However, this restriction is not absolute; some sampling-based estimators described in the database literature make use of summary statistics over the dataset, such as the pre-computed average value of each tuple attribute [194]. In this case, $\mathscr{F}$ may have access to statistics that consider every t_j in the dataset.

Since applying a function such as $\mathscr{F}$ to a set of random variables results in a new random variable, Y is itself a random variable. Performing a trial over this random variable (that is, performing the sampling process to obtain each x_j and then applying $\mathscr{F}$ to the result) gives us an actual *estimate* for the answer to the query. It is this estimate that is returned to the user. Often, the estimate contains a single numerical value. However, in the general case it may contain a vector of values; for example, Y may be defined so that it estimates the result of a `GROUP BY` query.

2.4.2 Bias and Variance: Quantifying Accuracy

In general, the utility or accuracy of Y is quantified by determining Y's *bias* and *variance*. The bias of Y is defined as:

$$bias(Y) = E[Y] - Q$$

where $E[Y]$ is the expected value of Y and Q is the actual query result. This determines how far away, on average, Y will be from Q. The variance of Y is:

$$\sigma^2(Y) = E[(Y - E[Y])^2] = E[Y^2] - E^2[Y]$$

which measures the spread of Y around its average value. High spread is bad, since it means that the estimator has a great degree of variability and often falls far from its expected value. If Y has low bias and low variance, then Y is (usually) a high-quality estimator.

It is instructive to make these concepts a little more concrete by considering the application of these principles to an actual sampling process and estimator. Consider the case of simple random sampling with replacement, as described in Section 2.2. Without loss of generality and for simplicity of notation, here and in most of our discussion of sampling we use the notation t_j to denote the jth tuple in the dataset and we assume that t_j is a real number. While this may seem restrictive, it is really not. t_j can in fact be the result of any mathematical or logical expression over actual data items. For example, consider the query:

```
SELECT SUM(r.extendedprice * (1.0 - r.tax))
FROM R as r
WHERE l.suppkey = 1234
```

In this case, we can set t_j to be `r.extendedprice * (1.0 - r.tax)` if `r.suppkey = 1234` where `r` is the jth tuple in `R`; otherwise, if `r.suppkey <> 1234` then t_j is set to zero.

Since we assume that t_j encapsulates any underlying selection predicate as well as the function to be aggregated, the answer to a single-relation `SUM` query can always be written as:

$$Q = \sum_j t_j$$

Now imagine that we have a particular dataset instance where $\langle t_1, t_2, \ldots, t_{10} \rangle$ take the values $\langle 3, 4, \ldots, 19 \rangle$, as in our example from Section 2.2. For our dataset of size 10 and a sample of size 5, we can formalize the estimator from Section 2.2 as:

$$Y = \mathscr{F}\big(\langle (t_1, X_1), \ldots, (t_{10}, X_{10}) \rangle\big) = \frac{10}{5} \sum_{j=1}^{10} t_j X_j$$

As discussed in Section 2.2, each X_j is a binomially-distributed random variable with parameters $p = 1/10$ and $n = 5$ under simple random sampling with replacement.[3] Thus $E[X_j] = np = 1/2$ and hence $E[Y] = Q$, so that Y is unbiased. Finally, note that each of the five samples is statistically independent.

[3]Note that while each X_j is binomial, it must be the case that $\sum_j X_j$ is 5, since $n = 5$ samples are taken overall. Thus, the various X_j's are marginally binomial, but not independent. This correlation does not affect $E[Y]$, but must be taken into account when computing $E[Y^2]$, as we discuss subsequently.

We now consider the variance of this particular estimator. In this case, since $E[Y] = Q$, we have $E^2[Y] = Q^2$. However, in order to compute the variance, it is also necessary to know $E[Y^2]$. This can be derived via algebraic manipulation:

$$E[Y^2] = E\left[\left(2\sum_j t_j X_j)\right)^2\right] = E\left[4\sum_i \sum_j t_i t_j X_i X_j\right]$$
$$= E\left[4\sum_i t_i^2 X_i^2 + 8\sum_{i<j} t_i t_j X_i X_j\right] = 4\sum_i (t_i^2 E[X_i^2]) + 8\sum_{i<j} t_i t_j E[X_i X_j]$$

This leaves us with two summations. In the first summation, the term $E[X_i^2]$ must be evaluated. This is simply the second moment of the binomial distribution, and according to any standard textbook on discrete distributions, this value is $np + n(n-1)p^2 = \frac{1}{2} + \frac{1}{5} = \frac{7}{10}$. Computing $E[X_i X_j]$ is more non-standard due to the constraint that the various X_j's must sum to n, but in this case its value can be evaluated as $n(n-1)p^2 = \frac{1}{5}$.[4] Plugging this into the above equation, we have:

$$E[Y^2] = \frac{14}{5}\sum_i t_i^2 + \frac{8}{5}\sum_{i<j} t_i t_j$$

And so the variance of Y, denoted as $\sigma^2(Y)$, is:

$$\sigma^2(Y) = \frac{14}{5}\sum_i t_i^2 + \frac{8}{5}\sum_{i<j} t_i t_j - Q^2$$

Plugging in the actual values for our dataset, we have:

$$\sigma^2(Y) = \frac{14}{5} \times 1166 + \frac{8}{5} \times 4025 - 9216 = 488.8$$

[4]The intuition behind the formula $E[X_i X_j] = n(n-1)p^2$ is as follows. We can view X_i (and X_j) as the number of heads over a sequence of n coin flips; a "heads" on the kth flip in the sequence associated with X_i means that the kth tuple sampled was t_i. Let $X_{i,k}$ be a random variable controlling the output of the kth flip, so that $X_i = \sum_k X_{i,k}$. Then $E[X_i X_j] = E[\sum_{k1,k2} X_{i,k1} X_{j,k2}]$. Since we can push the expectation into the sum, we have $E[X_i X_j] = \sum_{k1,k2} E[X_{i,k1} X_{j,k2}]$. Thus, we need only consider how to evaluate $E[X_{i,k1} X_{j,k2}]$. If $k1 = k2$, then $X_{i,k1} X_{j,k2}$ is always zero; that is, if the two coin flips are both the kth flip in the sequence, then they can never both come up heads—this would imply that the kth sample selected both t_i and t_j, which is not possible. If, on the other hand, the two coin flips are not both the same flip in the sequence, the probability that both are heads is p^2, since the two flips are independent. In this case, $E[X_{i,k1} X_{j,k2}] = p^2$. Summing over all n^2 pairs of coin flips that contribute to $E[X_i X_j]$, we have $E[X_i X_j] = n^2 p^2 - np^2$, which is $n(n-1)p^2$.

The observant reader may note that 488.8 differs considerably from the value $17.2 \times \frac{100}{5} = 344$ that was computed via the CLT in the prior section. The reason for this is that 488.8 is the *actual* variance of our estimator Y; there are no approximations here. The value 344 from the prior section was obtained by using the sample itself to estimate the variance of the underlying population (since, in practice, the true population variance of 488.8 would be unknown to the user).

More generally, for a Horvitz-Thompson estimator Y as in equation (2.1), based on a sample of size N, let $\pi_i = E[X_i]$ and $\pi_{ij} = E[X_i X_j]$ for $1 \le i, j \le N$. Using the fact that $E[Y] = \sum_i t_i$ (because Y is unbiased), algebraic manipulations as before show that

$$\begin{aligned}\sigma^2(Y) = E[Y^2] - E^2[Y] &= E\Big[\Big(\sum_i \frac{X_i t_i}{\pi_i}\Big)^2\Big] - \Big(\sum_i t_i\Big)^2 \\ &= E\Big[\sum_i \sum_j \frac{X_i t_i}{\pi_i}\frac{X_j t_j}{\pi_j}\Big] - \sum_i \sum_j t_i t_j \\ &= \sum_i \sum_j \frac{\pi_{ij} t_i t_j}{\pi_i \pi_j} - \sum_i \sum_j t_i t_j = \sum_i \sum_j \Big(\frac{\pi_{ij}}{\pi_i \pi_j} - 1\Big) t_i t_j.\end{aligned} \tag{2.3}$$

In practice, we need to estimate $\sigma^2(Y)$ from the sample at hand. Observe that $\sigma^2(Y)$ is expressed as a sum (over the cross-product of the dataset with itself), so that we can use the HT trick yet again to derive an unbiased estimator of $\sigma^2(Y)$:

$$\hat{\sigma}^2(Y) = \sum_i \sum_j \frac{X_i X_j}{\pi_{ij}} \Big(\frac{\pi_{ij}}{\pi_i \pi_j} - 1\Big) t_i t_j = \sum_{i,j \in \text{sample}} \frac{X_i X_j}{\pi_{ij}} \Big(\frac{\pi_{ij}}{\pi_i \pi_j} - 1\Big) t_i t_j. \tag{2.4}$$

Plugging the values for our dataset into these formulas yields the same answers as before: $\sigma^2(Y) = 488.8$ and $\hat{\sigma}^2(Y) = 344$. The variance formulas, however, also hold for other sampling schemes besides simple random sampling with replacement. If the sampling scheme is such that an item can appear at most once in the sample—so that the HT estimator has the form given in equation (2.2)—then $\pi_i = p_i$ and $\pi_{ij} = p_{ij}$, where p_i is the probability that data item i is included in the sample and p_{ij} is the probability that items i and j are both included in the sample. Moreover, $E[X_i^2] = E[X_i]$ since $X_i = 0$ or 1, so that the above formulas takes on the special forms

$$\sigma^2(Y) = \sum_i \Big(\frac{1}{p_i} - 1\Big) t_i^2 + 2 \sum_{i<j} \Big(\frac{p_{ij}}{p_i p_j} - 1\Big) t_i t_j$$

and

$$\hat{\sigma}^2(Y) = \sum_{i \in \text{sample}} \frac{1}{p_i}\Big(\frac{1}{p_i} - 1\Big)t_i^2 + 2 \sum_{\substack{i,j \in \text{sample} \\ i<j}} \frac{1}{p_{ij}}\Big(\frac{p_{ij}}{p_i p_j} - 1\Big)t_i t_j.$$

The key point here is that, whereas $E[Y]$ only depends on the individual inclusion probabilities p_i, the variance of Y depends on the joint inclusion probabilities p_{ij}, which can be challenging to calculate. This additional complexity explains why the variance of an estimator can be much harder to estimate than the expected value. It also explains why two different sampling schemes can have the same marginal inclusion probabilities and hence can both lead to unbiased HT estimators, but the variance properties of the estimators can differ dramatically if the joint inclusion probabilities differ.

2.4.3 From Bias and Variance To Accuracy Guarantees

Although bias and variance are important statistics that are very useful for describing the error of a sampling-based estimate, most users are likely to be more comfortable with probabilistic *confidence bounds* (see Sarndal et al., Section 2.11 [268]). A confidence bound is a probabilistic guarantee of the form:

"There is a $p \times 100\%$ chance that the true answer to the query is within the range l to h."

Central Limit Theorem. There are many ways to provide for confidence bounds. The most common method is to assume that the error on an unbiased, sampling-based estimator Y is normally distributed. That is, we assume that Y can be modeled as:

$$Y \approx Q + \mathcal{N}(0, \sigma^2(Y))$$

In this equation, $\mathcal{N} = \mathcal{N}(0, \sigma^2(Y))$ is a normally-distributed random variable with mean zero and a variance of $\sigma^2(Y)$. Then, if we choose numbers *lo* and *hi* so that:

$$p = \int_{lo}^{hi} f_{\mathcal{N}}(x)dx$$

where $f_{\mathcal{N}}$ is the probability density function of $\mathcal{N}$, we know that there is a $p \times 100\%$ chance that $lo \leq Q - Y \leq hi$. (Here we use the fact that $Q - Y = -\mathcal{N}$ has the same distribution as $\mathcal{N}$ because the normal distribution

is symmetric about the origin.) By algebraic manipulation, it then holds that if $l = Y + lo$ and $h = Y + hi$—here l and h are random variables—there is a $p \times 100\%$ chance that the random interval $[l,h]$ contains Q. For example, since

$$\int_{-1.98\sigma(Y)}^{1.98\sigma(Y)} f_{\mathcal{N}}(x)dx = 0.95,$$

if we assume normality of the error, then for unbiased Y we are justified in saying that there is around a 95% chance that Q is within $Y - 2\sigma(Y)$ to $Y + 2\sigma(Y)$, which is perhaps the most commonly-used confidence bound. Such a confidence bound is often inverted to arrive at the equivalent statement that "Y estimates Q to within $\pm 2\sigma(Y)$ with 95% probability." By approximating $\sigma^2(Y)$ with a sample-based estimate $\hat{\sigma}^2(Y)$ as in Sections 2.2 and 2.4.2, and then taking the square root to get an estimate $\hat{\sigma}(Y)$ of $\sigma(Y)$, we can roughly assess the accuracy of Y at estimation time (and potentially increase the number of samples if the accuracy is not deemed sufficient).

Although there is never any guarantee that the error of Y is normally distributed, the statistical justification for assuming normality is typically the CLT, which states that as the number of independent samples taken from a distribution approaches infinity, the observed difference between the mean of the distribution and the mean of the samples looks increasingly like a sample from a normally-distributed random variable. In the authors' own experience, for most of the estimators one would encounter in a data analysis environment, normality is a safe assumption. This seems to be true even when the samples are correlated (as they will be if sampling without replacement is used), and when the number of samples is not very, very large. The robustness of the normality assumption stems from statistical theory—see, for example, [22, 159]—which asserts that variants of the CLT hold in great generality, that is, under various weakenings of both the independence and identical-distribution assumptions. In our experience, normality is especially safe if the specified p does not exceed 95%; deviations from normality tend to be most pronounced in the tails of the error distribution. In Section 2.2, we used exactly the CLT bound to assert that the answer to our example query was 88 ± 37.10 with roughly 95% certainty.

Chebyshev Bounds. However, normality is never guaranteed. The authors have generally found that statisticians are more accepting of distributional

assumptions than are computer scientists, who tend to be more conservative. If one eschews distributional assumptions, then *distribution-free* bounds can be used instead. These are looser, but more comforting. One common bound is due to Chebyshev's inequality, which implies that for an unbiased estimator Y,

$$Pr[|Y - Q| \geq p^{-\frac{1}{2}}\sigma(Y)] \leq p$$

Thus, according to Chebyshev's inequality, there is a $p \times 100\%$ chance that Q is between $Y - p^{-\frac{1}{2}}\sigma(Y)$ and $Y + p^{-\frac{1}{2}}\sigma(Y)$. While such bounds are comforting, they are often much looser than CLT-based bounds. Had we applied Chebyshev bounds to our example from Section 2.2, we would have obtained a confidence interval of 88 ± 117.32.

Hoeffding Bounds. Other distribution-free bounds are Hoeffding bounds [170] and Chernoff bounds [158]. Whereas bounds based on Chebyshev's inequality assume that the variance of the underlying distribution is known, Hoeffding bounds are applicable when $Y = \frac{1}{n}\sum_i X_i$, where the value of X_i ranges from low_i to hi_i. In this case,

$$Pr[|Y - E[Y]| \geq d] \leq 2\exp(-\frac{2d^2n^2}{\sum_i(hi_i - low_i)^2})$$

Hoeffding bounds could apply to our example from Section 2.2 as follows. Recall that we sampled values $\langle 10, 5, 9, 5, 15\rangle$ which we can multiply by $\frac{1}{p}$ to obtain the sequence $\langle 100, 50, 90, 50, 150\rangle$; the mean of this sequence is an unbiased estimate for Q. If we assume that 50 and 150 represent reasonable low and high bounds on the possible numbers we could obtain via this process, we can approximate $\sum_i(hi_i - low_i)^2$ by $n(150-50)^2 = 50,000$. We then solve the equation

$$0.05 = 2\exp(-\frac{2d^2n^2}{50,000})$$

for d, giving $d = 192.06$; this implies that 88 ± 192.06 is a 95% confidence interval for the query answer. Note that while this interval is quite wide, it would have been even wider had we used the correct low and high values from the dataset, which are often unknown at query time. The excessive width of Hoeffding bounds is their main drawback; in addition to being distribution-free, their main benefit is the fact that a variance estimate was not required.

Obtaining a variance estimate can be quite challenging for some sampling problems, and Hoeffding bounds are attractive in such circumstances.

Chernoff Bounds. Chernoff bounds are closely related to Hoeffding bounds, but apply to bounding $Y = \frac{1}{n}\sum_i X_i$ when each X_i can only take the value zero or one. Thus, they are not used as often as the other bounds for direct construction of confidence bounds for sampling-based estimators, but they do appear as important tools for constructing accuracy proofs for various dataset approximation methodologies, sampling-based and otherwise.

Biased Vs. Unbiased Estimates. The preceding discussion has assumed throughout that Y is unbiased. In practice, not all good estimators are unbiased. Biased estimators are sometimes more accurate than unbiased estimators, and often easier to construct. For example, consider the problem of estimating the size of the join of a relation with itself, using a sample of the relation. An example of such a query is obtained via a slight modification of the SQL we have been using as a running example:

```
SELECT COUNT (*)
FROM R as r1, R as r2
WHERE r1.a BETWEEN r2.a - 3 AND r2.a + 3
```

Imagine that we draw a size $n = 5$ with-replacement sample of `R`, then use the sample as a proxy for `R`. We join the sample with itself, and scale the result by $\frac{1}{n^2p^2} = 4$. If we had used two independent samples of `R` rather than a single one, we would obtain an unbiased estimate for Q via this process (see Section 2.6.1). But joining the sample with itself produces bias. To show this, we begin with the expectation of the resulting estimator Y:

$$E[Y] = E\left[4\sum_{j,k} I(t_j.\texttt{a BETWEEN } t_k.\texttt{a} - 3 \texttt{ AND } t_k.\texttt{a} + 3)X_jX_k\right]$$

where I is the indicator function returning 1 if the boolean argument evaluates to `true` and 0 otherwise. We can simplify this as follows, using $I(t_j.\texttt{a}, t_k.\texttt{a})$

as shorthand for $I(t_j.\texttt{a BETWEEN } t_k.\texttt{a}-3 \texttt{ AND } t_k.\texttt{a}+3)$:

$$\begin{aligned}
E[Y] &= E\Big[4\sum_j\sum_k I(t_j.\texttt{a}, t_k.\texttt{a})\Big] \\
&= E\Big[4\sum_j I(t_j.\texttt{a}, t_j.\texttt{a})X_j^2 + 4\times 2\sum_{j<k} I(t_j.\texttt{a}, t_k.\texttt{a})X_jX_k\Big] \\
&= 4\sum_j I(t_j.\texttt{a}, t_j.\texttt{a})E[X_j^2] + 4\times 2\sum_{j<k} I(t_j.\texttt{a}, t_k.\texttt{a})E[X_jX_k]
\end{aligned}$$

From Section 2.4.2, we know that $E[X_j^2] = \frac{7}{10}$, and $E[X_jX_k] = \frac{1}{5}$. Thus we have:

$$E[Y] = \frac{14}{5}\sum_j I(t_j.\texttt{a}, t_j.\texttt{a}) + \frac{8}{5}\sum_{j<k} I(t_j.\texttt{a}, t_k.\texttt{a})$$

Since $Q = \sum_j I(t_j.\texttt{a}, t_j.\texttt{a}) + 2\sum_{j<k} I(t_j.\texttt{a}, t_k.\texttt{a})$, $Q \neq E[Y]$ and Y is biased. Intuitively, the bias here results from the fact that this estimator over-emphasizes the importance of tuples that join with themselves.

If (as in this case) Y is biased, there are three obvious tactics to use. One is to ignore the bias. Many estimators exhibit bias that diminishes linearly (or faster) with increasing sample size. The term *asymptotically unbiased* is often used for such estimators. If the bias is significant, a second tactic is to estimate the bias as well as the query result, correct for the bias, and thus obtain an unbiased estimator. However, this can sometimes result in an estimator whose standard error—defined as $(bias^2(Y) + \sigma^2(Y))^{1/2}$—is actually greater than the original biased estimator. A third tactic is to simply accept the bias, then compute, estimate, or bound the bias to obtain an estimate for the standard error of the estimator. One can then use either CLT-based or Hoeffding-based confidence bounds, replacing $\sigma(Y)$ in the relevant formulas with the standard error.

Unfortunately, the standard deviation cannot be replaced by the standard error with impunity, because as the bias increases, the actual and user-specified coverage rates for the confidence bounds tend to diverge. A rule-of-thumb is that for $p \leq 0.95$, then it is generally safe to replace the standard deviation with the standard error as long as the ratio of $bias(Y)$ to $\sigma(Y)$ does not exceed 0.5; see Sarndal et. al [268], page 165.

2.5 Different Flavors of Sampling

In this section, we catalog the various sampling schemes applicable in a data analysis environment. For each scheme, we give a high-level description of the scheme, and also describe the standard unbiased estimator used along with the sampling process to provide an unbiased estimate for the query answer Q in the case of a single-relation `SUM` query. Many of the results follow directly from our results for HT estimators. The focus here is on the high-level statistical aspects of the various schemes; actual implementation details are deferred until Section 2.7.

2.5.1 Simple Random Sampling With Replacement

This is the classic form of random sampling, and is precisely the form of sampling used in the example estimation procedure detailed in Section 2.4.2. Logically, to draw a sample of size n from a relation `R` of size $|\mathtt{R}|$, the following two steps are undertaken, n times:

(1) Produce a random number r from 1 to $|\mathtt{R}|$, inclusive.
(2) Obtain the rth item from the dataset and add it to the sample.

For simple random sampling with replacement (SRSWR), the standard estimator for a single-relation `SUM` query is a simple generalization of the estimator from Section 2.4.2:

$$Y = \frac{|\mathtt{R}|}{n} \sum_j X_j t_j$$

Under SRSWR, it is possible to view the sampling process as performing a sequence of trials over tuple-valued random variables, where each trial produces a sampled tuple. Under this view, random variable is independent of the rest, and each is identically distributed (since each sampled tuples is drawn from the discrete distribution of all possible tuples). That is, the random variables controlling the sampling are "i.i.d." and the classical CLT will apply to any estimator that sums the sampled values, or that sums some function of the sampled values. In the case of the `SUM` query estimate given above, the variance of this estimator is simply

$$\sigma^2(Y) = \frac{|\mathtt{R}|^2 \sigma^2(\mathtt{R})}{n}$$

and the CLT applies. In this expression, $\sigma^2(\mathtt{R})$ is the variance of the t_j's in the underlying relation. As discussed above, in practice $\sigma^2(\mathtt{R})$ must be estimated, and the variance of the sampled data items is substituted for the variance of the actual relation.

SRSWR as a sampling method has several advantages. First, much of the statistical analysis is simpler for this kind of sampling because SRSWR is the only type of sampling where a sample of n tuples can be viewed as a sequence of trials over n "i.i.d." random variables. Since much of classical statistics—for example, the classical version of the CLT—applies most straightforwardly to such a case, analysis is typically easier.

A drawback of SRSWR is that for most estimation problems, without-replacement sampling has lower variance than with-replacement sampling for a fixed sampling size (see the next subsection below). This is especially the case as the sampling fraction grows large—greater than 5 or 10%. Indeed, most fixed-size, without-replacement estimators will have zero variance when the sample size $n = |\mathtt{R}|$, which is not the case under SRSWR.

2.5.2 Simple Random Sampling Without Replacement

In simple random sampling without replacement (SRSWoR), the sampling process is constrained so that it cannot draw the same data item more than once. Logically, to draw a sample of size n from relation $|\mathtt{R}|$, the following two steps are undertaken, until n samples have been obtained:

(1) Produce a random number r from 1 to $|\mathtt{R}|$, inclusive.
(2) If the rth item from the data has not been previously added to the sample, then obtain this item and add it to the sample.

For SRSWoR, the standard HT estimator for a single-relation `SUM` query is identical to the case of SRSWR:

$$Y = \frac{|\mathtt{R}|}{n} \sum_j X_j t_j$$

However, the variance changes slightly, and becomes:

$$\sigma^2(Y) = \frac{|\mathtt{R}|(|\mathtt{R}| - n)\sigma^2(\mathtt{R})}{n}$$

It should be clear from the above formula that SRSWoR generally provides for lower-variance estimates than SRSWR, due to the quantity n being subtracted from $|\mathtt{R}|$ in the variance's numerator. The intuition behind the reduced variance is that SRSWoR controls the number of times that item j is in the sample (it is always zero or one), whereas SRSWR allows item j to (possibly) be sampled numerous times, and hence sees fewer of R's tuples for a given sample size.

2.5.3 Bernoulli and Poisson Sampling

Bernoulli sampling (and its generalization, known as Poisson sampling) is quite a bit different from both SRSWR and SRSWoR. In Poisson sampling, a (possibly separate and unique) inclusion probability p_j is associated with *each tuple* in the dataset. In Bernoulli sampling, each p_j must be the same. Given all of the p_j's, each X_j is an independent Bernoulli (0/1) random variable where $Pr[X_j = 1] = p_j$. Since (unlike both SRSWR and SRSWoR) all of the various X_j's are independent, $Pr[X_i X_j = 1] = E[X_i X_j] = p_i p_j$. Thus, drawing a sample using Poisson sampling is equivalent to flipping $|\mathtt{R}|$ independent coins in sequence; a "heads" on the jth coin flip implies the that jth tuple from the dataset is included in the sample.

Specifically, to draw a Poisson sample from a dataset, the following two steps are undertaken for each data item. For the jth tuple:

(1) Generate a random number r from 0 to 1.
(2) If r is less than p_j, include the tuple in the sample.

Note that the sample size is random. Using equation (2.2) and the results in Section 2.4.2, we see that, under Poisson sampling, the HT estimator Y for a single-relation SUM query has the simple form

$$Y = \sum_{i \in \text{sample}} \frac{t_i}{p_i}$$

and, moreover,

$$\sigma^2(Y) = \sum_i (\frac{1}{p_i} - 1) t_i^2$$

and

$$\hat{\sigma}^2(Y) = \sum_{i \in \text{sample}} \frac{1}{p_i} (\frac{1}{p_i} - 1) t_i^2.$$

Since the form of this variance is quite different than for SRSWR and SRSWoR, it can be challenging to directly compare Poisson sampling with the other two options. In practice, Poisson sampling can either be marginally worse than the other options or perhaps far better, depending upon whether the Poisson sample has been appropriately biased. Since Poisson sampling results in a variable-sized sample, it will generally have relatively higher variance compared to the other two options for a comparably-sized sample if the sampling process is not biased towards particular tuples; that is, if $p_i = p_j$ for every i, j pair.

However, one big advantage of Poisson sampling is that by carefully tailoring the p_j's so that the process is more likely to select more important data items, variance can be greatly reduced, sometimes by a very significant amount. This is often called *biased* sampling. A carefully biased sample can more than compensate for any increase in variance due to a variable sample size. In particular, it should be clear by examining the above variance formula that by making p_j large for those items with a correspondingly large t_j while at the same time keeping the sample size small by making p_j small for those items with a correspondingly small t_j, a relatively low-variance estimator can be produced.

2.5.4 Stratified Sampling

The strength of Poisson sampling is that it is quite easy to bias the sample to the more important data objects by simply tweaking the various p_i's. SRSWR can be biased in a similar fashion by amending the sampling process so that, rather than giving each data object identical selection probabilities, the process used to obtain each tuple is biased. Specifically, choose p_j's between 0 and 1 so that $\sum_j p_j = 1$. Then to draw a sample of size n from relation $|\mathtt{R}|$, the following three steps are undertaken, n times:

(1) Generate a random number r between 0 and 1.
(2) Find j such that $\sum_{i<j} p_i \leq r \leq \sum_{i \leq j} p_i$.
(3) Obtain the jth item from the dataset and add it to the sample.

Then, at each sampling step,

$$Pr[j\text{th item selected}] = Pr\left[\sum_{i<j} p_i \leq r \leq \sum_{i \leq j} p_i\right] = \sum_{i \leq j} p_i - \sum_{i<j} p_i = p_j.$$

Unfortunately, it is not so easy to bias SRSWoR in a similar fashion. Imagine that we attempt to apply a similar scheme to SRSWoR. When a tuple is obtained using SRSWoR, by definition it is not possible to obtain the same tuple a second time. This means that after adding the jth tuple to the sample during SRSWoR, all of the other p's must be updated. Given that the jth tuple has been sampled, the probability of sampling t_i for $i \neq j$ should be larger for the next sampling step. The difficulty here is that each p becomes a random variable that changes depending what is observed during the sampling process. As a result, attempting to characterize the sampling process statistically is exceedingly difficult at best.

Due to this difficulty, a different strategy, called *stratification*, is used to bias SRSWoR. In stratified sampling, all of the tuples in the dataset are first grouped into m different subclasses or *strata*. To obtain a sample of size n, SRSWoR is performed separately on each strata; the number of samples obtained from the ith strata is n_i, where $\sum_{i=1}^{m} n_i = n$. To estimate the answer to a single-relation `SUM` query, the total aggregate value of each individual stratum is estimated; by adding up the estimates, an estimate for the sum over the entire dataset is obtained. Formally, let $\mathtt{R}_i$ denote the ith stratum. Then the HT estimator is defined by setting

$$Y_i = \frac{|\mathtt{R}_i|}{n_i} \sum_{t_j \in \mathtt{R}_i} X_j t_j$$

and

$$Y = \sum_{i=1}^{m} Y_i.$$

Since the various sampling processes are independent, the variance of Y is nothing more than the sum of the individual variances:

$$\sigma^2(Y) = \sum_{i=1}^{m} \sigma^2(Y_i)$$

Furthermore, since each strata is sampled using SRSWoR, it is easy to calculate $\sigma^2(Y_i)$ accordingly.

One of the most classic results in survey sampling theory is the so-called *Neyman allocation*, based upon the pioneering work of Jerzy Neyman [241]. The Neyman allocation prescribes a set of n_i values to minimize $\sigma^2(Y)$.

Specifically, to optimize n_i, choose each n_i so that:

$$n_i = \frac{n|\mathtt{R}_i|\sigma^2(Y_i)}{\sum_{j=1}^{m}|\mathtt{R}_j|\sigma^2(Y_j)}$$

In practice, a small pilot sample can be used to estimate the $\sigma^2(Y_j)$'s and hence the optimal allocation.

If the dataset can be broken into strata, stratified sampling can provide for dramatic variance reduction compared to SRSWoR. There are two ways in which stratification can reduce variance. First, the simple process of breaking R into strata provides a natural avenue for variance reduction because one can create strata that are internally homogeneous, even in a dataset that is quite heterogeneous. For example, imagine a dataset containing two "clusters" of values; one set of values that are close to 10, and another set that are close to 100. Mixing them together results in a population with relatively high variance—with an even number of 10's and 100's, $\sigma^2(\mathtt{R})$ will be 2025. Thus, an estimate obtained via SRSWoR may be relatively inaccurate for small sample size. However, by stratifying into one set of values that are close to 10 and one set that is close to 100, it may be that $\sigma^2(\mathtt{R}_1)$ and $\sigma^2(\mathtt{R}_2)$ are both quite small—if $\mathtt{R}_1$ is composed entirely of 10's and $\mathtt{R}_2$ is composed entirely of 100's, then $\sigma^2(\mathtt{R}_1) = \sigma^2(\mathtt{R}_2) = 0$.

Second, even if it is not possible to produce strata that are all internally homogeneous, the Neyman allocation naturally targets (biases) more resources to those strata that are more likely to add to estimation error, thereby reducing variance in a fashion that is analogous to optimizing the various p_j's in Bernoulli sampling. The only case where the Neyman allocation cannot help is when all of the internal strata variances are identical.

Several notable incarnations of stratified sampling have appeared before in the database literature. As early as 1992, Haas and Swami [157] proposed the use of stratification for increasing the accuracy of join size estimation via random sampling. A more recent example of stratification in the database literature is the paper of Chaudhuri et al. [47] (an extended version of this paper appeared subsequently [48]). The key idea of this work is to partition the data into various stratum in such a way as to maximize the accuracy of queries that are answered using the resulting stratification. As another example, Babcock et al. [13] propose a stratification scheme where strata are constructed so as to ensure that, for any particular `GROUP BY` query, one or more strata can be

targeted that will potentially contribute many tuples to the query.

2.6 Sampling and Database Queries

Thus far, we have focused mostly on sampling for a simple `SUM` query over a single relation. While this is clearly a useful application of sampling, not all database queries are single-relation queries. In this section, we consider the problem of sampling for other queries familiar from the database world.

We begin with a discussion of the "easy" case: aggregation queries that contain a mix of relational selections, projections, cross products (joins), and grouping operations, followed by the final aggregation operation. These operations are "easy" because for the most part, all of them commute with sampling in the sense that the sampling operation can be pushed past the operations, deep down into a query plan. In other words, it is possible to first sample one or more relations, and then apply various combinations of these operations to obtain a well behaved estimator (often an HT estimator or variant thereof) of the final query result.

The "hard" case includes queries where the HT approach breaks down, as discussed in Section 2.3. This includes queries involving duplicate removal, antijoins, and outer joins. These operations do not commute with sampling. For example, sampling a relation and then applying duplicate removal to the sample breaks the HT approach because tuple inclusion probabilities depend on the numbers of duplicates in the database—in other words, the data distribution—and hence are unknown at estimation time. For these hard queries, much more advanced estimation procedures are needed.

Finally, we conclude the section by considering sampling for other aggregates such as `AVERAGE` and `VARIANCE`.

2.6.1 The Easy Case: Rel Ops that Commute with Sampling

For any query plan that contains only selection, projection, cross product (join), and grouping, making use of a sampling to estimate the final result is easy: sample each underlying relation first, then use any applicable query plan to evaluate the desired query over the samples. The final step is to appropriately scale up the query result(s) to compensate for the sampling. In the setting of the cross-product operation, we have already seen this approach in action in the final example of Section 2.2. More generally, if the sampling

fraction for the ith of m relations is p_i, then by using any applicable query plan to evaluate the query over the samples and scaling the aggregate result by $\prod_{i=1}^{n} p_i$, one can obtain an unbiased estimate for the answer to multi-relation SUM query using the HT approach. Historically, this particular estimator has appeared many times in the database literature for estimating the size of joins [271, 154, 153, 172, 173]. Perhaps the most well-known application is the so-called "ripple join" estimator of Haas and Hellerstein, which seems to be the first time that the estimator was considered for use beyond COUNT [148].

In more detail, consider an arbitrary select/project/join/grouping query of the form:

SELECT AGG($f(r_1 \bullet r_2 \bullet \ldots \bullet r_m)$)
FROM $\mathtt{R}_1$ AS r_1 , $\mathtt{R}_2$ AS r_2 , ...
WHERE $g(r_1 \bullet r_2 \bullet \cdots \bullet r_m)$
GROUP BY $h(r_1 \bullet r_2 \bullet \cdots \bullet r_m)$

Here "$\bullet$" denotes tuple concatenation, and f, g, and h are arbitrary functions which take concatenations of tuples from the base relations as input. Let $t_{j_1,j_2,\ldots,j_m}$ be the result of concatenating the j_1th tuple from $\mathtt{R}_1$, the j_2th tuple from $\mathtt{R}_2$, and so on, to form one candidate input tuple to the AGG operation. Then, following standard database semantics, the following is the final set of values that are input into the aggregate function AGG for the group having identifier gid:

$$Ans = \{f(t_{j_1,j_2,\ldots,j_m}) \text{ where } g(t_{j_1,j_2,\ldots,j_m}) = \mathtt{true} \text{ and } h(t_{j_1,j_2,\ldots,j_m}) = gid\}$$

Using our definition from Section 2.4, a *sample* of this result set is a mathematical object taking the form:

$$\left\{(X_{j_1,j_2,\ldots,j_m} f(t_{j_1,j_2,\ldots,j_m}) \text{ where } g(t_{j_1,j_2,\ldots,j_m}) = \mathtt{true} \text{ and } h(t_{j_1,j_2,\ldots,j_m}) = gid\right\}$$

In this expression, as in the case of single-relation sampling, the various $X_{j_1,j_2,\ldots,j_m}$'s control the number of copies of tuples from Ans that appear in the sample.[5]

[5]To keep the notation reasonable, in the remainder of this chapter we will generally use $t_{j_1,j_2,\ldots,j_m}$ to denote the complex expression "$f(t_{j_1,j_2,\ldots,j_m})$ where $g(t_{j_1,j_2,\ldots,j_m}) = \mathtt{true}$ and $h(t_{j_1,j_2,\ldots,j_m}) = gid$ and zero otherwise". Any time an arithmetic operation such as $+$ or $\times$ is applied to a tuple $t_{j_1,j_2,\ldots,j_m}$, the reader can assume that this is shorthand for applying the operation to the *result* of applying f, g, and h to the tuple.

Clearly, one way to obtain such a sample is to first materialize the set *Ans*, and then apply a sampling scheme such as SRSWoR or SRSWR directly to *Ans*. This would define the various $X_{j_1,j_2,\ldots,j_m}$'s directly according to the sampling scheme that is used. However, if we *first* apply a scheme such as SRSWoR or SRSWR to each base relation, and then answer the query over the resulting samples, we *still* obtain a set that is precisely of this form, though with different $X_{j_1,j_2,\ldots,j_m}$'s in general. More precisely, if the underlying sampling scheme applied to base relation i defines a random variable $X_j^{(i)}$ which specifies how many copies of the jth tuple from relation i are included in the sample from this relation, then it is the case that

$$X_{j_1,j_2,\ldots,j_m} = \prod_{i=1}^{m} X_{j_i}^{(i)}.$$

This expression follows immediately from basic query semantics; for a simple select/project/join/grouping query, if one includes n copies of a tuple t in a relation, then n copies will appear in the query output for every tuple that depends upon t.

Given this mathematical definition for $X_{j_1,j_2,\ldots,j_m}$, it is quite simple to define an HT estimator that is unbiased for `SUM` queries and works by first sampling the underlying relations, then applying the query to the samples, and then scaling up the result. For example, imagine that we perform Bernoulli sampling on each underlying relation, and that the probability of including any given tuple in the result set is p. Because the sampling of each relation is independent, we have

$$E[X_{j_1,j_2,\ldots,j_m}] = \prod_{i=1}^{m} E[X_{j_i}^{(i)}] = p^m$$

Thus, an unbiased HT estimator for the answer to a `SUM` query over a group with identifier *gid* is simply

$$Y = \sum_{j_1}\sum_{j_2}\cdots\sum_{j_m} \frac{X_{j_1,j_2,\ldots,j_m} t_{j_1,j_2,\ldots,j_m}}{p^m} = \frac{1}{p^m} \sum_{j_1,\ldots,j_m \in \text{sample}} t_{j_1,j_2,\ldots,j_m}.$$

In the general case, it is possible to use this tactic to create an unbiased estimator when applying almost any sampling scheme to the underlying relations. As long as the various samplings are independent, it will always be the case that

$$E[X_{j_1,j_2,\ldots,j_m}] = \prod_{i=1}^{m} E[X_{j_i}^{(i)}].$$

Thus, the desired HT estimator is obtained by multiplying each $t_{j_1,j_2,\ldots,j_m}$ in the final sample by $1/\prod_{i=1}^{m} E[X_{j_i}^{(i)}]$.

We have not discussed (and will not discuss) how to compute and estimate the variance of an estimator such as Y. Conceptually it is not too difficult to generalize equations (2.3) and (2.4) to the general setting considered here, but the derivations can be quite tedious and error prone. As in all variance computations, one must compute the expected value of products of pairs of the variables that control the sampling process: $X_{i_1,i_2,\ldots,i_m} \times X_{j_1,j_2,\ldots,j_m}$. In contrast to computing the expected value of $X_{j_1,j_2,\ldots,j_m} = \prod_{i=1}^{m} X_{j_i}^{(i)}$—which is quite simple because the input relations are sampled independently—computing the expected value of $X_{i_1,i_2,\ldots,i_m} \times X_{j_1,j_2,\ldots,j_m}$ is more involved because both $X_{i_1,i_2,\ldots,i_m}$ and $X_{j_1,j_2,\ldots,j_m}$ have constituent parts that refer to tuples sampled from the *same* input relations. Specifically, $E[X_{i_k}^{(k)} X_{j_k}^{(k)}] \neq E[X_{i_k}^{(k)}]E[X_{j_k}^{(k)}]$ in general, because these random variables refer to tuples that are both sampled from the kth relation; clearly $X_{i_k}^{(k)}$ and $X_{j_k}^{(k)}$ are not independent when $i_k = j_k$, since both random variables then refer to the same tuple. Even when $i_k \neq j_k$, the random variables $X_{i_k}^{(k)}$ and $X_{j_k}^{(k)}$ will be statistically dependent under any fixed-size sampling scheme, since $\sum_j X_j^{(k)} = n_k$, where n_k is the fixed number of tuples sampled from the kth relation. Thus potentially tricky correlations arise, and the possible cases must be considered carefully. We refer the interested reader to several research papers which include the relevant derivations [190, 147].

This section has explained some of the basic concepts behind sampling for general, multi-relation join queries. Given the vast scope for exploring both the theory and practice of sampling for such queries, many details have been left out. Thus, we encourage the interested reader to look more carefully at this topic. The best place to start is with the Haas-Hellerstein ripple join paper [148]. We close this subsection by trying to address a few of the more important details, at least at a high level.

Sampling and Joining: Do They Really Commute? We note that several papers in the database literature—see, for example, [50, 244]—explicitly or implicitly claim that sampling *cannot* be pushed down past a join a query plan, and that the only way to sample from the result of a join of `R` and `S` is to sample from `R` and then fully join that sample with all of the tuples in `S` (or vice versa). On the surface, this seems to conflict with much of what has

been presented in this section, so some clarification is in order.

For a sampling scheme $\mathscr{S}$ over a population U and a subset $A \subseteq U$, denote by $P_{\mathscr{S}}(A;U)$ the probability that scheme $\mathscr{S}$ will output A as a sample. Then two sampling schemes $\mathscr{S}_1$ and $\mathscr{S}_2$ over U are said to be *equivalent* if $P_{\mathscr{S}_1}(A;U) = P_{\mathscr{S}_2}(A;U)$ for all $A \subseteq U$. For certain sampling schemes, such as Bernoulli sampling, and certain database operations, such as selection, the sampling scheme in which selection precedes sampling is equivalent, in the above sense, to the scheme in which selection follows sampling. In general, however, sampling relations `R` and `S` and then joining the samples is *not* equivalent to joining `R` and `S` and then sampling from the join. For example, suppose that in scheme $\mathscr{S}_1$ we take a 10% Bernoulli sample from each of `R` and `S` and join the samples. In scheme $\mathscr{S}_2$, we take a 1% Bernoulli sample from the join of `R` and `S`. It is true that the sample-inclusion probability for a tuple $\mathtt{r} \bullet \mathtt{s}$ is 1% for both schemes. However, suppose that `r` joins with both tuples `s` and `t` from `S`. Under $\mathscr{S}_1$, the probability that both $\mathtt{r} \bullet \mathtt{s}$ and $\mathtt{r} \bullet \mathtt{t}$ are in the sample is

$$Pr[\mathtt{r}, \mathtt{s}, \text{ and } \mathtt{t} \text{ are sampled}] = 0.1 \times 0.1 \times 0.1 = 0.001,$$

whereas under $\mathscr{S}_2$, the probability is

$$Pr[\mathtt{r} \bullet \mathtt{s} \text{ and } \mathtt{r} \bullet \mathtt{t} \text{ are sampled}] = 0.01 \times 0.01 = 0.0001,$$

and the schemes are not equivalent. Indeed, as discussed before, $\mathscr{S}_1$ introduces statistical dependencies between the sampled elements of the join. So in this sense, the sampling and join operations do not commute.

However, when estimating the result of an aggregation query—such as a `SUM` query—over a join, *we do not care* that sampling does not commute with join in the foregoing sense. We can still push sampling below the join in the query plan, as long as we compute our HT estimator properly, that is, as long as we scale up properly. Of course, the scale-up procedure for schemes such as $\mathscr{S}_1$ and $\mathscr{S}_2$ may differ, since the $X_{i,j}$ random variables may have different distributions, but in both cases we can compute an HT estimator. So in this sense, sampling and join do commute. Note that, in our example, both schemes actually use the same HT estimator Y, since the individual item-inclusion probabilities are the same. The variance $\sigma^2(Y)$, however, differs sharply between the schemes, because—as we demonstrated—the joint inclusion probabilities differ.

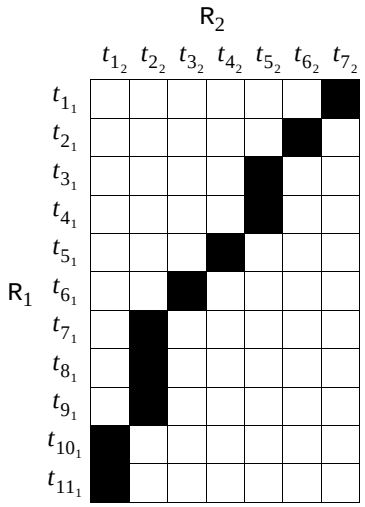


Fig. 2.1 An example join of two input relations

Sampling and then Joining: Accuracy Concerns. That being said, there *are* some valid concerns associated with pushing samples through more complex query plans. The most significant concern relates to the potentially high variance of estimators that result from sampling and then joining. Sampling and then joining differs from sampling over a single-relation query in that for the case of a single-relation query, the sampling fraction is relatively unimportant; even a few hundred samples from a billion-tuple relation can result in an estimate having high accuracy if the values that are being aggregated themselves have low variance. This is not the case when sampling multiple relations and then joining. This process is far more sensitive to the sampling fraction and may need much larger sample sizes than sampling when for the result of a single-relation query.

Consider the case of a two-relation join of $\mathtt{R}_1$ and $\mathtt{R}_2$, as illustrated in Figure 2.1. This figure depicts the two-dimensional grid that results from computing the cross product of the two input relations, where the tuples from one relation create the x-axis of the grid, and the tuples from the second relation create the y-axis of the grid. The tuples from the cross product that actually contribute to the answer to the query are the black cells in the grid. The difficulty of sampling for the answer to this query is that if the relations are sampled first and then the samples are joined, a black cell is present in the

resulting sample only if *both* of its constituent tuples from $\mathtt{R}_1$ and $\mathtt{R}_2$ were present in the sample. In this particular example, since each tuple in $\mathtt{R}_1$ joins with exactly one tuple in $\mathtt{R}_2$, we have a foreign-key join and there are 11 tuples in the answer set. If three samples are taken from both $\mathtt{R}_1$ and from $\mathtt{R}_2$ and then joined, then the expected number of answer tuples found is $\frac{3}{11} \times \frac{3}{7} \times 11 = \frac{9}{7}$, or slightly more than one out of the eleven result tuples. This is an "effective" sampling rate of less than 12%, even though the sampling rates for the two input relations were 43% and 27%, respectively.

In the general case of a foreign-key join where the size of the join is equivalent to the size of the larger relation, the expected number of samples found by sampling first and then joining is:

$$\frac{(\text{no. of samples from } \mathtt{R}_1) \times (\text{no. of samples from } \mathtt{R}_2)}{\text{size of the smaller relation}}$$

If each relation contains N tuples and we want to sample n tuples per relation to achieve a final expected sample size of c, where c is on the order of several hundred, then we must take $n = (cN)^{1/2}$ samples from each relation, which corresponds to a sampling fraction of $f = n/N = (c/N)^{1/2}$. If we want our sampling fraction in each relation to be less than or equal to f, then we must have $N \geq c/f^2$. For example, with a target sampling fraction per relation of at most 1% and a target sample size of $c = 200$ tuples, each relation must contain at least two million tuples. For a foreign-key star join involving k relations, the required absolute sample size per relation in the worst-case configuration (that is, the smallest possible join size) is $n = c^{1/k} N^{1-(1/k)}$ and the required relation size for a sampling fraction of f is $N = c/f^k$. So for a three-way join, each relation must have at least 200 million tuples. So clearly the sample-then-join approach breaks down for foreign-key joins if the relations are not very large or if many relations are involved in the join. (See [155] for a detailed discussion of the effectiveness of sample-then-join in the setting of join selectivity estimation.) In any case, unlike the single-relation scenario, the required absolute sample sizes are quite large, and grow with the size of the input relations.

This particular drawback of multi-relation sampling is widely recognized. One key idea in the database literature aimed at getting around this problem is Archarya et al.'s proposal for using *join synopses* [4]. Since most database queries make use of foreign key joins, Archarya et al. propose the idea of (logically) pre-joining the database relations across those foreign keys, then

drawing a sample of the resulting relation. This sample (called a *join synopsis*) is maintained for use in approximate query processing. Any multi-relation query which employs a subset of the pre-computed, foreign key joins in the join synopsis can then be re-written as a single relation query over the synopsis, thereby avoiding the inaccuracy incurred via the sampling of multiple relations. The drawback of this method is that it only handles those joins that are built into the join synopsis.

Another proposal for dealing with such hard cases is the stratification-based scheme called *bifocal sampling* proposed by Ganguly et al. [104]. For each possible join key, this method first partitions each relation into two subsets: those tuples with dense keys (appearing many times in the relation), and those with sparse keys (only appearing a few times). Samples are obtained from both sets. When joining two relations, the total aggregate value obtained by joining only those tuples with dense keys is estimated using the standard sample-then-join estimation procedure described in this section. The intuition is that this is the "easy case", where sampling followed by a join works quite well. Then the total aggregate value involving tuples with sparse keys is estimated with the help of an index. That is, for a sampled tuple from `R`, the precise set of matching tuples from `S` is located using an index, which effectively reduces the sampling process to single-relation sampling from `R`. The index is used because sparse keys can be problematic when sampling and then joining: if an important key value which joins with many tuples from the other relation is not included in the sample, then a severe underestimate is likely. In this way, the very hard case for sampling and then joining is avoided.

2.6.2 The Hard Case: Rel Ops that Do Not Commute

The reason that sampling "commutes" with the operations of selection, projection, joins, and grouping is that for such queries, whether one samples first and then runs a query plan, or whether runs a query plan and then samples, one can always derive an appropriate random variable of the form $X_{j_1,j_2,\ldots,j_m}$ that controls the number of copies of $t_{j_1,j_2,\ldots,j_m}$ in the answer set. Furthermore, the statistical properties of $X_{j_1,j_2,\ldots,j_m}$ depend only upon the sampling process, and not upon the actual data in the database. Thus, quantities such as the expected value of $X_{j_1,j_2,\ldots,j_m}$ are easily derived and HT estimators can be computed.

As discussed previously, this approach fails for certain types of queries. Consider the following antijoin example, which is similar to the `NOT IN` example of Section 2.3:

```
SELECT SUM(EMP.SALARY)
FROM EMP WHERE NOT EXISTS (
  SELECT * FROM SALES
  WHERE EMP.ID = SALES.ID)
```

The problem here is that if one first samples `EMP` and `SALES` and then runs the query, the statistical properties of the random variable X_i, which controls whether or not the ith tuple from `EMP` contributes to the final sum, actually depend upon the data itself. If there are many tuples in `SALES` that match up with the ith tuple in `EMP`, then this tuple is more likely to be discounted via a match with some member of the sample of `SALES` and X_i is more likely to be zero. If there are few tuples in `SALES` that match up with the ith tuple in `EMP`, then X_i is more likely to be non-zero.

Thus, if one samples first and then runs the query, it is difficult to characterize the random process that governs which tuples appear in the resulting "sample". The end result is that for all practical purposes, the antijoin operation does not commute with sampling. Other operations, such as outer joins, duplicate removal, and semijoin are all closely related to antijoin—for example, the antijoin generalizes the duplicate removal operation—and hence they share many of the difficulties experienced when sampling for the result of an antijoin.

Although these operations do not commute with sampling, the situation is not hopeless; sampling-based estimation for aggregation queries is still possible. One cannot, however, simply apply these operations to samples of the underlying relations and then scale up the final answer in a simple manner. More sophisticated estimation techniques are required. We summarize some of the significant results from the database literature below.

Sampling and Duplicate Removal. The most widely-studied of these "difficult" operations is the SQL `DISTINCT` operator (that is, duplicate removal) over a single relation, in conjunction with the `COUNT` aggregate. This is the so-called "`COUNT-DISTINCT` problem". The `COUNT-DISTINCT` problem has received so much attention for a number of reasons. First, a good solution to

the COUNT-DISTINCT problem would be of great utility in cost-based query optimization. It can be expensive to compute the exact number of distinct values for a column in a relation, and yet this quantity is of key importance when estimating the size of a join—hence, estimating the number of distinct values using a small sample is desirable. Second, when using sampling to estimate the answer to a GROUP BY query, it is always possible that one or more groups may exist in the data, and yet no samples from the group have yet been obtained. Thus, an estimate for the number of distinct values for the grouping attribute/s would give a user an idea of how many such unseen groups still exist. Finally, in a ROLAP setting, estimating the size of a datacube—that is, the number of nonempty cells—for purposes of storage allocation often amounts to estimating the number of distinct combinations of dimension attributes present in the underlying dataset.

Perhaps the earliest paper in the database literature to consider this problem is due to Hou et al. [172], where they suggest using Goodman's estimator, which first appeared in the statistical literature in 1949 (see Section 3.4 of Hou et al. for more details). The most widely-referenced paper in the database literature to consider this problem is due to Haas et al. [152]. In an extension of this paper, Haas and Stokes [156] comb both the statistical literature and the database literature and come up with eight different estimators (including Goodman's estimator), which they show can all be viewed as "generalized jackknife" estimators, and derive error formulas for such estimators. They then empirically test the estimators on 47 different real data sets. They observe that for low skew cases—that is, where the duplication factors for various distinct values do not differ widely—an "unsmoothed second-order jackknife" or a "stabilized" version of this estimator works best (see Shao [272] for an introduction to the jackknife). For high skew cases, an estimator from the statistical literature (Schlosser's estimator) works best. The authors therefore propose a hybrid estimator that combines the three estimators. To use this estimator, the squared coefficient of variation of the distinct-value duplication factors is first estimated from the sample, and then one of the three estimators is selected accordingly.

Charikar et al. [44] note that there is a spectrum of instances for the COUNT-DISTINCT problem; at one extreme there are those problem instances with little or no skew (for example, there is just one distinct value that encompasses the entire dataset), and at the other there are those problem instances

with a tremendous amount of skew (there is one distinct value that encompasses most of the dataset, and a large number of values with one tuple each). The problem is that with a small sample, no estimator can handle both cases well. Thus, an estimator must always do a poor job on one or the other, which sets a theoretical bound on the accuracy of any `COUNT-DISTINCT` estimator. They go on to propose a quite simple estimator that meets this theoretical bound. This is called the "Guaranteed-Error Estimator". They also propose a version of the Guaranteed-Error Estimator that is analogous to Haas et al.'s hybrid estimator, with the additional goal of providing for a smooth transition from the low-skew to high-skew case. Experiments show that this new, adaptive estimator works well in practice, though we have found that the relative performance of the various estimators is rather sensitive to the accuracy criterion used. Overall, sampling-based estimation of `COUNT-DISTINCT` queries remains a very challenging problem.

`SUM` **Aggregates over Antijoins and Semijoins.** Very few papers have tried to attempt sampling-based estimates for these more general queries. Jermaine et al. [191] consider the problem of sampling for two-relation, `NOT IN` queries (such as the `EMP` and `SALES` example given above), and characterize the bias of the estimator which samples the two input relations first, then joins the samples. They suggest that one way to handle such queries in practice is to use an approach that samples the two input relations, then selects a few of the tuples from the outer relation (`EMP` in our example). For those tuples, exact counts of the number of matches in the inner relation are computed. This information is used to unbias the resulting estimate.

One problem with this approach is that it requires the use of an index on the inner relation. In subsequent work, Joshi and Jermaine [196] consider estimation procedures for antijoins that do not require any such index. An unbiased estimator is developed, which unfortunately has high variance. The authors therefore consider the application of a "superpopulation method" to the problem (see Sarndal, Section 13.6 [268]) which first learns a generative model for the data, and then develops a biased estimator that happens to work well on datasets that are generated using the learned model.

In the (somewhat biased!) opinion of the author of this chapter, sampling-based estimates for the answer to queries containing antijoins, semijoins, and other difficult operations is an under-studied problem, and there are still many

limitations in the state-of-the-art. Chief among those is that existing work is limited to antijoins or semijoins of two relations, and not to more complex query plans that contain other operations such as additional joins in the inner or outer query. Thus, we hope that more research attention will be paid to this problem.

2.6.3 Beyond SUM Aggregates

The final topic that we discuss with respect to SQL queries over samples is the problem of estimating the answer to aggregate functions other than SUM (and COUNT, which is nothing more than the aggregate SUM(1)). Our discussion largely follows Haas [145], who provides an in-depth discussion of estimating (and giving confidence bounds for) the AVERAGE, STD_DEV and VARIANCE aggregate functions over multi-relation queries.

We begin by considering those aggregate functions that can be written as arithmetic functions of multiple SUM queries. These include AVERAGE, STD_DEV and VARIANCE.

Average. We first consider AVERAGE queries of the form:

SELECT AVERAGE($f(r_1 \bullet r_2 \bullet \cdots \bullet r_m)$)
FROM R_1 AS r_1 , R_2 AS r_2 , ...
WHERE $g(r_1 \bullet r_2 \bullet \cdots \bullet r_m)$

This can be written as the ratio of two individual SUM results. The numerator is:

SELECT SUM($f(r_1 \bullet r_2 \bullet \cdots \bullet r_m)$)
FROM R_1 AS r_1 , R_2 AS r_2 , ...
WHERE $g(r_1 \bullet r_2 \bullet \cdots \bullet r_m)$

And the denominator is:

SELECT SUM(1)
FROM R_1 AS r_1 , R_2 AS r_2 , ...
WHERE $g(r_1 \bullet r_2 \bullet \cdots \bullet r_m)$

Given this relationship, one obvious tactic is to estimate these two quantities at the same time, using the same samples, to obtain two estimators which we call $Y_{\mathtt{SUM}}$ and $Y_{\mathtt{COUNT}}$. By dividing these two estimators, we obtain an estimator for the final AVERAGE. Fortunately, "ratio estimators" of this sort over sam-

ples result in an (approximately) unbiased estimate for the final query result, and so $Y_{\texttt{AVERAGE}} = Y_{\texttt{SUM}}/Y_{\texttt{COUNT}}$ is generally a high quality estimate for the actual query result. Not only is this estimator approximately unbiased, but in practice it also has very low variance; typically much lower than the variance of either the numerator or the denominator. The reason is that since the two estimates use the same samples, they are highly correlated. If the numerator overestimates (or underestimates) the `SUM`, the denominator will also overestimate (or underestimate) the `COUNT`, and the resulting estimate may still be high quality.

Although it is intuitively clear that the estimator $Y_{\texttt{AVERAGE}}$ has low variance, actually computing this variance is challenging. The problem is the division operation in the estimator's definition. Since the variance of any estimator Y is $E[Y^2] - E^2[Y]$, the variance of our `AVERAGE` estimator is written as

$$E\left[\frac{Y_{\texttt{SUM}}^2}{Y_{\texttt{COUNT}}^2}\right] - E^2\left[\frac{Y_{\texttt{SUM}}}{Y_{\texttt{COUNT}}}\right].$$

The difficulty here is that $Y_{\texttt{SUM}}$ and $Y_{\texttt{COUNT}}$ are themselves sums over many different, correlated $X_{j_1,j_2,\ldots,j_m}$ random variables, and so each of the two terms in the variance is a ratio of two correlated, complex sums. One might obtain a reasonable estimator for $E^2[\frac{Y_{\texttt{SUM}}}{Y_{\texttt{COUNT}}}]$ by squaring $\frac{Y_{\texttt{SUM}}}{Y_{\texttt{COUNT}}}$, but dealing with $E[\frac{Y_{\texttt{SUM}}^2}{Y_{\texttt{COUNT}}^2}]$ is particularly nasty, since it requires computing the expected value of a ratio of two complicated squares.

Fortunately, there are some tools at our disposal for handling this difficult computation. The standard way that this is handled in statistics is via use of the so-called *delta method*, also known as *Taylor linearization*; see Sarndal [268, Section 5.5]. In our example, the technique uses a first-order Taylor series to remove the ratios from the variance computation. Specifically, write the `AVERAGE` estimator as

$$Y_{\texttt{AVERAGE}} = g(Y_{\texttt{SUM}}, Y_{\texttt{COUNT}}) = \frac{Y_{\texttt{SUM}}}{Y_{\texttt{COUNT}}}.$$

To apply the delta method, expand the function $g(x,y) = x/y$ in a first-order Taylor series around the true query results $Q_{\texttt{SUM}}$ and $Q_{\texttt{COUNT}}$ to obtain the ap-

proximation $Y_{\texttt{AVERAGE}} \approx \tilde{Y}_{\texttt{AVERAGE}}$, where

$$\begin{aligned}\tilde{Y}_{\texttt{AVERAGE}} &= g(Q_{\texttt{SUM}}, Q_{\texttt{COUNT}}) + \frac{\partial g}{\partial x}(Q_{\texttt{SUM}}, Q_{\texttt{COUNT}}) \times (Y_{\texttt{SUM}} - Q_{\texttt{SUM}}) \\ &\quad + \frac{\partial g}{\partial y}(Q_{\texttt{SUM}}, Q_{\texttt{COUNT}}) \times (Y_{\texttt{COUNT}} - Q_{\texttt{COUNT}}) \\ &= \frac{Q_{\texttt{SUM}}}{Q_{\texttt{COUNT}}} + \frac{1}{Q_{\texttt{COUNT}}}(Y_{\texttt{SUM}} - Q_{\texttt{SUM}}) - \frac{Q_{\texttt{SUM}}}{Q^2_{\texttt{COUNT}}}(Y_{\texttt{COUNT}} - Q_{\texttt{COUNT}}).\end{aligned}$$

The justification is that for a sufficiently large sample size, $Y_{\texttt{SUM}.}$ and $Y_{\texttt{COUNT}}$ will be close to $Q_{\texttt{SUM}}$ and $Q_{\texttt{COUNT}}$, so that the Taylor series will be a good approximation. We then approximate the variance of $Y_{\texttt{AVERAGE}}$ by the variance of $\tilde{Y}_{\texttt{AVERAGE}}$. Computing the variance of a linear function of random variables is relatively easy, using the fact that $\sigma^2(aX+c) = a^2\sigma^2(X)$ and $\sigma^2(X+Y) = \sigma^2(X) + \sigma^2(X) + 2\mathrm{Cov}(X,Y)$ for any random variables X, Y and constants a, c, where "Cov" denotes covariance. We obtain

$$\begin{aligned}\sigma^2(\tilde{Y}_{\texttt{AVERAGE}}) = \frac{1}{Q^2_{\texttt{COUNT}}}\sigma^2(Y_{\texttt{SUM}}) + \frac{Q^2_{\texttt{SUM}}}{Q^4_{\texttt{COUNT}}}\sigma^2(Y_{\texttt{COUNT}}) \\ - 2\frac{Q_{\texttt{SUM}}}{Q^3_{\texttt{COUNT}}}\mathrm{Cov}(Y_{\texttt{SUM}}, Y_{\texttt{COUNT}}).\end{aligned} \tag{2.5}$$

We can compute the variances of $Y_{\texttt{SUM}.}$ and $Y_{\texttt{COUNT}}$ (somewhat tediously) as discussed in Section 2.6.1; the covariance computation is similar. When estimating $\sigma^2(Y_{\texttt{AVERAGE}})$ from a sample, we replace the unknown constants $Q_{\texttt{SUM}}$ and $Q_{\texttt{COUNT}}$ in equation (2.5) by their estimates $Y_{\texttt{SUM}}$ and $Y_{\texttt{COUNT}}$.

Variance. `VARIANCE` is a trickier extension of `AVERAGE`, since the variance of a set of values is the average squared value in the set, minus the square of the average value in the set. To develop an estimator for this quantity, we consider the following `SUM` query:

`SELECT SUM(`$f^2(r_1 \bullet r_2 \bullet \cdots \bullet r_m)$`)`
`FROM` R_1 `AS` r_1 `,` R_2 `AS` r_2 `, ...`
`WHERE` $g(r_1 \bullet r_2 \bullet \cdots \bullet r_m)$

We then define an estimator $Y_{\texttt{SUM}^2}$ that is appropriate for this query. Given $Y_{\texttt{COUNT}}$, $Y_{\texttt{SUM}}$, and $Y_{\texttt{SUM}^2}$, the following is a reasonable estimator for the query result:

$$\frac{Y_{\texttt{SUM}^2}}{Y_{\texttt{COUNT}}} - \frac{Y^2_{\texttt{SUM}}}{Y^2_{\texttt{COUNT}}}$$

Note that this estimator may have some bias, because while the term $\frac{Y_{\texttt{SUM}^2}}{Y_{\texttt{COUNT}}}$ is (nearly) unbiased for the average squared value of all $t_{j_1,j_2,\ldots,j_m}$ in the data, and while $\frac{Y_{\texttt{SUM}}}{Y_{\texttt{COUNT}}}$ is (nearly) unbiased for the average $t_{j_1,j_2,\ldots,j_m}$ in the data, squaring $\frac{Y_{\texttt{SUM}}}{Y_{\texttt{COUNT}}}$ does not necessarily result in an unbiased estimate for the square of the average $t_{j_1,j_2,\ldots,j_m}$. In fact, squaring an estimator generally results in an estimator with at least some upward bias; thus, in the authors' experience the resulting, overall estimator can underestimate the variance. However, as the sampling fraction increases, the bias shrinks and the resulting estimator is very reasonable. Just as in the case of `AVERAGE`, an estimate for the variance of this estimator can be obtained via application of the Taylor linearization method.

Standard Deviation. An estimator for the `STD_DEV` aggregate is obtained by simply taking the square root of the estimator for `AVERAGE`. While this may induce some additional bias, it is also a reasonable choice. Again, an estimate for the variance of the estimator can be obtained via the Taylor linearization method.

Median. Though `MEDIAN` is not one of the standard SQL aggregate functions (because it cannot always be evaluated in a single pass through the dataset) it is quite common in practice, and it is also amenable to sampling. The obvious estimator is applied by simply obtaining the median value of the items that have been sampled from the dataset. A procedure for estimating the variance of this estimator is more involved; we point the reader to Sarndal et al. [268], Section 5.11 for details of the variance computation in the case of a single relation.

Min and Max. Sampling for the `MIN` and `MAX` aggregate functions from a finite population is an exceedingly difficult task. The problem is that a dataset can define an arbitrary distribution, and without any prior or workload knowledge it is virtually impossible to use a sample to guess the largest (or smallest) value without inspecting every data item. For example, imagine that a relation contains the following values:

$$\langle 1, 1, 2, 4, 5, 7, 8, 10^{10} \rangle$$

Now, imagine a second relation that contains the following values:

$$\langle 1, 1, 1, 7, 12, 19, 36, 45 \rangle.$$

It should be clear that without domain knowledge, any estimator for the MAX aggregate function which generally works well on a relation that resembles the first instance will not work well on a relation that resembles the second. The problem is that if an estimator works well on the first type of dataset by scaling up some statistic that is computed over the sampled data items, it will (likely) radically overestimate the MAX value for the second type. By the same token, an estimator tailored for the second type of dataset would radically underestimate on those of the first type.

Given its difficulty, the problem has not been studied extensively in the database literature. The relevant statistical literature focuses mostly on estimation procedures for parametric distributions [213]. The only paper in the database literature which proposes using sampling for aggregates of this form employs a so-called "Bayesian" estimator [242] that does, in fact, employ prior information to make the problem more reasonable [293]. The basic idea is to first (offline) learn what the distribution of values in common queries tends to look like. This can be done by looking at a historical query workload. For example, we might first construct a statistical model that says that one half of the time, the set of tuples queried looks like the first relation; the other half of the time, the set of tuples queries looks like the second. Then, when a query is issued and a sample is obtained, the original, offline model is updated to take into account the new samples that have been obtained (this is a classic example of the Bayesian approach). This additional information can be used to construct a relatively high-quality estimator. For example, if, after taking five samples we obtain $\langle 1, 1, 7, 32, 45 \rangle$, we might more accurately guess that we are in the second regime, and so taking the largest value in the sample (45) will provide a reasonable estimate for the query answer. If we determine that we are in the first regime, then we might estimate the MAX as being much larger than the largest sampled item.

Top-k. A top-k query computes the k most frequent values in a dataset R, along with their frequencies. The challenge is that simply taking the top-k values in a random sample of size n and scaling the corresponding frequencies upward by a factor of $|R|/n$ overestimates the true frequencies of the top-k values, even though such a scale-up procedure unbiasedly estimates the frequency of any individual, specified value in the data. This difficulty stems from the fact that the expectation of the maximum of a set of random variables

(the frequency estimators) is greater than the maximum of their expectations (the true frequencies). Cohen et al. [55] provide sampling and estimation procedures for this problem.

2.7 Obtaining Samples from Databases

Thus far, we have focused exclusively on how one might *make use of* random samples from a dataset. We now consider the question of how one might *obtain* random samples from a dataset that is stored, for example, by a database management system on a modern hard disk. Specific topics include: tuple-based versus block-based sampling, sampling in a single pass from a relation, sampling from an index, and single-pass sampling in such a way as to take into account deletions.

2.7.1 Block-Based vs. Tuple-Based Sampling

At the present time, most (large) datasets are stored on hard disks, and this situation will continue for at least the next few years. This fact is of fundamental importance in sampling, because hard disks are mechanical devices. To obtain a random tuple from the dataset (for example, when implementing SRSWoR), the disk head must physically be moved to the track containing the desired data, and the disk must spin until the desired tuple falls under the disk head before it can be transferred to the CPU. Because the moving parts in a hard disk are subject to the laws of Newtonian physics, it is difficult to accelerate the disk head fast enough to cut the cross-disk "seek" time down under a few milliseconds. Furthermore, due to the huge forces involved in spinning a disk quickly, a full revolution of the disk will always require a few more milliseconds. This means that the process of obtaining an item from a randomly-selected location on disk takes around three to five milliseconds, even for the fastest disks. Such considerations can often make random sampling from a disk a very expensive proposition—at 5ms per sample, this is only 200 samples per disk, per second.

One way around this problem is to obtain an entire *block* of data tuples at one time, as opposed to a single tuple. Datasets are naturally organized into blocks (or pages) that are atomic in the sense that if one tuple from a page is read into main memory, then all of the tuples are read into memory. The reason for such an organization is that the additional cost to read hundreds

or thousands of tuples that are co-located with a given tuple is small or even negligible due to the density with which bits are written on the surface of the disk. In many applications, it is hard to justify wasting this data. In the remainder of this section, we refer to the tactic of obtaining an entire page or block of tuples from a randomly selected location on disk (as opposed to a single tuple) as *block-based* or *block-level* random sampling.

Obviously, block-based random sampling radically increases the rate at which data can be randomly sampled from disk. However, the X_j random variables governing how many copies of each tuple are selected via block-based random sampling do not retain their statistical properties vis-a-vis tuple-based sampling, and quite strange correlations can be induced. For example, imagine that tuples are generally clustered on disk via their insertion time into the dataset. This makes it more likely that block-based sampling will select groups of tuples that were inserted into the dataset at roughly the same time, which may alter the statistical properties of any subsequent estimators.

Fortunately, this issue can be dealt with. Imagine that rather than viewing a relation $\mathtt{R}$ as a pool of $|\mathtt{R}|$ tuples that may be sampled, the relation is viewed as a pool of $|\mathtt{R}|_b$ blocks that might be sampled, and the sampling process is used to select those blocks at random, rather than tuples. For example, to implement SRSWoR in order to obtain n blocks, we might run the following two steps, until n blocks have been obtained:

(1) Produce a random number r from 1 to $|\mathtt{R}|_b$, inclusive.
(2) If the rth block from the dataset has not been previously added to the sample, then obtain this block and add it to the sample.

Given such a sampling process, most of the estimators discussed thus far (most notably, the standard selection/projection/join/grouping estimator from Section 2.6.1) can be applied with only a slight modification. This is done as follows. Rather than letting $t_{j_1,j_2,\ldots,j_m}$ refer to the value obtained after applying functions f, g, and h to the result of concatenating tuple j_1 from the first relation with tuple j_2 from the second relation and so on (see Section 2.6.1), $t_{j_1,j_2,\ldots,j_m}$ instead refers to the *total* aggregate value obtained by applying f, g, and h to the *cross product* of *all* of the tuples in the j_1th block from the first relation, the j_2th block from the second relation, and so on. Specifically, let $B_j^{(i)}$ refer to the set of tuples in the jth block from the ith relation and let

$f'(t) = f(t)$ if $g(t) = \texttt{true}$ and $h(t) = gid$; otherwise, $f'(t) = 0$. Then let

$$t_{j_1,j_2,\ldots,j_m} = \sum_{t_1 \in B_{j_1}^{(1)}} \sum_{t_2 \in B_{j_2}^{(2)}} \cdots \sum_{t_m \in B_{j_m}^{(m)}} f'(t_1 \bullet t_2 \bullet \cdots \bullet t_m).$$

Also redefine $X_{j_1,j_2,\ldots,j_m}$ to be the random variable that controls whether the sample contains block j_1 from relation 1, block j_2 from relation 2, and so on. Given these redefinitions of $t_{j_1,j_2,\ldots,j_m}$ and $X_{j_1,j_2,\ldots,j_m}$ in terms of blocks rather than tuples, our prior discussion about sampling for `SUM`, `COUNT`, `AVERAGE`, `VARIANCE`, and `STD_DEV` over joins and grouping formally applies without modification. Crucially, the analysis of variance and unbiasedness still holds without modification, and all of the estimation and variance computations formulas hold as written (but with the new block-level interpretation). Perhaps this result is not surprising, since we have merely redefined the block as the atomic sampling unit.

The resulting estimator will usually be more accurate than if only one tuple is used from each sampled block. Furthermore, if the tuples in each block are truly uncorrelated, then the extra accuracy gained by using all of the data in a block or page can be very significant. A block or page may have hundreds or thousands of tuples. Since variance generally decreases linearly with sample size in the case where samples are uncorrelated, using all of the tuples in a block can result in a thousand-fold decrease in variance for the same number of disk-head movements.

We note, however, that block-based sampling is not always the preferred retrieval method. We refer the interested reader to Haas and König, who carefully consider the issue [150]. For example, as the authors point out, it may be that one is worried not just about the number of random seeks, but also about the CPU cost of processing all of the tuples in a block, which can sometimes be expensive. In this case, tuple-based sampling may be the better choice, or a hybrid scheme that samples blocks and tuples may be preferred. Also, the foregoing trick—of simply redefining the estimation process so that it uses sampled blocks rather than sampled tuples—is only applicable when the underlying computation makes use of summations over tuples that can be redefined as summations over blocks. For example, the computation underlying a `COUNT-DISTINCT` query does not use such summations. For such queries, the question of block-based versus tuple-based sampling is much more complicated. See Haas et al. [151] for a partial discussion of how to

apply block-based sampling to the COUNT-DISTINCT problem. In general, current COUNT-DISTINCT estimators based on block-level sampling schemes tend to be extremely variable; more work on this problem is needed. Finally, see Chaudhuri et al. [49] for other applications of block-based sampling.

2.7.2 Sampling via a Full Scan

Compared to either block-based or tuple-based sampling, the fastest way to get data off of a disk is always via a sequential scan: first, the disk head is moved to the beginning of the data file, and then data are read in sequence from start to finish. A sequential scan will typically read tuples from disk ten times as fast as a block-based algorithm, and possibly tens of thousands as times as fast as a tuple-based algorithm.

The goal is to read all of the tuples in a file from start to finish such that, for any $n \geq 1$, a size n random sample of the file is available after n tuples have been read. The problem, of course, is that the first n tuples in a data file are likely not anything close to a random sample of the tuples in the file. This situation is easily remedied if the data in the file have been randomly shuffled before the scan is started. This idea was pioneered by Hellerstein, Haas, and Wang [165] in their paper on "online aggregation." To perform the required random shuffling, it is easiest to attach a random bit string to each tuple in the dataset. Then, an efficient, external sort (such as a two-phase, multi-way merge sort) is used to order the data so that they are sorted in ascending order of this bit string. After the sort, the first n tuples in the file are equivalent to n tuples selected via SRSWoR. Another approach is to use spare compute resources to perform the required re-ordering in the background, so that as new data are added, the random order is still maintained [261].

2.7.3 Sampling From Indexes

One of the difficulties associated with implementing tuple-based sampling is ensuring that tuples are selected with the correct probability. This can be non-trivial because each disk page may have a different number of tuples on it. If one first selects a page at random, then selects a random tuple from within the page, tuples on pages that have fewer tuples are more likely to be sampled. This can result in bias.

One way to overcome this problem is using an available indexing struc-

ture to guide the sampling process. The pioneering work in this area is due to Olken and Rotem. In one paper, they consider how to use the structure of a hashed file to implement sampling so that each tuple is selected with the appropriate probability [247]. In that paper, a number of accept/reject-based sampling algorithms are proposed. "Accept/reject" methods first locate a tuple at random, then compute the probability of locating the tuple, and then decide randomly whether to accept or reject the random tuple. If the tuple has a large probability of being located, then it is rejected with high probability to compensate for the higher probability of being located.

In another paper, Olken and Rotem consider accept/reject algorithms for sampling from B+-Trees [245]. One problem with such accept/reject algorithms is that if things go poorly, many, many tuples may need to be located before any tuple is accepted for inclusion in the sample. Antoshenkov [9] follows up Olken and Rotem's B+-Tree algorithms and attempts to address the problem with accept/reject sampling in this context by proposing the use of a "pseudo-ranked" B+-Tree. If a B+-Tree contains "rank" information (that is, each node in the tree contains statistics on how many tuples are contained in each of its subtrees) then sampling is quite easy, and an accept/reject algorithm is not needed. Specifically, a random number r from 1 to $|\mathtt{R}|$ is generated, and the rank information is used to locate the rth tuple in the tree. But (as Antoshenkov points out) maintaining the rank information is costly when the tree is updated, because it requires updates to all of the nodes from root to leaf at each and every insertion into the tree. Thus, Antoshenkov proposes pseudo-ranked B+-Trees, where only bounding information on the number of tuples in each subtree is stored. If this bound is reasonably tight, then the vast majority of the rejections can be avoided.

We close this subsection by noting that one very desirable characteristic of algorithms for sampling from B+-Trees is that they can be used for sampling directly from the result of a relational selection predicate, without having to sample the entire relation and then filter the samples to obtain a sample of a desired size. For example, a database may store data corresponding to many years of retail sales, but a user query might focus on only two days' worth of data. If the tuple timestamps are indexed by a B+-Tree, it is possible to use the structure of the tree to sample directly from those two days. Any branches of the tree falling outside of the desired range are treated as having zero tuples, and the accept/reject or ranked sampling algorithm is modified accordingly.

2.7.4 Sampling From Tuple Streams

Most of the methods discussed thus far for drawing a sample from a dataset have assumed that the dataset is fully materialized and the goal is to draw a single tuple at random. A larger sample is drawn via repeated invocations of the sampling algorithm. This is not always the case. It may be that the data are treated as a "stream", where the goal is to maintain a sample of all of the data that have been encountered thus far, or to maintain data structures that allow a sample to be produced once the last tuple from the stream as been encountered. Such a stream-based scenario is relevant when the data flow through the system at high speed without ever being stored on disk [74]. It is also relevant when one wishes to draw a sample of non-trivial size from a data file, so it is better to draw the sample in batch in a single, sequential pass rather than via a large number of random accesses.

In this subsection, we give only the briefest overview of stream-based sampling algorithms. For a more detailed treatment, we refer the interested reader to Haas' chapter "Data-Stream Sampling: Basic Techniques and Results" in Garofalakis et al.'s book on data stream processing [107], as well as to the recent PhD dissertation of Gemulla [113].

Certain stream-based sampling algorithms are trivial. For example, Bernoulli (or Poisson) sampling over a stream in one pass is easy. Suppose that the inclusion probability for the jth tuple is p_j. As this tuple passes by, we need only simulate a coin-flip where the probability of seeing "heads" is p_j. If a heads is obtained, the tuple is added to the sample. Otherwise, the sample is ignored. Rather than simulating individual coin flips, we can speed up the algorithm by simulating the gaps between subsequent acceptances: at each acceptance, the algorithm simulates the gap until the next acceptance—which can be done quite easily—and then essentially "goes to sleep" until the next accepted tuple arrives.

Perhaps the most classic stream-based sampling algorithm is the SRSWoR algorithm for a data stream of unknown size, which is known as the *reservoir algorithm*. The basic idea was proposed at least once or twice in the early 1960's; see, for example, Fan et al. [95] and their "method 4." It was then refined and made popular in the computer science research literature by Vitter [283]. The underlying concept is quite simple, and many variations are possible, each with its own performance characteristics. The algorithm starts

by selecting the first n tuples from the input stream and putting them into an array called the "reservoir." These tuples are trivially a size-n "random sample" of the tuples seen so far. Subsequently, when the ith tuple in the stream arrives ($i > n$), the tuple is accepted into the reservoir with probability n/i, replacing a randomly chosen reservoir element; with probability $1-(n/i)$, the ith tuple is ignored. It can be shown that, at any time during processing, the contents of the reservoir form a uniform, size-n random sample of all tuples seen so far. (A sampling scheme is *uniform* if any two population subsets of the same size are equally likely to be output by the scheme; samples produced by a uniform sampling scheme are also called "uniform.") Vitter shows how the algorithm can be speeded up by simulating the gap between acceptances, as described for Bernoulli sampling; in the current setting however, simulating the gap is far from trivial, and accept/reject techniques are needed.

Several variations of the reservoir algorithm have appeared in the database literature. Brown and Haas [27] consider use of the reservoir algorithm for sampling in parallel from multiple streams. Several papers [114, 192] consider the case when the sample to be maintained is too large to keep in memory, and must be stored on disk.

Another widely-known stream-based sampling algorithm from the database literature is the algorithm for producing *concise samples* and its close cousin, the algorithm for producing *counting samples* [125]. Both of these algorithms attempt to produce a bounded-size, compressed representation of a uniform sample, where not only is each sampled tuple included in the representation, but information on the number of repetitions in the stream is also represented. In the extreme case where only a few values ever appear in the stream, these sampling algorithms may not even need to sample at all, and the entire stream can be summarized via a set of (tuple, count) pairs.

A unique problem when sampling from a data stream in a dynamic environment is that the data stream may also contain deletions of existing data, as well as insertions of new data. This is especially relevant if the goal is to maintain a sampling-based synopsis of a large dataset in an online fashion by only monitoring the insert/delete stream, so that at any instant, one always has access to a sample of the current version of the underlying dataset without having to look at any of the base data.

Gemulla et al. [116, 118] have provided a state-of-the-art algorithm for maintaining a bounded-size SRSWoR from a data stream in a single pass

while allowing both insertions and deletions. Bounded-size samples are desirable because they reduce the complexity of memory management and allow control of the computational costs for algorithms that use the samples. The authors propose a "random pairing" (RP) algorithm, assuming that the data size is relatively stable. The idea is to "compensate" for prior deletions by pairing each inserted tuple with an "uncompensated" prior deletion. In more detail, when a deletion is encountered, the RP algorithm removes the tuple from the reservoir, if it is present. If the deleted tuple is present, a counter c_1 is incremented; if it is not, c_2 is incremented. If both counters equal 0 just before a new insertion is encountered, the insertion is processed according to the classical reservoir algorithm. Otherwise, the insertion is randomly "paired" either with a "bad" prior deletion that resulted in a tuple being removed from the reservoir or with a "good" deletion that did not affect the reservoir; the respective pairing probabilities are $c_1/(c_1+c_2)$ and $c_2/(c_1+c_2)$. If the new insertion is paired with a bad deletion, it is added to the reservoir, so that the reservoir size increases, and c_1 is decremented by 1. If the new insertion is paired with a good deletion, it is not added to the reservoir and c_2 is decremented by 1. In a later paper [117], the authors consider the problem of maintaining a Bernoulli sample over a multiset in the presence of insertions and deletions.

A problem that has plagued the literature in this area is that some streaming algorithms which attempt to bound the sample size (including some of the foregoing algorithms) introduce subtle departures from the asserted uniformity of the samples. The problems usually arise from attempts to combine Bernoulli sampling—which has unbounded sample size—with a bounding step that involves ideas such as periodically purging the Bernoulli sample or switching over to reservoir sampling when the sample sizes hits an upper bound. Nonuniformity for the concise/counting samples previously mentioned and for a scheme called "Bernoulli sampling with purging" is demonstrated in [27] and [113, Sec. 3.5.1B], respectively. The "hybrid Bernoulli" (HB) scheme in [27] and a "random pairing with resizing" (RPR) algorithm given in [116, 118] (designed to extend the RP algorithm to handle datasets that grow over time), also can be shown to suffer from this drawback.

One general approach for handling this problem is to settle for probabilistic bounds on the sample size that are exceeded with very small probability. Alternatively, the nonuniformity problem for the HB scheme can be elimi-

nated for an insertion-only stream by suitably randomizing the time at which a switchover to reservoir sampling starts. Gemulla [113, Sec. 4.2] shows that the RPR scheme can also be repaired. The basic idea when enlarging a sample from size n to size n' ($> n$) is to initially convert the sample to one that is statistically equivalent to a sample produced by an RP scheme with maximum sample size n' and d uncompensated deletions. (Here d is a parameter of the resizing algorithm.) This conversion requires access to the base data, but the author shows that any resizing algorithm must involve such access. After the initial conversion, the RP algorithm is run until the number of uncompensated deletions drops to 0, which implies that the sample size is equal to n'. The parameter d can be chosen to optimize the trade-off between the cost of accessing the base data and the cost of waiting for new stream elements to arrive.

The discussion so far has focused on uniform samples, but it is often desirable to obtain a sample from a stream in a biased way. For example, Babcock et al. [14] consider how to maintain a sample of the k most recent tuples from the stream. This involves expiring data that were included in the sample but are not in the k most recent tuples. Aggarwal [5] considers how to decay the inclusion probability of each tuples as new tuples arrive in the stream, so that with high probability, only the most recent tuples are included in the sample. In this work, the "recency" of a tuple t is defined in terms of the number of tuples that have appeared since t passed by on the stream. Gemulla and Lehner [115] consider the obvious question of how to maintain a sample when the "recency" of t is defined in terms of the *time* that has passed since t first appeared on the stream. This is challenging, since the rate at which tuples appear over time can change. If t arrives at a time when many tuples have recently arrived, but then (over time) fewer and fewer tuples show up, t will have its inclusion probability rise over time. This means that it is not acceptable to simply throw out all of the tuples that are not immediately included in the sample, and so it is not possible to guarantee a particular sample size using bounded space. In response to this, the authors develop a sampling scheme that works using bounded space, and does provide a lower bound on the expected sample size. Other interesting schemes for biased sampling have been proposed in the setting of network monitoring and related problems; see [54] and references therein.

2.7.5 Materialized Sample Views

Finally, we mention the problem of trying to maintain a *materialized sample view* in the presence of updates to the dataset. Just as a materialized view is a pre-computed query result for some important query, a materialized sample view is a materialized sample from the query result for some important query. Perhaps the best sources for information on this problem are Olken's PhD thesis [243] and Olken and Rotem's paper on the subject [246].

2.8 Online Aggregation Via Sampling

We close the chapter on sampling with a brief discussion of online aggregation. One of the key advantages of sampling is that it is additive: if one obtains a small sample, then adds some data to it, one obtains a larger (and presumably more accurate) synopsis of the base data. This means that if a user is not happy with the accuracy incurred via a sampling-based estimate, then more data can be added to the sample via an incremental procedure, and the estimate can be computed once again. Indeed, the process of adding more data and then re-estimating the query answer can be repeated again and again until the desired accuracy has been obtained; this process is known as *online aggregation*. The ability to support online aggregation is quite unique to sampling. The assumption which generally underlies online aggregation is that data are randomly shuffled on disk, so that an efficient relation scan can obtain random samples of larger and larger size. In contrast, other approximate query processing methods such as wavelets, sketches, etc., are "one and done" methods—if the estimate obtained is not accurate enough, then the user has no recourse except to create a larger synopsis, usually at great expense.

The idea of online aggregation was first proposed by Hellerstein, Haas, and Wang in their 1997 paper [165]. The goal was to provide quick interactive access to large datasets, allowing the user to determine on the fly whether the current query results were "sufficiently" accurate; if so, then the query could be terminated, thereby letting the user get on to their next interactive query. In their original paper, they considered single-relation queries with (possibly) a `GROUP BY` thrown in. One of the key technical innovations of the original online aggregation paper (originally suggested by M.R. Stonebraker) is the idea of "index striding," where the convergence of the different groups to the

true query answer is controlled by using an index to sample from the different groups at different rates. This is in contrast to the obvious way of handling a `GROUP BY` query: drawing a single sample without regard to the grouping operation, and then applying a series of different estimators to the sample, with one estimator for each group—a group's estimator assigns zero values to all of those tuples that do not belong to its group.

Later, Haas and Hellerstein extended their idea to multi-relation queries [147, 148], and they proposed the idea of a *ripple join*. In a ripple join of two relations `R` and `S`, a group of random tuples are first read from `R`, then a group of random tuples are read from `S`. These tuples are joined, and an estimate is produced. Then another group of tuples are read from `R`; all of the tuples read thus far from `R` are joined with all of the tuples read thus far from `S`, and another estimate is produced. Next, another group of tuples are read from `S`; all tuples read thus far from `S` are joined with all tuples read thus far from `R`, and another estimate is produced. This process is repeated until all of the tuples from `R` and `S` have been joined. The sampling rates are dynamically adjusted to that sampling effort is biased toward the input relation with the most variable data. The basic idea of a ripple join was later extended so that the ripple join was applicable in a parallel database environment [219].

One big problem with the ripple join is that it is not scalable—it only works if the results become sufficiently accurate quickly, so that the query can be aborted very early on. The whole algorithm relies on obtaining a few more tuples from an input relation, and then being able to efficiently join those tuples with all of the tuples obtained thus far from the other input relation. In practice, this means storing each relation in a hash table, and probing the hash table for matches when new tuples are read into memory. Unfortunately, this does not work if the tuples read thus far from both relations can not be stored in such an in-memory hash table due to the amount of data. In such a case, the ripple join must go to disk to find matches for new tuples, which is an exceedingly expensive proposition.

In response to this issue, Jermaine et al. propose a system for online aggregation called DBO that uses a whose sequence of ripple joins in tandem with a full query evaluation plan [189]. Every time that main memory fills with tuples, another ripple join estimate is produced and used to increase the accuracy of the current estimate for the query result. Then, all of those tuples are sorted and written back to disk as part of a sort-merge join. As new

data are read, main memory again fills and another ripple join estimate is produced. Again, this estimate is incorporated into the current estimate for the ultimate query result. Those tuples are again sorted and written back to disk. This is repeated until all of the input relations have been broken into sorted runs. Furthermore, the sort-merge joins make use of a randomized sort order, so that the merge phase of each sort-merge join outputs tuples in random order. Thus, those tuples can be used as input for ripple joins higher up in the query plan. In this way, it is possible to obtain tighter and tighter estimates via additional ripple joins throughout query processing, up until the time that the final answer is produced.

3

Histograms

The histogram is a venerable and well studied dataset synopsis, used in a wide variety of modern computer systems. In the database setting, histograms have been an integral component of query optimizers since at least the mid-1990's [43, p. 662], and are often used by the information management and statistical communities for purposes of data visualization. So far, histograms have not been used extensively for approximate query processing in commercial systems, but have great potential for this purpose. As a consequence, histogram-based approximation techniques have been extensively studied by the database research community; the AQUA approximate querying system developed at Bell Laboratories [126] is a notable example of a research prototype. A renewed surge of interest in histogram techniques has been driven by the networking community, who need small-space synopses to summarize patterns in packet traffic, in order to reduce congestion, detect denial-of-service attacks, and so forth. This new generation of histograms must contend with stringent space and time requirements, as well as the ever shifting statistical characteristics of streaming data.

As discussed in the review paper of Ioannidis [180], use of histograms and bar charts for data summarization and visualization goes back to the 18th century, and the term "histogram" itself was coined by statistician Karl Pear-

son in the 1890's. Histograms have been studied in the database literature for over 25 years, starting with the paper of Piatetsky-Shapiro and Connell [252]. Not surprisingly then, the statistical and database literature on histograms is enormous. As with other approximation techniques, we do not attempt to give an exhaustive survey. Rather, we focus on distinguishing those aspects of histograms that are pertinent to approximate query processing; key themes of our development are the adoption of a model-free point of view and the generalization of the notion of a histogram far beyond its classical origins. Our emphasis throughout is on those results and techniques that appear to have the greatest potential for practical use.

In the remainder of this chapter, we first introduce the key issues pertinent to histograms via a discussion of several simple, classical histograms, and delineate some important differences between the statistical and database viewpoints. We then focus on the use of one-dimensional (1D) histograms for approximate query answering, especially for "range-sum" queries, e.g., queries that specify the number of data items having values in a specified range. Although multi-dimensional histograms and general queries are of primary importance in our setting, a careful discussion of 1D-histograms is worthwhile because of their relative simplicity, well-developed theory, and historical importance. Moreover, techniques developed for 1D-histograms often appear as components of more elaborate algorithms for multi-dimensional histograms. We then discuss multi-dimensional histograms and approximate answering of general queries. It will be apparent from our discussion that 1D-histograms for range sums have received a disproportionate share of attention in the literature; perhaps by highlighting the relative paucity of results on multidimensional histograms and general queries, we will inspire further research on these challenging but important topics. We conclude the chapter by discussing histograms over streaming data, as well as specialized techniques for real-valued data.

3.1 Introduction

To ease our way into the topic of histograms, we first give an extended example of two classical types of histograms over real-valued data: equi-width and equi-depth. We then summarize the key features of the histogram approach to approximate query answering. Finally, we contrast the statistical and database

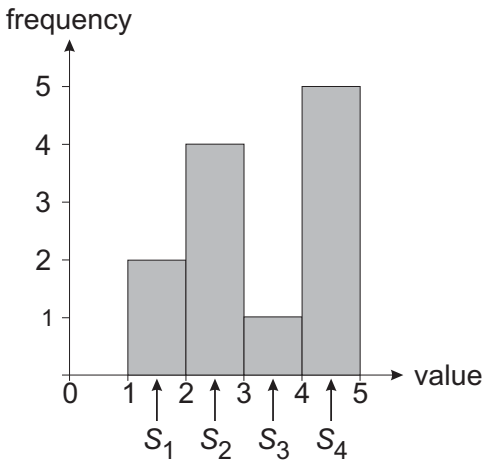


Fig. 3.1 Classical equi-width histogram on continuous data

approaches to histogram construction and use.

3.1.1 Classical Histograms on Real-Valued Data

We first consider the classical equi-width histogram on real-valued data, instantly recognizable to anyone who has taken a basic course in statistics. For example, we can approximately represent the set of twelve data points

$$D = \{\, 1.61, 1.72, 2.23, 2.33, 2.71, 2.90, 3.41, 4.21, 4.70, 4.82, 4.85, 4.91 \,\}$$

by the histogram pictured in Figure 3.1. Each data point is assigned to one of four disjoint *buckets*, or subsets, denoted S_1, S_2, S_3, and S_4. In our example, the buckets correspond to the four disjoint subintervals of the data-value domain $[1,5]$ that are displayed on the horizontal axis; each data point is assigned to the bucket whose associated subinterval contains the point. The height of each histogram bar corresponds to the number of points that fall in the associated bucket, i.e., to the *frequency* for the bucket.

This histogram is specified by the lower boundary of the leftmost bucket (1.0), the bucket width (1.0), and the four frequency counts for the four buckets (2, 4, 1, 5). Although, in this example, the data has merely been reduced from twelve numbers to six numbers, it is clear that the degree of compression can be much higher, e.g., a similar histogram representing 10^6 points having

values between 1 and 5 would still be represented by six numbers. Clearly, the histogram is a lossy representation of the data.

If the bucket boundaries are specified a priori (and hence are known to bracket the data values), then the histogram can be constructed during a single pass through the data, using four counters to compute the bucket frequencies. Thus the histogram requires $O(1)$ time per item—or $O(N)$ time in total—and $O(B)$ space to construct, where N is the number of data values and B is the number of buckets. Also note that the histogram is easy to maintain in the presence of insertions and deletions: simply determine the bucket associated with an insertion or deletion transaction, and increment or decrement the corresponding counter as appropriate.

Suppose that we are given the histogram in Figure 3.1 and wish to estimate the number of points N whose values lie between 1.1 and 4.5. This corresponds to a *range-count* query that can be expressed in SQL syntax as

```
SELECT COUNT(*) FROM D
WHERE D.val >= 1.1 AND D.val <= 4.5
```

Clearly, the contribution from the buckets S_2 and S_3 to the query answer are four points and one point, respectively, and these numbers are exact. The computations for these buckets are easy since they are completely contained within the query range $[1.1, 4.5]$. The problem is to estimate the contributions from the buckets S_1 and S_4, each of which partially overlaps the query range. A standard approximation is the *continuous-value assumption*, which simply assigns a count by multiplying the bucket frequency by the fraction of the bucket interval that lies within the query range. In our example, the fraction of the S_1 and S_4 intervals that lie within the query range are $(2.0-1.1)/(2.0-1.0) = 0.9$ and $(5.0-4.5)/(5.0-4.0) = 0.5$, respectively. Thus the respective contributions from these two buckets are estimated as $(0.9)(2) = 1.8$ and $(0.5)(5) = 2.5$, leading to an overall estimate of $\hat{N} = 1.8+4+1+2.5 = 9.3$ for the query answer. Since the actual query answer is $N = 8$, the approximation overestimates the answer by about 16%. In general, the time required to answer such a range-sum query is $O(B)$.

As can be seen from the above discussion, a simple equi-width histogram can be constructed, maintained, and queried very efficiently, and can result in a high degree of data compression. One deficiency of this histogram type is that the lowest and highest data values must be known (or at least bracketed)

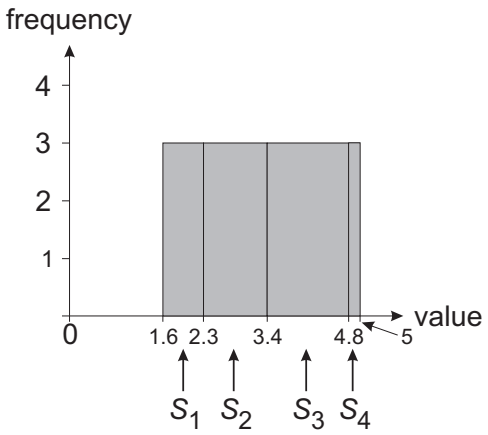


Fig. 3.2 Equi-depth histogram on continuous data

in order to determine the bucket boundaries. More importantly, the accuracy of an equi-width histogram can be quite poor. To see this, consider the histogram of Figure 3.1, and suppose that we add 999,999 additional data points, all taking values in the interval $[3.8, 3.9]$, so that the count for bucket S_3 is now 10^6. If we issue a range-count query that counts the number of points that lie in the interval $[3.0, 3.5]$, the continuous-value assumption yields an estimate of $\hat{N} = 500,000$. The true answer is $N = 1$ (which corresponds to the original data value of 3.41), so that the estimate is high by over 5 orders of magnitude. Indeed, for any given data-value range and specified number of buckets, there exist datasets and queries having arbitrarily large errors. If, in our specific example, we add more buckets, and hence decrease the bucket width, so that the details of the data distribution in the interval $[3.8, 3.9]$ are captured adequately, then most of the resulting buckets will be empty; this problem is exacerbated in higher dimensions. We can record only the non-empty buckets, but then we must also, in general, record not only the bucket frequencies, but also the bucket boundaries, which in turn reduces the number of buckets that can be stored within a given space allocation, and increases the time required to answer queries. The histogramming methods discussed in subsequent sections provide a number of different approaches, both heuristic and theoretical, to controlling histogram error while providing acceptable performance.

For the present, we briefly outline a classical approach to bounding worst-case estimation error, namely, *equi-depth* histograms. This class of histograms—first proposed in the database literature by Piatetsky-Shapiro and Connell [252]—selects bucket boundaries so that each bucket contains the same number of data points. Selecting bucket boundaries for N data points and B buckets is essentially equivalent to computing the (N/B)-quantiles of the dataset. Figure 3.2 displays a 4-bucket equi-depth histogram for the dataset D given above. (Note that the equi-depth representation is not unique in general.) For such a histogram, one must store the lower boundaries of each bucket, the upper boundary of the rightmost bucket, and the total number of data points. Using the equi-depth histogram together with the continuous-value assumption, the estimated answer to the previous range-count query on $[1.1,4.5]$ is $\hat{N} = 3+3+\big((4.5-3.4)/(4.8-3.4)\big)3 = 8.4$, and the over-estimation error is now only about 5%. More importantly, we know that the estimated answer to *any* range-count query can be off by at most ± 6, since at most two buckets can partially overlap the query range, and each bucket can erroneously include or exclude at most 3 points. More generally, for a B bucket equi-depth histogram and any query that selects at least $100s\%$ of the data points, the error in the query answer is at most $\pm\big(200/(sB)\big)\%$. Thus we can control the worst-case error for a large class of queries and datasets.

We pay a price, however, for the increased control: to construct the histogram, one either has to sort the data or use a quantile-computation algorithm. The former approach is expensive for large datasets, requiring $\Omega(N)$ space—typically unacceptable in practice—for an in-memory sort of a dataset containing N points, and otherwise requiring multiple passes over the data for an external sort. Specialized algorithms for directly computing quantiles are similarly expensive. Indeed, Munro and Paterson [234] have shown that, for any such algorithm, at least $\Omega(N)$ memory is required to compute a specified quantile in a single pass. The memory requirement can be reduced to $O(N^{1/p})$ at the expense of requiring p passes through the data, which is also typically unacceptable. One must therefore resort to approximate computation of quantiles. Greenwald and Khanna [132], for example, give an algorithm that computes an *ε-approximate quantile*, that is, an element whose rank r' is within $\pm\varepsilon N$ of the nominal rank r. The space requirement of this algorithm is $O\big(\varepsilon^{-1}\log(\varepsilon N)\big)$. Various versions of the quantile-digest algorithm [68, 168] can also be used in this setting. See [296] for a recent litera-

ture review.

3.1.2 Key Issues for Histograms

We see from the foregoing discussion that a histogram is obtained by partitioning the dataset into buckets (i.e., subsets) and, for each bucket, storing summary statistics about the data values in the bucket. Information about the buckets themselves, such as bucket boundaries, is also stored. At query time, the summary and bucket information is used to (approximately) reconstruct the data in the bucket in order to approximately answer the query. Important aspects of a histogram include:

- *Bucketing scheme* As discussed in the sequel, buckets need not be disjoint in general, and may even be recursive, in order to better capture the structure of the data. Data items can be assigned to buckets based on local considerations such as similarity of values or value frequencies, or on broader criteria such as "global optimality" of the histogram with respect to a workload or specified class of queries.
- *Statistics stored per bucket* In our examples so far, the only information stored for a bucket is the number of points in the bucket, together with information about bucket boundaries. The choice of which information to store is usually determined by the method used to approximate the data values in the bucket. As discussed previously, there is a trade-off between the amount of information stored per bucket and the number of buckets that can be maintained.
- *Intra-bucket approximation scheme* The only scheme discussed so far has been the continuous-value assumption for frequency counts, but many other approaches are possible. The approximation scheme also depends on the class of queries that the histogram is being used to answer.
- *Class of queries answered* Historically, much of the focus has been on answering range-count queries of the type discussed above, as well as *point-count* queries (or simply *point* queries) for discrete data, such as

```
SELECT COUNT(*) FROM D
WHERE D.color = 'red'
```

Such queries correspond to "selectivity estimation" queries that are used extensively in database query optimizers. More recently, there has been increasing interest in approximate answers to online analytical processing (OLAP) queries and general SQL queries.

- *Efficiency* Here efficiency refers to the space and time requirements for constructing the histogram, as well as the cost of using the histogram to approximately answer a query. The space requirements, in particular, can determine whether or not use of a given histogram is practically feasible—for some histograms the space needed for construction exceeds the size of the data itself. The construction costs depend on how the data is accessed; we distinguish between "datacube" and "relational" access models for discrete data, as defined in Section 3.2.2.
- *Accuracy* The goal is usually to give the most accurate answers possible, given a constraint on the allowable size (in bytes) of the histogram. Many different notions of accuracy have been proposed, and much research has gone into producing histograms that are "optimal" in the sense of having the smallest approximation error subject to a size constraint. Especially in the case of 1D-histograms and in the context of selectivity estimation for query optimization, many of the different histogram types discussed in the sequel have similar (and practically acceptable) accuracies, provided that the histograms are allocated "sufficient" storage space; e.g., the experiments in [286] indicate that, for 1D-histograms, the marginal benefits of increasing the bucket count diminish sharply beyond about 20 buckets. The differences are more marked for multi-dimensional histograms, and even 1D-histograms might only be allocated a small amount of memory if many histograms are being maintained in parallel; see, for example, the discussion of a "synopsis warehouse" architecture in [19, 27]. Also, the precision requirements might be higher for general approximate query answering than for the special case of se-

lectivity estimation in the context of query optimization (where many other sources contribute to the uncertainty of a query-plan cost estimate). Because approximate query-answering systems are not yet in wide use, there is a paucity of practical guidance on accuracy requirements.

- *Error estimates* This issue is closely related to, but different from, the issue of histogram accuracy. Since histograms yield approximate query answers, it is highly desirable to provide the user with tight error bounds for the specific query of interest. Some histogram construction methods provide a guarantee on the overall "average" error over a set of queries, which does not yield a useful error estimate for an individual query, or on the maximum error over a class of queries, which usually provides a query-specific error estimate that is too loose. Similarly, some methods guarantee that the histogram is optimal (or ε-close to optimal) for a class of queries, but again this does not yield individual error estimates.
- *Incremental maintenance* Especially when summarizing high-speed data streams, it is often important to efficiently update a histogram in the presence of insertions to and deletions from the dataset.

3.1.3 Statistical vs Database Viewpoint

In the statistics literature, histograms originally were used simply to summarize and visually represent data. More recently, however, histograms have often been viewed as nonparametric density estimators [270]. That is, the dataset is modeled as a collection of independent and identically distributed (i.i.d.) samples from an unknown probability density function f, and the histogram is a piecewise-constant approximation of f. (The estimation problem is "nonparametric" in that f is not assumed to lie in any particular parametric family of distributions such as the normal, exponential, or gamma families.) A typical problem in this setting is to choose bucket widths in a one-dimensional equi-width histogram to minimize the *mean integrated squared error*, defined as

$$\text{MISE} = E\left[\int [\hat{f}(x) - f(x)]^2\, dx\right],$$

where E is the expectation operator and $\hat{f}$ is the piecewise-constant histogram approximation to the density function. (Note that $\hat{f}$, being computed from random samples from f, is viewed as a random function, and hence the expectation operation appears in the definition of MISE.) If, for example, the number of data points N is large, f is twice differentiable, and the first derivative f' is square integrable, then the MISE is minimized by choosing the bucket width equal to $cN^{-1/3}$, where the constant $c = c(f)$ depends on the specific form of f. Scott [270] describes various data-based bounds and approximations to the unknown constant $c(f)$. In general, there are better, smoother approximants to f, including frequency polygons, averaged shifted histograms, and kernel density estimators.

In contrast, the database literature almost always takes a model-free point of view, considering the data that is present in the database as the only data of interest.[1] The goal is simply to obtain a low-error compressed representation of the dataset. As discussed in Section 3.7.2 below, however, direct application of statistical histogramming techniques to real-valued data sometimes leads to good estimation accuracy, even in the absence of a rigorous statistical model.

Another key difference between the two points of view is that the vast majority of enterprise database applications center on *discrete* data, i.e., discrete numerical data or categorical data. Without loss of generality, we can take the domain of a discrete attribute to be a set of integers of the form $\mathscr{U} = \{1, 2, \ldots, M\}$. We often use interval notation and denote this set by $[1, M]$; this is convenient since buckets are often defined via disjoint segments of [1,M], analogous to the partitioning of the real line as in Figure 3.1. For discrete numerical data or ordered categorical data, there is usually a natural mapping of the discrete values to the integers, e.g., day-month-year data can be expressed in terms of the number of days elapsed since a fixed reference date. In principal, we can also map unordered categorical data to integers, since the queries of interest in this setting are point queries so that the mapping can be arbitrary. Maintaining histograms on unordered categorical data is problematic, however, because an index is needed to map each categorical value to a histogram bucket; the space and query-time requirements for

[1] We assume throughout that data values are known with certainty; see Cormode et al. [62] for a discussion of histograms over uncertain data.

such a mapping index are typically unacceptable in practice. We henceforth reserve the term "discrete data" for data whose domain has a natural mapping to the integers. Note that real-valued data can also be mapped to the integers by discretizing, but, as discussed in Section 3.7.2, methods that are especially tailored to real-valued data tend to have superior performance. Conversely, techniques developed for real-valued data can have inferior performance in the discrete setting.

3.2 One-Dimensional Histograms: Overview

In the next few sections we study 1D-histograms in the setting of approximate query processing. Except for Section 3.7.2, we focus on discrete data.

3.2.1 Basic Notation and Terminology for 1D-Histograms

As discussed above, we can think of a multiset D of discrete data as having domain $\mathscr{U} = [1, M]$, and we denote by $f(i)$ the number of points in D having value $i \in \mathscr{U}$, i.e., the *frequency* of value i. The queries considered so far have focused on computing counts, i.e., sums of frequencies, over regions of the domain of data values. In order to encompass not only multisets of discrete data, but also a range of important applications including selectivity estimation, time series processing, and OLAP queries, we generalize this setup as follows. Let g be a nonnegative function defined on $\mathscr{U}$, and consider the problem of estimating quantities such as $\alpha = \sum_{i \in Q} g(i)$, where $Q \subseteq \mathscr{U}$ is the *query region*. When $g = f$ and Q is of the form $Q = \{ l, l+1, l+2, \ldots, u \}$ or $Q = \{ i \}$, this goal corresponds to approximating the answers to a range-count or a point-count query, respectively. For general functions g, this goal corresponds to approximating the answers to *range-sum* and *point-sum* queries. For example, $g(i)$ might be the total sales revenue for a company on day i, and we might be interested in approximately answering the range-sum query

```
SELECT SUM(SALES) FROM D
WHERE D.day >= 7 AND D.day < 14
```

The assumption that g is nonnegative slightly simplifies the mathematics, and entails no loss of generality, because a dataset containing negative g values can always be shifted by subtracting the smallest such value from each data element. In Section 3.6, we discuss the use of histograms for estimating the

answers to more general types of SQL queries.

Our discussion initially focuses on bucketing schemes that partition the domain $[1, M]$ into exhaustive, mutually disjoint segments. We first discuss methods for estimating the answer to query result, given a fixed set of buckets. We then consider the problem of determining a high quality set of bucket boundaries, and also discuss more complex, hierarchical bucketing schemes.

3.2.2 Data Access Models

To help focus attention on practical histogram schemes, we need to identify histograms whose construction and usage costs are acceptable for various applications. When analyzing the cost of a specific algorithm for constructing a histogram on discrete data, we first need to specify the format in which the data is available; the data access model strongly influences the cost of histogram construction. In this section we discuss data access models and in the following section we discuss histogram costs. We assume throughout a centralized, sequential processing model. Recent work—see, e.g., Kempe et al. [207]—has focused on the computation of histograms in novel processing environments such as distributed peer-to-peer networks, but this topic is beyond the scope of our discussion.

We focus on two primary data access models: the "datacube" and "relational" models. In the *datacube* model, the data is accessed as described previously, namely, as a list of values in the form $g(1), g(2), \ldots, g(M)$. This scenario corresponds, for example, to the case where each i is the index of a cell in a (one-dimensional) datacube and $g(i)$ is the "measure value" in the cell, as in the previous sales-revenue example. The assumption here is that the number of cells M is so large that the datacube will not fit in memory, and a histogram synopsis is needed to quickly answer OLAP queries. In the important special case where g equals the frequency function f, the datacube model assumes that the exact frequency distribution has been tabulated for the data, but that there are so many distinct data values that this distribution needs to be summarized. The datacube model also encompasses the *discrete time-series* model, where $g(1), g(2), \ldots, g(M)$ are viewed as time-ordered observations. We emphasize that, despite our classical-sounding terminology, these models are relatively abstract, and hence apply in settings that go beyond traditional database management systems.

CityId (j_n)	Sales (v_n)
2	20
2	30
1	40
5	10
3	5
3	7
4	15
2	6

Relational model

i:	1	2	3	4	5
$g(i)$:	40	56	12	15	10

Datacube model

Fig. 3.3 Data access models (SUM aggregate), $N = 8$ and $M = 5$

In contrast, the *relational* model assumes that the data is available simply as a list of length N of the form $(j_1, v_1), (j_2, v_2), \ldots, (j_N, v_N)$, where each j_n is an element of $[1, M]$ and v_n is a value associated with j_n. The data can be viewed as a sequence of "tuples" from a two-column relational table. The value $g(i)$ is obtained by applying an aggregation operation such as SUM or AVERAGE to the elements in the set $A_i = \{ v_n : j_n = i \}$. E.g., (i_n, v_n) might represent a transaction in which v_n dollars worth of merchandise was sold on day i_n, so that, using the SUM aggregate, $g(i)$ represents the total revenue on day i. In relational-database terms, the tuples are grouped by the first attribute and then aggregated. Figure 3.3 contrasts the relational and datacube access models (using the SUM aggregate as above); the relational access model corresponds to a scan of the table whereas the datacube access model corresponds to a scan of the array. We usually restrict attention to the most common case of the relational model, in which $v_n = 1$ for all n and the aggregation operator is SUM. This corresponds to the case in which the dataset D is available as a list of values in $[1, M]$ and $g = f$, so that $g(i)$ is the number of points in D having value i. This corresponds to the situation in a relational database where the data comprises a single column of a relational table, and we are executing a COUNT query over the column, grouping by column values. To simplify notation, we drop the v_n variables and simply assume that $D = (j_1, j_2, \ldots, j_N)$. Clearly, histogram construction is more expensive under the relational model, since the data elements have not been pre-grouped ac-

cording to data value. We assume, unless specified otherwise, that the values in D are unsorted in the relational model. If the values are sorted, then the data can be easily converted to datacube format during a scan. In general, conversion of relational data to datacube format requires at least one pass over the data prior to histogram construction [139]. Typically such an explicit conversion is impractical, and histogram construction algorithms tailored to relational data are needed—see the discussion at the end of Section 3.4.2—or the input to a datacube-based algorithm must be obtained by converting a sample of the relational data, as described in the following section. Unless otherwise specified, we assume the datacube access model.

3.2.3 Histogram Cost Considerations

What are "reasonable" costs for constructing and using a histogram? For at least some applications and data sizes, the user is able to tolerate a complete pass through the data, or perhaps even two passes, for histogram construction, provided that the per-item processing cost is small. This is especially true for the datacube access model, and less so for the relational data access model. When the amount of data is massive, an increasingly important scenario, then even a single pass through the data might be prohibitively expensive.

In this latter case, a common approach is to build a histogram from a small uniform sample of the data [235, 252, 258]. As discussed by Poosala et al. [258], for example, Kolmogorov's Theorem implies that the sampling error can be controlled simultaneously for all possible range queries by taking a sufficiently large sample. Donjerkovic and Ramakrishnan [91] point out that the sample size needs to be inflated to deal with the information loss incurred by compressing the sample into a histogram; e.g., for an equi-depth histogram with B buckets, the authors use Kolmogorov's Theorem to obtain the rule-of-thumb formula $s \approx 100B^2$ for the required sample size s, under the assumption that the sampling error should be an order of magnitude smaller than the histogram error. On the other hand, when using "maxdiff" and related histograms for query optimization—see Section 3.4.1—Poosala et al. found experimentally that a sample size of 2000 was more than sufficient for 1D-histogram construction. Chaudhuri et al. [51] consider a more sophisticated approach to collecting a sample on which to build an equi-depth histogram. For tuple-level sampling schemes, they use Chernoff bounds to

compute the sample size necessary to bound either a maximum-error metric or a stronger "symmetric difference" error metric. For page-level sampling schemes—which are far more I/O efficient than tuple-level schemes but incur dependencies between tuples stored on the same disk page—the authors propose an adaptive sampling method. At each step of the sampling process, the adaptive method uses cross-validation to estimate the sampling error in the equi-depth histogram that is computed from the disk pages sampled so far. If the error exceeds the target, the sampling process continues; otherwise, sampling stops.

Although the foregoing sampling approaches are currently used in commercial systems such as DB2 and Microsoft SQL Server, they raise the question of whether the sample could instead be used directly for approximate query answering. Moreover, any potential histogram-related error bounds for an individual query must be inflated to reflect the additional error due to sampling, and the resulting overall error bounds will then be probabilistic in nature. In addition, any optimality assertions about a histogram—see, e.g., the algorithms in Section 3.4.2—will then be compromised.

Even when a complete pass through the data is acceptable, many published algorithms have an unacceptably large $\Omega(M)$ or $\Omega(N)$ space requirement (under the datacube or relational access model, respectively) for constructing the histogram. One possibility for dealing with this situation is to use a sample, as discussed above. For example, an early histogram implementation in DB2 piggybacked a reservoir-sample computation [283] on top of a complete database scan by the RUNSTATS statistics-collection utility, and then computed a "compressed histogram" (see Section 3.4.1) from the sample. Indeed, any synopsis computed from a full scan can be used in this manner, provided that the histogram can be computed directly from the synopsis without having to first generate the complete (approximated) dataset. E.g., Gilbert et al. [128] use sketches to limit the space needed to construct a histogram; see Section 3.7.1. Again, this approach raises the question as to whether the underlying synopsis could instead be used directly for query answering. A related strategy is to first take a sample, which is used to determine the histogram buckets, and then compute the bucket statistics via a complete data scan. Under this strategy, there is no sampling error for the bucket statistics, since they are computed exactly. The buckets themselves may be chosen suboptimally, however, and hence any optimality guarantees are weakened

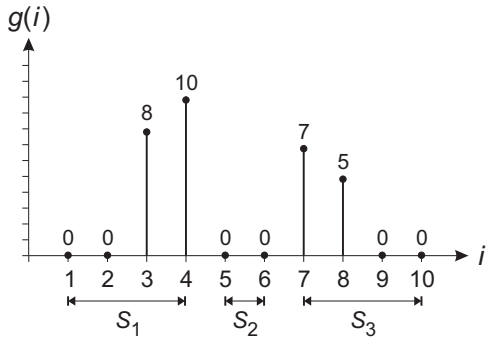
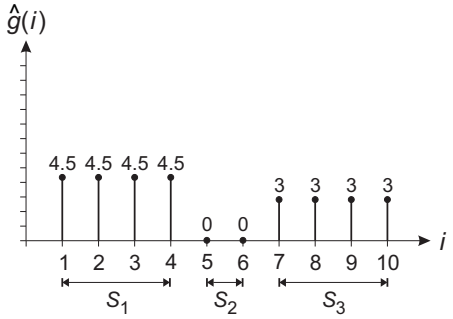


Fig. 3.4 A discrete dataset and three buckets

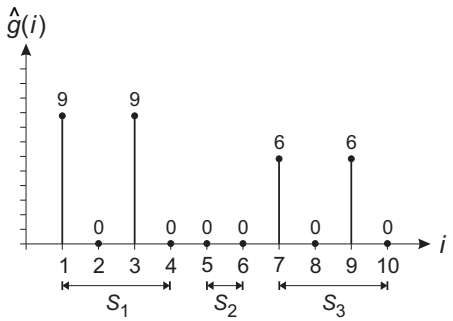
(but not as much as when bucket statistics are also estimated). An alternative strategy is to use a disk-based algorithm to compute the histogram. There has been very little research in this area; one exception is literature on disk-based algorithms for computing quantiles (as needed, for example, to compute an equi-depth histogram); see [144]. Of course, use of a disk-based algorithm will typically incur a significant increase in the time required for histogram construction. Finally, the memory requirement can be reduced by computing a suitable approximation to the desired histogram—we will give many examples of this approach in the sequel. This latter strategy can be extremely effective.

3.3 Estimation Schemes

We focus on the datacube access model and assume that we have partitioned $[1,M]$ into mutually disjoint buckets $S_1, S_2, \ldots, S_B$, where each S_j is of the form $S_j = \{l_j, l_j+1, \ldots, u_j\}$. For each bucket S_j, we want to summarize the array of numbers $V_j = \big(g(l_j), g(l_j+1), \ldots, g(u_j)\big)$ so that the summary is compact, and can later be exploited to reconstruct V_j with low error. For "sparse" datasets, many of the entries in V_j might equal 0. For example, Figure 3.4 shows a set of discrete data with $\mathscr{U} = [1,10]$ and $V = (0,0,8,10,0,0,7,5,0,0)$. If the function g is interpreted as a frequency function, then the dataset can be viewed as containing thirty data points, with eight points having value 3, ten points having value 4, and so forth. Alternatively, the i values might be interpreted as indicating time, and $g(i)$ might



(a) Continuous-value assumption



(b) Uniform-spread assumption

Fig. 3.5 Histograms using uniform estimation schemes

denote inches of rainfall (rounded to the nearest inch) on day i. The domain $\mathscr{U}$ has been partitioned into three buckets for purposes of histogram creation: $S_1 = \{1,2,3,4\}$, $S_2 = \{5,6\}$, and $S_3 = \{7,8,9,10\}$. The value sets are $V_1 = (0,0,8,10)$, $V_2 = (0,0)$, and $V_3 = (7,5,0,0)$.

3.3.1 Uniform Schemes

The continuous-value assumption described previously can be applied essentially unchanged to discrete data. For each bucket S_j, store the quantity $g_j^+ = \sum_{i \in S_j} g(i)$. Then, during approximate query processing, estimate the contribution $\alpha_j = \sum_{i \in S_j \cap Q} g(i)$ of bucket S_j to the query result as $\hat{\alpha}_j = (|S_j \cap Q|/|S_j|)g_j^+$, where $|W|$ denotes the number of elements in set W. Note that the continuous-value assumption is equivalent to approximating the function g within a bucket S_j as $\hat{g}(i) = g_j^+/|S_j|$ for each $i \in S_j$, and then approxi-

mating the query answer as $\hat{\alpha} = \sum_{i \in Q} \hat{g}(i)$.

An alternative estimation scheme, called the *uniform-spread* assumption, explicitly takes into account the discrete nature of the data, and also facilitates approximate answering of JOIN and GROUP BY queries (see Section 3.6). For each bucket S_j, store both the sum g_j^+ defined above, as well as d_j, the number of "positive values" $i \in S_j$ such that $g(i) > 0$.[2] Then approximate V_j by first assuming that the positive values are evenly spread across the range of S_j. That is, approximate the true set P_j of positive values, e.g., by the set $\hat{P}_j = \{ l_j, l_j + k_j, l_j + 2k_j, \ldots, l_j + (d_j - 1)k_j \}$, where $k_j = \lfloor (u_j - l_j + 1)/d_j \rfloor$ and $\lfloor x \rfloor$ denotes the largest integer less than or equal to x. Finally, approximate the function g within S_j by setting

$$\hat{g}(i) = \begin{cases} g_j^+/d_j & \text{if } i \in \hat{P}_j; \\ 0 & \text{if } i \notin \hat{P}_j. \end{cases}$$

The approximate query answer is then $\hat{\alpha} = \sum_{i \in Q} \hat{g}(i)$. Note that if the data is dense, in that $g(i) > 0$ for all $i \in S_k$, then the uniform-spread assumption coincides with the continuous-value assumption. One disadvantage of the uniform-spread assumption is that, in the discrete-data setting where $g(i)$ is the frequency of value i, one must compute the number of distinct values present in each bucket, which is nontrivial under the relational model of data storage. Exact computation is typically too expensive, so that the number of distinct values in a list of discrete data points must be estimated based on either a hash-sketch, which uses limited memory but requires a scan of the data—see Section 5.4—or on a sample of the data points in a bucket—see [44, 156, 258] and Section 2.6.2. Such estimation can noticeably increase the time and space requirements for histogram construction, and the sampling-based approach, in particular, can yield inaccurate estimates even at reasonable sample sizes.

Figure 3.5 displays histograms for the dataset in Figure 3.4 based on the continuous-value and uniform-spread assumptions. For the range-sum query with $Q = \{ 3, 4, 5, 6, 7 \}$, for which the exact result is $\alpha = 25$, the two schemes yield respective estimates of $\hat{\alpha} = 12$ and $\hat{\alpha} = 15$, respectively. Although the uniform-spread assumption outperforms the continuous-value assumption in this example, a fairly comprehensive set of experiments conducted by Wang

[2] If $g(i)$ is the frequency of value i, then d_j is simply the number of distinct values that appear in S_j.

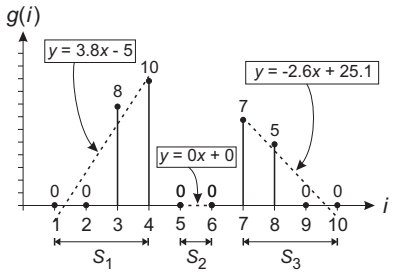


Fig. 3.6 Linear spline-based estimation scheme

and Sevcik [288] indicate that, in general, the opposite result holds true for OLAP queries (both one-dimensional and multi-dimensional) over discrete data.

3.3.2 Spline-Based Schemes

The foregoing schemes approximate the g-values in a bucket S_j, i.e., the V_j array, via a simple average. A more elaborate approach fits a linear function to these values, in an attempt to capture the values more accurately. Such a histogram is sometimes called a *spline* histogram [209, 295], and is closely related to the notion of a "frequency polygon" from the statistical literature [270, Ch. 4]. Instead of storing a single average value for a bucket S_j, a spline histogram stores the intercept β_j and slope γ_j of the fitted line. The contribution from S_j is now estimated as $\hat{\alpha}_j = \sum_{i \in S_j \cap Q} (\gamma_j i + \beta_j)$. For each bucket, the line can be fit using a standard least-squares algorithm. This estimation scheme is well suited to "turnstile" streaming scenarios (see Section 3.7.1): for a fixed set of buckets, the fitted line can be incrementally updated as the g-values in the bucket change due to updates; see, e.g., Zhu and Larson [298]. Figure 3.6 illustrates the use of the spline estimation scheme for the dataset of Figure 3.4. For the range-sum query given above, with $Q = \{3,4,5,6,7\}$, the approximate answer is $\hat{\alpha} = 23.5$, yielding a lower error than the uniform-spread and continuous-value schemes.

Zhang and Lin [295] found that the approximation can be improved further by modifying the line-fitting procedure to enforce the constraints $\sum_{i \in S_j} (\gamma_j i + \beta_j) = \sum_{i \in S_j} g(i)$ and $\sum_{i \in S_j} i(\gamma_j i + \beta_j) = \sum_{i \in S_j} i g(i)$. When the function g coincides with the frequency function f, then the constraints require

that both the number and the sum of the data points in S_k must match their true values. (The method in [295] actually assumes that the number d_j of distinct values in S_j is known, and initially applies a uniform-spread assumption. Thus the sums appearing on the left side of the above constraints are actually over the set $\hat{P}_j$ rather than S_j, where $\hat{P}_j$ is defined as in Section 3.3.1.)

This approach can, in principle, be extended by using polynomial rather than linear functions to model the data points. It is not clear whether such additional complication is worth the effort, however, since then more coefficients must be stored for each bucket, to the detriment of the number of buckets that can be maintained. Also, the fitting process becomes ever more expensive. Even in the case of linear splines, other techniques, such as the 4LT method discussed below, seem to have superior performance in general.

3.3.3 Four-Level Trees

Buccafurri et al. [37] have provided an extremely effective estimation scheme that involves storing, for each bucket S_j, the usual bucket total g_j^+, together with an additional integer that encodes a description of the data distribution in the bucket. This description can be represented as a four-level tree (4LT) that hierarchically represents partial sums of the g function. The experiments in [37] show that the increase in accuracy of the 4LT representation far outweighs any accuracy loss caused by the decrease in the number of buckets that will fit into allocated memory. The technique has been developed for functions g whose domain is a subset of the integers, but the ideas can potentially be extended to other g functions.

We illustrate the method using an example taken from [37], where integers are assumed to comprise 32 bits. Consider a fixed bucket S_k consisting of 16 values, specifically,

$$V_k = (7,5,18,0,6,10,0,6,0,6,9,5,13,0,8,7).$$

When V_k is divided into j (≥ 1) equally sized segments, denote by $\sigma_{i/j}$ the sum of the values in the ith such segment, so that $\sigma_{1/1} = 100$ is the sum of all of the values, $\sigma_{2/4} = 6+10+0+6 = 22$ is the sum of the values in the second quarter of V_k, and so forth. The 4LT for this bucket is illustrated in Figure 3.7. The nodes at the jth level of the tree ($1 \leq j \leq 4$) correspond to the partial sums $\sigma_{1/n_j}, \sigma_{2/n_j}, \ldots, \sigma_{n_j/n_j}$, where $n_j = 2^{j-1}$. The root node at

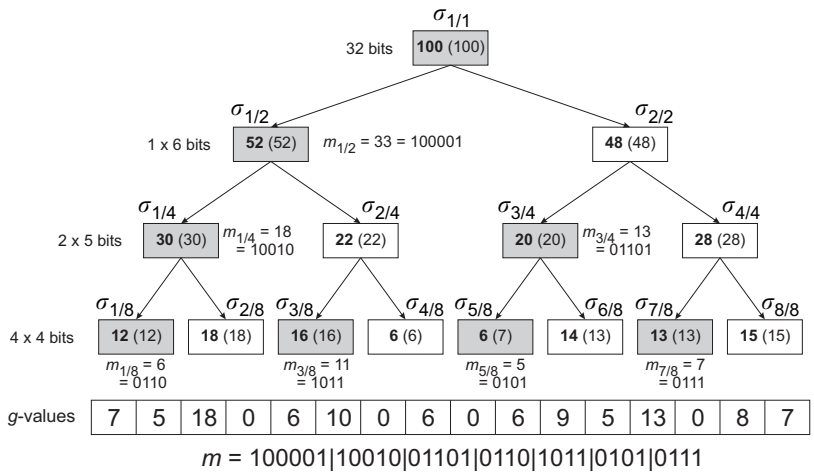


Fig. 3.7 4LT representation of g-values in a bucket

level 1 is represented exactly, using 32 bits. At level $j > 1$, the value of σ_{i/n_j} is represented approximately, using b_j bits, where $b_2 = 6$, $b_3 = 5$, and $b_4 = 4$. More precisely, we compute and store the b_j-bit integer

$$m_{i/j} = \left\langle \frac{\sigma_{i/j}}{\sigma^*_{i/j}}(2^{b_j} - 1) \right\rangle,$$

where $\sigma^*_{i/j}$ is the (exact) σ value at the parent of the node corresponding to $\sigma_{i/j}$, and $\langle x \rangle$ denotes the integer value obtained from x by rounding. Then, at query time, the value of $\sigma_{i/j}$ is approximated as

$$\hat{\sigma}_{i/j} = \left\langle \left(\frac{m_{i/j}}{2^{b_j} - 1} \right) \hat{\sigma}^*_{i/j} \right\rangle,$$

where $\hat{\sigma}^*_{i/j}$ is the approximated[3] value of $\sigma^*_{i/j}$. For example, $\sigma_{5/8} = 6$ is approximated using four bits by computing

$$m_{5/8} = \left\langle \frac{\sigma_{5/8}}{\sigma_{3/4}}(2^4 - 1) \right\rangle = \left\langle \frac{6}{20}(15) \right\rangle = 5 = 0101_2.$$

[3]The values are approximated starting at the root—where the "approximation" is trivial—so that the value of $\hat{\sigma}^*_{i/j}$ is available when $\hat{\sigma}_{i/j}$ is to be computed.

Then $\sigma_{5/8}$ is approximated[4] as

$$\hat{\sigma}_{5/8} = \left\langle \left(\frac{m_{5/8}}{15} \right) \hat{\sigma}_{3/4} \right\rangle = \langle (5/15)(20) \rangle = 7.$$

In the figure, the approximated value of each $\sigma_{i/j}$ is shown in parentheses.

The final 32-bit integer that is stored with the bucket is obtained by concatenating the $m_{i/j}$ values corresponding to the shaded nodes in the figure. The $m_{i/j}$ values for the non-shaded nodes do not need to be stored, because the corresponding $\sigma_{i/j}$ values can be approximated directly by subtracting the approximated σ value for the node's sibling from the approximated σ value for the node's parent. E.g., we approximate $\sigma_{6/8}$ as $\hat{\sigma}_{6/8} = \hat{\sigma}_{3/4} - \hat{\sigma}_{5/8} = 20 - 7 = 13$. To actually estimate the contribution of the bucket to a range-sum query, we simply add up the $\hat{\sigma}$ values that cover the elements contained in Q; we try to use the $\hat{\sigma}$ values at the highest levels of the tree, since these have the greatest accuracy. If necessary, the continuous-value assumption is used to estimate g-values at a finer granularity than is provided by the leaf nodes of the 4LT. For example, if the query region Q overlaps the first thirteen values in the bucket, then we estimate the bucket's contribution as $\hat{\sigma}_{1/2} + \hat{\sigma}_{3/4} + 0.5\hat{\sigma}_{7/8} = 52 + 20 + (0.5)(13) = 78.5$, where the first two terms estimate (exactly, as it happens) the contributions of the first eight plus the next four g-values, and the last term estimates the thirteenth g-value using the continuous-value assumption. The true answer is 85, so the error in the estimate is approximately 8%.

3.3.4 Heterogeneous Estimation Schemes

Ioannidis [179] notes that histogram accuracy can be improved by allowing different estimation schemes in different buckets. Although such histograms are relatively expensive to compute, they may be appropriate when high accuracy is needed and storage space for the histogram is limited.

Wang and Sevcik [288] developed a specific approach along these lines, defining a "hierarchical model-fitting" (HMF) histogram that uses one of the continuous-value, uniform-spread, or 4LT schemes for a given bucket, depending on which scheme best approximates the data in the bucket. The

[4] This example may seem somewhat artificial, because we can already represent the integer $\sigma_{5/8} = 6$ using only four bits; identifying such "simple" cases, however, would require at least one more bit, and would complicate the algorithm further.

criterion used to compare schemes is inspired by the "minimum description length" principle from statistics [265], which attempts to balance model complexity and accuracy in a principled manner. As discussed in [288], a scheme M_2 for a bucket is preferred over a scheme M_1 if $k_2 \leq k_1$ and $D(p\|q_2) < D(p\|q_1)$, where k_l is the number of "parameters" of M_l, that is, the number of summary statistics that scheme M_l requires to be stored for the bucket, and $D(p\|q_l)$ is a "relative entropy" measure of the distance between the true and estimated g-values in the bucket. In more detail, the g-values (assumed nonnegative) in bucket S_j can be normalized to a probability mass function p by dividing each value $g(i)$ by the bucket sum g_j^+. The approximated g-values under scheme M_l can similarly be normalized to a probability mass function q_l. The relative-entropy distance of q_l from p may then be defined as $D(p\|q_l) = \sum_{i \in S_j} p(i) \log\big(p(i)/q_l(i)\big)$. (The authors actually use a slightly different, more complex definition that more effectively measures the distance between the two probability mass functions when the data in a bucket is sparse.) If $D(p\|q_2) < D(p\|q_1)$ but $k_2 > k_1$, then M_2 is preferred to M_1 if the benefit/cost ratio

$$\rho = \frac{D(p\|q_1) - D(p\|q_2)}{k_2 - k_1}$$

is sufficiently large, i.e., if the decrease in distance per additional parameter is large enough.

Buccafurri et al. [32] also propose the use of different estimation methods in different buckets, in the context of a histogram having a recursive bucket structure (see Section 3.4.3). They also use a greedy algorithm to determine the overall design of the histogram.

Recently, Kanne and Moerkotte [198] have proposed a heterogeneous estimation scheme that uses several new types of buckets. The simplest buckets store the number of distinct values and, instead of the traditional average bucket frequency, the "q-middle" frequency. This latter frequency is defined for bucket S as $\sqrt{\big(\min_{i \in S} g(i)\big)\big(\max_{i \in S} g(i)\big)}$. In addition, a bucket might also store the average frequency and the frequencies of the values that define the bucket boundaries. The idea is to approximate each frequency in the bucket by the q-middle frequency for point queries or range queries with relatively short ranges, and by the average frequency for range queries with longer ranges, using a threshold on range length to determine which estimation scheme to use. Several other bucket types tailored to range queries

are also proposed, as well as a bucket type that stores exact frequencies in a compressed form. The goal is to control the "q-error" of the histogram; see Section 3.4.2 for definitions.

3.3.5 A Probabilistic Scheme

Buccafurri et al. [33] follow an approach that is quite different from the foregoing methods, using a probability model over datasets. Suppose that the total sum s of the g-values in the dataset and the total number d of positive g-values are known. Fix a range-sum query Q and consider the class of histograms such that the summary data stored in each bucket comprises the average of the g-values in the bucket, together with the number of positive g-values in the bucket. Denote by $\mathscr{G}$ the finite set of all nonnegative functions defined on $\mathscr{U}$ such that the sum of the g-values is s and exactly d of the g-values are positive. For a given histogram H, denote by $\mathscr{G}_H$ the subset of functions in $\mathscr{G}$ that are "compatible" with H; that is, $g \in \mathscr{G}_H$ if and only if the partitioning of g according to the buckets of H yields summary statistics that coincide exactly with those in H. The maximum-entropy-style method in [33] imposes a probability distribution over $\mathscr{G}_H$ in which each of its elements is equally likely. This distribution then induces a probability distribution over the answer to Q. The authors provide closed-form formulas for the mean and variance of this latter distribution—the mean is the estimated query answer and the variance is taken as a measure of the precision of the answer.

Importantly, this approach can gracefully incorporate additional knowledge about the data by restricting the set $\mathscr{G}$ appropriately. The authors focus on knowledge that comprises either a lower bound on the number of positive g-values lying within the query range or a lower bound on the number of g-values in the range that are equal to 0. Under many bucketing schemes, the leftmost and the rightmost g-values in a bucket are forced to be non-zero, and this knowledge can lead to more precise query-answer estimates than are obtained under, e.g., the continuous-value or uniform spread assumptions.

3.3.6 Error Estimates for Individual Queries

The bucketing schemes in Section 3.4.2 below provide, as a byproduct, an estimate of the aggregate error of the entire histogram, e.g., the maximum error for a point query, or the average L_2 error over all point queries, or even

a workload-weighted average error. In a similar spirit, Donjerkovic and Ramakrishnan [91] provide a sampling-based scheme for assessing the worst-case estimation error for range queries in an equi-depth histogram. None of these estimates, however, give the user precise feedback about the estimation error for the particular query that is being evaluated. An estimate such as the maximum error over any point query is likely to be very conservative with respect to a specific query. The probabilistic method of the previous section returns a "precision number" in the form of a variance, but this number is more an indicator of the degree of potential error than a metric of the true error of the approximate answer relative to exact answer that is computed over the actual g-values that comprise the dataset.

To address the precision issue, one simple scheme—c.f. Poosala et al. [256]—stores the maximum and minimum g-value in a bucket, along with the usual summary statistic(s), such as the average g-value. Then a rough precision estimate can be returned to the user. For example, consider a range-sum query α that is approximately answered using a given histogram, where we summarize each bucket by its average g-value. Suppose that all but one bucket is contained within the query range. Also suppose that, for the partially covered bucket S_j, values $g(i), g(i+1), \ldots, g(i+n-1)$ lie within the query range. Then, since the contributions to the query answer from the completely covered buckets are exact, we know that $\hat{\alpha} - L \leq \alpha \leq \hat{\alpha} + U$, where $L = n(g_j^{\text{avg}} - g_j^{\text{min}})$ and $U = n(g_j^{\text{max}} - g_j^{\text{avg}})$. If the histogram is effective at minimizing the variability of g-values in a bucket, then g_j^{max} and g_j^{min} will be close to g_j^{avg}, and the bounds will be useful. Jagadish et al. [186] suggest a similar idea, and Read et al. [263] elaborate on the approach in the setting of more complex histograms. Of course, the additional information inflates the per-bucket memory requirements, so that a histogram that incorporates the foregoing scheme will have fewer buckets, and hence less accuracy. This trade-off seems worthwhile, however, in order to provide users with feedback on the quality of their approximated query-answers.

3.4 Bucketing Schemes

The scheme used to assign data points to buckets for purposes of summarization is crucially important to achieving good approximate query answers. In this section, we describe a range of bucketing schemes for 1D-histograms,

ranging from heuristic to provably optimal. We focus, as usual, on general range-sum queries under the datacube access model, i.e., estimating quantities of the form $\alpha = \sum_{i \in Q} g(i)$, where $Q \subseteq \mathscr{U} = [1, M]$ for some $M \geq 1$; Section 3.7.2 describes techniques especially suitable for real-valued data.

As seen from the discussion in Section 3.3, the crux of histogram-based estimation is to approximate the g-values in a bucket with a few summary numbers, such as an average or the slope and intercept of a line. As shown in early work by Ioannidis and Christodoulakis [178, 182] (in the context of using histograms to estimate join sizes for categorical data), it follows that good (and sometimes optimal) approximations are often obtained by grouping together elements of $\mathscr{U}$ having similar g-values. For example, a *serial* histogram first sorts the elements of $\mathscr{U}$ by g-value to obtain an ordered sequence $U' = (i_1, i_2, \ldots, i_M)$ with $g(i_1) \leq g(i_2) \leq \cdots \leq g(i_M)$, and then assigns elements of $\mathscr{U}$ to buckets by partitioning U' into disjoint segments. Arbitrary bucketing by g-values is impractical, however, because then an index is needed to map i-values to buckets. We therefore restrict attention to bucketing schemes that partition the domain $[1, M]$ into disjoint segments.

3.4.1 Heuristic Schemes

We first consider various heuristic schemes—specifically the equi-width, equi-depth, "compressed," and "maxdiff" bucketing methods—that have been adopted in commercial database systems for use within the query optimizer because of their relatively low construction costs. These methods do not come with optimality guarantees: they typically perform well on average, occasionally providing a seriously inaccurate result for some unlucky query.

Equi-Width Histograms

The equi-width histogram technique described in Section 3.1.1 can be applied to discrete data as well as continuous data. The histogram can be computed in a single pass with $O(1)$ cost per data item and the space complexity is $O(B)$, under either the datacube or relational access models. (Recall the definitions of these models from Section 3.2.2, as well as the discussion on construction costs.)

Equi-Depth Histograms

The equi-depth bucketing scheme of Section 3.1.1 can also be applied to discrete data. In the current setting, the goal is to have the sum of g-values be the same for every bucket, i.e., $\sum_{i \in S_j} g(i) = \sum_{i \in S_k} g(i)$ for all j and k. When defining the buckets, some minor technical complications can arise when certain data values are very frequent, so that multiple quantiles coincide; see [252] for details.

Under the datacube model, and assuming that the sum $g^+ = \sum_i g(i)$ of g-values is known, the list of g-values is simply scanned once to determine the bucket boundaries, requiring $O(B)$ space in total; if g^+ is unknown, then an additional scan of the data is required. Under the relational model, computing the histogram corresponds to finding $B-1$ equally-spaced quantiles. A naive algorithm scans the data into memory (if possible), sorts it, and reads off the quantiles, incurring an $O(N)$ space cost and an $O(N \log N)$ time cost. Alternatively, a disk-based sort can be used, but this requires multiple passes over the data. Typically these straightforward approaches are impractical, but can be applied to a sample of the data. The disk-based quantile algorithm in [144] assumes that the dataset is stored in K disk-resident blocks, and assumes a fixed-size memory buffer for holding data. Using algorithms that employ an efficient median-finding algorithm, the authors show how to compute an exact set of buckets with $O(K \log B)$ I/O cost and $O(K \log B)$ CPU cost; these costs are still quite high. In practice, as discussed in Section 3.1.1, algorithms for computing approximate quantiles must be employed in order to achieve acceptable space and time complexity. For example, the one-pass algorithm of Greenwald and Khanna [132] has a worst-case time complexity $O\big(N\varepsilon^{-1}\log(\varepsilon N)\big)$ and a space complexity of $O\big(\varepsilon^{-1}\log(\varepsilon N)\big)$. The recent algorithm of Zhang and Wang [296] requires more space, $O\big(\varepsilon^{-1}\log^2(\varepsilon N)\big)$, but less time, $O\Big(N\log\big(\varepsilon^{-1}\log(\varepsilon N)\big)\Big)$.

Singleton-Bucket Histograms

An *SB-histogram* places elements of $[1,M]$ having extremely large or small g-values into singleton buckets, and reverts to a classical bucketing scheme to represent the remaining data. E.g., an *end-biased* histogram [178, 182] places the B_1 elements of $[1,M]$ having the largest g-values and the B_2 elements

having the smallest g-values in singleton buckets, and uses a single bucket to represent the remaining values (via the continuous-value assumption). When $B_2 = 0$, such a histogram is called a *high-biased* histogram. A *compressed* histogram [258] is similar to a high-biased histogram, except that an equi-depth histogram is used to represent the g-values for the $M - B_1$ non-singleton elements of $[1,M]$; in the usual implementation, the g-value for a singleton bucket must exceed g^+/B, where $g^+ = \sum_i g(i)$ as before.

Under the datacube model, two passes through the data are needed, each with $O(1)$ processing cost per item: one to identify the k largest g-values and compute the sum of the remaining g-values, and one to compute the equi-depth histogram for the non-large g-values. The overall space requirement for construction is $O(B)$, as with the equi-depth histogram.

Under the relational model, both quantiles and highly frequent values must be computed. Computation of quantiles is the same as for an equi-depth histogram. The frequencies of the most frequent values usually can be estimated with high precision from a simple uniform sample, as in [258]; Cohen et al. [55] give a more elaborate scheme for estimating the k highest frequency values, and Gemulla et al. [117] give a sampling algorithm that provides unbiased, low-variance estimates of both high and low frequencies, at the expense of storing "tracking counters" in the sample. Note in this connection that the more skewed the frequency distribution, the more important it is to estimate the frequent values and the easier this estimation problem becomes. One simple algorithm, proposed in [258], is to take a small sample and create a counter for each distinct value in the sample. The exact frequencies of these distinct values are then computed exactly based on a full scan of the data. The idea is that highly frequent values with appear in the sample with high probability. For example, if 10^6 data points have values distributed according to a Zipf(0.86) distribution (roughly an "80-20" law), then a 0.1% sample will contain the 10 most frequent values with probability approximately 99.9%. Alternatively, more sophisticated full-scan limited-memory algorithms can be used; see, e.g., [46, 65], as well as Section 5.3.4.1.

Maxdiff Histograms

A *maxdiff* histogram [258] with B buckets partitions the sequence $[1,M]$ by placing a bucket boundary between elements i and $i+1$ if the difference

$|g(i+1)-g(i)|$ is among the $B-1$ largest such differences. For the datacube model, only a single scan is needed, with $O(1)$ processing cost per item, and the total space requirement is $O(B)$. For the relational model, the data-values must be converted to the datacube format and then processed as above. Such a conversion requires a complete sort of the data, which is usually unacceptable in practice. For this reason, the maxdiff histogram for relational data is typically computed from a sample, as in [258].

Generalizing Bucketing Schemes

The foregoing bucketing schemes can be generalized by imposing a homogeneity requirement not on the g-values within a bucket, but on other data properties. For example, when the data is sparse, so that many g-values are equal to 0 and the uniform-spread estimation scheme is applicable, then similarity of g-values within a bucket is not the only criterion for a good bucketing scheme; similar spacing between successive positive elements in a bucket is also desirable. For $i \in [1,M]$ with $g(i) > 0$, define the *spread* of i as $s(i) = \min\{\, j \in [i+1,M] : g(j) > 0 \,\} - i$, where we take $s(i) = M - i$ if $g(j) = 0$ for $j \in [i+1,M]$. Similarly define the *area* of i as $a(i) = g(i)s(i)$. We can then replace the function g in the definition of the above bucketing scheme by the functions s or a [258]. That is, the bucketing schemes try to make the spreads or areas within each bucket as homogeneous as possible. The motivation behind the use of area is to balance the goals of equalizing g-values and equalizing spreads. The cumulative function $c(i) = \sum_{j=1}^{i} g(i)$ has also been proposed. Adapting the terminology in [258], one can then talk about, e.g., a maxdiff(g), maxdiff(s), maxdiff(c), or maxdiff(a) histograms, depending on whether the function g, s, c, or a is used when determining the buckets.[5] (As usual, the area scheme reduces to the g-value scheme as the data becomes dense.) Virtually all other bucketing schemes can be similarly generalized, and even unified in some cases. For example, we can view equi-width and equi-depth histograms as "equi-sum(s)" and "equi-sum(g)" histograms, where an equi-sum(h) histogram selects buckets so that the bucket sum $h_j^+ = \sum_{i \in S_j} h(i)$ is the same for each j. Under the datacube model, the bucketing algorithms are virtually unchanged, except that a lookahead buffer

[5] The original paper [258] considered the special case where g equals the frequency function f, and referred to frequency, spread, and area as the possible "source parameters" for determining buckets.

of length 1 is required to compute spreads. Under the relational model, the spreads must be estimated. Poosala et al. [258] found that computing spreads from a data sample yielded acceptable results; more sophisticated estimates of spreads can be derived from estimates of the number of distinct values in a bucket; see the discussion of equi-depth histograms in Section 3.3.1.

The experiments in [258] indicate that, for moderately sparse data, the maxdiff(a) scheme yields the best accuracy, but the accuracy of the maxdiff(g), compressed(g), and compressed(a) schemes is almost as good. Because of their good balance of accuracy and practicality, versions of the maxdiff and compressed histograms are currently used in SQLServer and DB2 products; see [180, 183]. As noted in Giannella and Sayrafi [121], however, the experimental results in the early literature on heuristic bucketing schemes are far from definitive, and no heuristic scheme uniformly dominates the others.

3.4.2 Schemes with Optimality Guarantees

In light of the purely empirical performance studies for the foregoing heuristic schemes, and the limitations of these studies, the question naturally arises of whether there exist efficient algorithms for determining a set of provably "optimal" or "near-optimal" histogram buckets that maximize estimation accuracy, subject to an upper bound on the histogram size. To formulate the problem more precisely, one must specify the estimation scheme used for the buckets, the class of bucketing schemes allowed, and the accuracy criterion. A large literature has grown up around this topic, which we briefly summarize in this section. Unless specified otherwise, all of the algorithms assume the datacube access model, so that the input g-values are given in the form $g(1), \ldots, g(M)$. As discussed previously, all of these algorithms assume at least one complete pass through the data; if sampling is necessary, then the optimality statements will not hold in general, because the bucket boundaries will not be based on the complete data. Another running assumption is that the maximum g-value R is such that $\log R = O(\log M)$, which is typically the case when g-values correspond to frequency counts. When the g-values are real, they are assumed to be of bounded precision, so that they can be rescaled to integers and satisfy the above assumption.

A Basic Dynamic-Programming Approach

We first describe the basic dynamic programming (DP) method of Jagadish et al. [186], which was the first optimality result to appear in the database literature.[6] This basic algorithm—which focuses on minimizing an L_2-error metric—typifies a broad class of optimal-histogram algorithms and is perhaps the easiest to understand. As will be seen, the cost of the algorithm is often unacceptable in practice, so much subsequent work has focused on improving the basic algorithm either by using approximate DP techniques or by focusing on more tractable error metrics such as L_∞.

The (somewhat limited) class of histograms considered determines buckets by decomposing the domain $\mathscr{U} = [1, M]$ into disjoint segments and estimating the g-values for the elements within a bucket using the continuous-value assumption; that is, for each $i \in [1, M]$, the histogram estimates $g(i)$ as the average g-value for the bucket containing i. The accuracy metric for the histogram is the (squared) L_2 distance between the vector of actual and estimated g-values: $E(H) = \sum_{i=1}^{M} \big(g(i) - \hat{g}_H(i)\big)^2$, where $\hat{g}_H(i)$ is the estimated g-value for $i \in [1, M]$, based on the histogram H. Such a histogram is called a *v-optimal* histogram in [183].

The DP approach exploits the fact that the error metric $E(H)$ decomposes into the sum of errors for the individual buckets. Fix $k \in [2, M]$, $j \in [1, M]$ and $i \in (1, j]$, and consider an arbitrary but fixed k-bucket histogram over $[1, j]$ whose rightmost bucket is $S_k = [i, j]$. By the decomposability of the error metric, the L_2 error $E_k[1, i, j]$ of this histogram can be expressed as $E_k[1, i, j] = E^0_{k-1}[1, i) + L_2[i, j]$, where $E^0_{k-1}[1, i)$ is the contribution to the L_2 error from the first $k-1$ buckets over $[1, i)$ and $L_2[i, j]$ is the contribution from bucket S_k. Clearly, a better histogram is obtained by adjusting the first $k-1$ buckets to coincide with those of the optimal $(k-1)$-bucket histogram over $[1, i)$. Denote the error of this latter histogram by $E^*_{k-1}[1, i)$, so that the improved version of the original k-bucket histogram has overall error $E^*_{k-1}[1, i) + L_2[i, j]$. This is the minimum possible error for a k-bucket histogram over $[1, j]$ having rightmost bucket $[i, j]$. Thus the best k-bucket histogram over $[1, j]$ is obtained by choosing a histogram of this latter type, using the best possible value of i, i.e., choosing i to minimize $E^*_{k-1}[1, i) + L_2[i, j]$. The error $E^*_k[1, j]$ for this

[6] As pointed out by Karras and Mamoulis [203], the algorithm can be viewed as a special case of a curve-approximation scheme due to Bellman [18].

optimal histogram thus satisfies the equation

$$E_k^*[1,j] = \min_{i\in(1,j]} E_{k-1}^*[1,i) + L_2[i,j]. \tag{3.1}$$

This is the Bellman optimality equation of dynamic programming, and we denote by $I_{k,j}^*$ the value of i that minimizes the right side; that is, $I_{k,j}^*$ is the position of the lower boundary of the rightmost bucket in an optimal k-bucket histogram on $[1,j]$. For convenience, also set $T_{k,j}^* = E_k^*[1,j]$, so that $T_{k,j}^*$ is the L_2 error of the optimal k-bucket histogram on $[1,j]$.

The DP algorithm proceeds by computing—via (3.1)—and storing $T_{k,j}^*$ and $I_{k,j}^*$ for $k = 1,2,\ldots,B$ and $j = 1,2,\ldots,M$. Observe in this connection that the L_2 errors can be computed efficiently by precomputing the quantities $R_i^{(1)} = \sum_{l=1}^{i} g(i)$ and $R_i^{(2)} = \sum_{l=1}^{i} g^2(i)$ for each $i \in [1,M]$. Then the L_2 error for a bucket with end points a and b is computed as follows, where $m = b-a+1$:

$$\begin{aligned} L_2[a,b] &= \sum_{i=a}^{b} \Big(g(i) - (1/m) \sum_{j=a}^{b} g(j) \Big)^2 \\ &= \sum_{i=a}^{b} g^2(i) - (1/m) \Big(\sum_{j=a}^{b} g(j) \Big)^2 \\ &= \big(R_b^{(2)} - R_{a-1}^{(2)} \big) - (1/m) \big(R_b^{(1)} - R_{a-1}^{(1)} \big)^2. \end{aligned}$$

To start the computation, we set $T_{1,j}^* = E_1^*[1,j] = L_2[1,j]$ for $j \in [1,M]$; we can set each $I_{1,j}^*$ to an arbitrary value. After every $T_{i,j}^*$ and $I_{i,j}^*$ has been computed, the lower bucket boundaries for the optimal B-bucket histogram are determined in order of decreasing position as

$$\begin{aligned} i_B &= I_{B,M}^*, \\ i_{B-1} &= I_{B-1,i_B-1}^*, \\ i_{B-2} &= I_{B-2,i_{B-1}-1}^*, \\ &\vdots \\ i_2 &= I_{2,i_3-1}^*, \\ i_1 &= 1. \end{aligned}$$

The L_2 error of the optimal histogram is simply $T_{B,M}^*$.

The time and space complexity for computing the $R_i^{(1)}$ and $R_i^{(2)}$ constants is $O(M)$. Computation of each $(I_{k,j}^*, T_{k,j}^*)$ pair requires $O(M)$ operations, cor-

responding to exploring all possible values of i for the minimization in Bellman's equation. There are $O(MB)$ such pairs, which also must be stored. Thus the overall time complexity is $O(M^2B)$—with multiple passes required over the data—and the space complexity is $O(MB)$. The algorithm can be extended to other accuracy metrics, but the time complexity for the general case is $O(M^3B)$; see [203], as well as Lin and Zhang [217]. (The latter paper gives a DP algorithm for spline histograms under a uniform-workload-based error metric for range queries.) Guha [134] provides a general technique for reducing the space complexity of synopsis construction algorithms, and shows how the technique can be specialized to reduce the space complexity of the foregoing algorithm from $O(MB)$ to $O(M)$. The basic idea is to use a divide-and-conquer strategy for placing bucket boundaries, where a dynamic program is used to find the dividing point; Guha shows that this program has smaller space complexity than the original dynamic program, and an overall space reduction is achieved because the resulting set of subproblems can share the same working space.

A key strength of the DP approach is that it extends straightforwardly to a broad range of different error measures, for example,

- the L_1 error, defined as $E(H) = \sum_{i=1}^{M} |g(i) - \hat{g}_H(i)|$, where each $\hat{g}_H(i)$ is the median of the g-values in the bucket containing i;
- the χ^2 error, defined as $E(H) = \sum_{i=1}^{M} \big(g(i) - \hat{g}_H(i)\big)^2 / \hat{g}_H(i)$, where $\hat{g}_H(i) = \sqrt{|S_j|^{-1} \sum_{l \in S_j} g^2(l)}$ and S_j is the bucket containing i;
- workload-weighted versions of the L_2 and L_1 error metrics; and
- the KL metric of Giannella and Sayrafi [121], defined as $E(H) = \sum_{j=1}^{B} |S_j| E_j$, where $|S_j|$ is the length of bucket S_j and

$$E_j = \log |S_j| - \sum_{i \in S_j} \frac{g(i)}{g_j^+} \log \frac{g(i)}{g_j^+},$$

  with g_j^+, as before, the sum of the g-values in bucket S_j.

The approach can also handle other estimation schemes, such as g-value summaries comprising the geometric mean or the median of the values in a bucket, or the spline-based schemes as described in Section 3.3.2. The precise formulas for time and space complexity may change, however, depending on the details of the error metrics and estimation schemes; see, for example,

[139, Sec. 4].

Jagadish et al. [185], extend this approach to the problem of globally optimizing a set of 1D histograms. Specifically, the problem is to allocate a total of B buckets among the histograms in order to minimize a global error metric. This metric is a weighted sum of the L_2 errors for the histograms in the set, where the weight of a histogram corresponds to the probability that an incoming query will probe the attribute associated with the histogram; these probabilities are estimated from a query workload. The authors provide both a DP and a greedy algorithm.

Improved DP Algorithms

Even with the improvement of Guha [134], the performance of the basic algorithm is likely to be unacceptable in practice. To address this issue, a number of algorithms have been developed to solve a relaxed version of the optimality problem. Such algorithms produce a histogram whose error is at most $(1+\varepsilon)$ times the minimum possible error—where ε is specified by the user—and are sometimes called *ε-approximate histograms*. The best results in this area are largely contained in recent work by Guha et al. [139]. For example, under the L_2 error metric, the authors provide a one-pass algorithm that produces a near-optimal histogram while having a near-linear time complexity of $O(M+Q\tau)$ and a space complexity of $O(B\tau+Q)$, where $\tau = \min(B\varepsilon^{-1}\log M, M)$ and $Q = B(B\varepsilon^{-1}\log\tau + \log M)\log\tau$. For typical values of B and ε, it holds that $\tau \ll M$ and $Q \ll B\tau\log M$. The algorithm is rather intricate, and the reader is referred to the original paper for details; the key idea is to follow the DP approach outlined above, but, for each $k \in [2,B]$, to efficiently and dynamically approximate the quantity $E^*_{k-1}[1,i)$ in Bellman's equation by a piecewise-constant function of i.

Guha et al. [139] show that their techniques extend to any error metric E such that

- the error $E[a,b]$ of a bucket with end points a and b only depends on a, b, and the g-values in the bucket;
- the overall histogram error is the sum of the bucket errors;
- $O(1)$ information can be maintained for each i value such that $E[a,b]$ can be computed efficiently, e.g., the quantities $R_i^{(1)}$ and

Case	γ_j	Bucket error
$c \leq \min \leq \max$	$\frac{2*\max*\min}{\max+\min}$	$\frac{\max-\min}{\max+\min}$
$\min \leq \max \leq -c$	$\frac{2*\max*\min}{\max+\min}$	$\frac{\min-\max}{\max+\min}$
$-c < \min < c \leq \max$	$\frac{\max(\min+c)}{\max+c}$	$\frac{\max-\min}{\max+c}$
$\min \leq -c \leq \max \leq c$	$\frac{\max(c-\max)}{c-\min}$	$\frac{\max-\min}{c-\min}$
$-c \leq \min \leq \max \leq c$	$\frac{\max+\min}{2}$	$\frac{\max-\min}{2c}$
$\min \leq -c < c \leq \max$	0	1

Table 3.1 Optimal g-value summary under the L_∞ relative-error metric

$R_i^{(2)}$ in the basic DP algorithm; and

- the error metric is "interval monotone" in that $E[a,b] \leq E[a-1,b]$ and $E[a,b] \leq E[a,b+1]$.

This class of metrics includes those described in the previous section. Note, however, that, e.g., the authors' scheme for the L_1 metric requires a preprocessing step having $O(M \log M)$ time and space complexity, which may be practically infeasible. Other metrics amenable to the methods in [139] include certain relative-error metrics [142] such as the L_2 relative error metric $\sum_{i=1}^{M} \big(g(i) - \hat{g}_H(i)\big)^2 / w_i^2$. Here $w_i = \max\big(g(i), c\big)$ and $\hat{g}_H(i) = \gamma_j$ with S_j being the bucket that contains i and γ_j being a constant chosen to minimize the contributions to the error from the g-values in S_j. Specifically, $\gamma_j = C_j - B_j^2 - A_j$, where $A_j = \sum_{i \in S_j} (1/w_j)$, $B_j = \sum_{i \in S_j} \big(g(i)/w(i)\big)$, and $C_j = \sum_{i \in S_j} \big(g^2(i)/w(i)\big)$. Note that c is a sanity constant that prevents undue influence by very small g-values. The methods in [139] also apply to the L_1 relative-error metric but, as with the simpler L_1 absolute-error metric mentioned in the previous subsection, the space complexity is unacceptably high.

Maximum-Error Metric

In general, the L_∞ metric—that is, the maximum error metric—is more tractable than other metrics such as L_1 and L_2. Guha et al. [142] provide an algorithm for computing a near-optimal histogram under the L_∞ relative-

error metric. This metric is defined as $\max_{1\le i\le M} |g(i) - \hat{g}_H(i)|/|w_i|$, where, as before, $w_i = \max\big(g(i), c\big)$ and $\hat{g}_H(i) = \gamma_j$ with S_j being the bucket that contains i. As with the L_2 relative error metric, γ_j is chosen to minimize the error contribution from bucket S_j. In this case, the formula for γ_j is encapsulated in Table 3.1, which also gives the resulting error contribution from the bucket. (The overall histogram error is the maximum of the bucket errors.) In the table, "min" and "max" refer to the minimum and maximum g-values in S_j, and c is the sanity constant; see [141] for a derivation. The time complexity of the algorithm is $O(M)$ and the space complexity is $O\Big(B^2\varepsilon^{-1}\log M\big(\log\log M + \log(B/\varepsilon)\big)^3\Big)$.

For the L_∞ absolute-error metric, defined as $\max_{1\le i\le M} |g(i) - \hat{g}_H(i)|$, Buragohain et al. [38] provide several simple approximate histogram algorithms that have excellent space and time complexities relative to other approaches; these algorithms are *not* based on dynamic programming. Their one-pass Min-Merge algorithm has space and time complexities of $O(B)$ and $O(M\log B)$, and constructs a histogram using at most $2B$ buckets that has L_∞ error less than or equal to the error for the optimal B-bucket histogram. The buckets of the histogram store the minimum and maximum g-values, and approximate each of the g-values in the bucket by the single value $(\max+\min)/2$, which gives a bucket error of $(\max-\min)/2$. The algorithm is quite simple, and works by dynamically maintaining a set of buckets that form a histogram of the values seen so far. Each arriving value is assigned its own bucket and appended to the right end of the histogram. Whenever the addition of a new bucket causes the budget of $2B$ to be exceeded, two adjacent buckets are merged; this bucket pair is chosen so as to minimize the increase in error.

The Min-Increment algorithm in [38] requires that the g-values lie in a subset of the integers that contains $R > 1$ elements; this condition is, of course, satisfied when $g(i)$ corresponds to the frequency of data value i. The algorithm produces a histogram having at most B buckets and an L_∞ error that lies within a $(1+\varepsilon)$ factor of the minimum possible error. The space and time complexities are $O(\varepsilon^{-1}B\log R)$ and $O(\varepsilon^{-1}M\log R)$. The Min-Increment algorithm proceeds by solving the "dual" problem of finding the histogram with the minimal number of buckets whose error does not exceed a given bound. This dual problem is solved for a range of exponentially increasing

error values of the form $e_i = (1+\varepsilon)^i$ for $i = 1,2,\ldots,\log R/\varepsilon$, which lie in the range $[1,R]$ of possible error values. The algorithm for solving the dual problem, called Greedy-Insert, is very simple. A list of buckets is maintained, with the rightmost bucket being "open" and the rest being "closed." The algorithm starts with a single, open bucket. Arriving points are added to the open bucket as long as the error does not exceed the specified error bound e; if the bound is about to be exceeded, the bucket becomes closed and the arriving g-value is placed in a new, open bucket. Observe that, because successive e_i values are separated by a factor of $(1+\varepsilon)$, there exists an i^* such that $e_{i^*-1} \leq e_{\text{opt}} \leq e_{i^*} \leq (1+\varepsilon)e_{\text{opt}}$, where e_{opt} is the L_∞ error of the optimal B-bucket histogram. The minimum bucket histogram with bound e_{i^*} is the desired approximate histogram, and the challenge is to identify i^*, since e_{opt} is unknown. The Min-Increment algorithm simply maintains a set of histograms as in the Greedy-Insert algorithm, one for each e_i value. Whenever the bucket count for such a histogram exceeds B, it is discarded; by the minimal-bucket property of Greedy-Insert, the error for such a histogram is less than or equal to e_{i^*-1}. After all of the g-values have been processed, the surviving histogram with the smallest e_i value is the desired approximate histogram. Buragohain et al. [38] extend the Min-Merge and Min-Increment algorithms to handle spline histograms as in Section 3.3.2.

Range-Query Error Metric

All of the algorithms so far use an error metric that is oriented toward minimizing "pointwise" errors in reconstructing the value of each $g(i)$. If a histogram is primarily to be used in approximately answering range queries, it makes sense to use an error metric that reflects this fact. Guha et al. [140] provide an algorithm that attempts to minimize error with respect to a workload $\mathscr{W}$ of *hierarchical range queries*. A range query Q_{ij} with $i \leq j$ asks for the range sum $\alpha_{ij} = g(i) + g(i+1) + \cdots + g(j)$. A set $\mathscr{Q}$ of hierarchical range queries satisfies the property that, for each pair $Q_{ij}, Q_{kl} \in \mathscr{Q}$, either the ranges $[i,j]$ and $[j,k]$ are disjoint or one contains the other. A workload $\mathscr{W}$ comprises a set $\mathscr{Q}$ of hierarchical range queries together with a probability distribution $\mathscr{P} = \{ p_{ij} \}$ over the queries in $\mathscr{Q}$ such that p_{ij} denotes the probability that an incoming query Q will correspond to $Q_{ij} \in \mathscr{Q}$, i.e., will be a range query over $[i,j]$. Let $\hat{\alpha}_{ij}^H$ denote the estimate of α_{ij} for a given histogram

H, and denote by $e_{ij}^H = (\alpha_{ij} - \hat{\alpha}_{ij}^H)^2$ the corresponding squared error. Next, let $E_H = \sum_{(i,j)} e_{ij}^H p_{ij}$ be the expected (squared) error with respect to the workload when using histogram H. Finally, let H^* be the histogram that minimizes the error and denote by $E^* = E_{H^*}$ the minimal error. Guha et al. develop an efficient DP algorithm that, for any $\varepsilon \in (0,1)$, produces a histogram having $O(B/\varepsilon)$ buckets and whose expected error with respect to $\mathscr{W}$ is the minimal B-bucket error E^*. The time complexity is $O(M + B^2\varepsilon^{-5}M^{\varepsilon}|\mathscr{Q}|)$ and the space complexity is $O(B\varepsilon^{-3}M^{2\varepsilon/3})$. The key idea is to restrict computation to a "sparse" system of intervals that covers the ranges in $\mathscr{Q}$, increasing the number of histogram buckets in the process but reducing the overall DP complexity. This method can be potentially extended to other error metrics and estimation schemes.

The foregoing bucketing schemes assume the datacube access model. To handle the relational access model (with SUM aggregate) in a one-pass manner, one needs algorithms that can incrementally update a histogram upon scanning a data item of the form (i,v) for some $i \in [1,M]$, which has the interpretation "increment the current value of $g(i)$ by v." As usual, a key example has $v \equiv 1$, so that g is interpreted as the frequency of data value i. Algorithms for handling this situation are a special case of algorithms for maintaining histograms over streaming data, and are discussed in Section 3.7.1.

q-Error Metric

Recently, Kanne and Moerkotte [198] have forcefully argued for the use of the *q-error* to evaluate the quality of a histogram, especially in the context of query optimization. If nonnegative g-values $g(1), g(2), \ldots, g(M)$ are approximated by histogram estimates $\hat{g}_H(1), \hat{g}_H(2), \ldots, \hat{g}_H(M)$ then the q-error of the approximation is defined as $\max_{1 \le i \le M} \max\big(g(i)/\hat{g}_H(i), \hat{g}_H(i)/g(i)\big)$. A key motivation for the use of q-error rests on recent results of Moerkotte et al. [231] that directly relate q-errors of cardinality estimates to query-plan costs. For example, for a query consisting of multiple sort-merge joins or Grace hash joins, it can be shown that $C(\hat{P}) \le q^4 C(P)$, where $C(\hat{P})$ and $C(P)$ are the query costs for optimal plans based on estimated and exact join cardinalities and q represents a worst-case q-error in the intermediate join-cardinality estimates. In [198], a dynamic-programming-style algorithm is given for exact computation of a "q-optimal" histogram. Such a histogram

has minimal size, subject to an upper bound on the q-error. The class of histograms considered allows heterogeneous bucket types, as described in Section 3.3.4. The exact algorithm is costly, and so a greedy heuristic is also provided. The experiments reported in [198] indicate that, for a wide range of datasets, it is possible to obtain a heterogeneous histogram of reasonable size for which the q-error at most 2. Note that special care must be taken to deal with the case $g(i) = 0$, since the q-error for such a value can be infinite.

3.4.3 Hierarchical Bucketing Schemes

The bucketing schemes discussed so far simply partition the range $[1, M]$ into disjoint segments. More elaborate schemes attempt to better capture the structure of the data by defining buckets in a hierarchical manner. The resulting data structures lie on the boundary between histograms and synopses such as wavelets. We describe two such proposals below, n-level trees and lattice histograms. In the context of multi-dimensional histograms, Section 3.5.1 below describes the hierarchical STHoles bucketing scheme of Bruno et al. [30].

n-Level Trees

The *n-level tree (nLT)* introduced in [35] is designed to answer hierarchical range queries. In an exact nLT, sums over the data are stored as an n-level binary tree. The root node stores the sum of $g(1)$–$g(M)$. The left child node stores (roughly) the sum of $g(1)$–$g(M/2)$, and the right child stores the sum of $g(M/2)$–$g(M)$. This scheme continues recursively, with the left child of a node storing the sum of g-values over the first half of the node's range and the right child storing the sum over the right half. The storage requirement of the nLT is then reduced by using the same tricks and approximations as in the 4LT method described in Section 3.3.3. Namely, only the sums for left-child nodes are stored, because the right-child sums can be reconstructed as a difference between the left-child sum and the parent-node sum. Moreover, the number of bits used to represent the sums decreases as the node level increases. Specifically, the sum for the root node (level 0) is represented using 32 bits, the sum of the left-child node (level 1) is represented using k ($< n$) bits. From then on, the number of bits in the representation decreases by 1 as the level increases by 1. The nLT is used to answer a query in a manner similar to the hierarchical-range-query histogram of Guha et al. [140] described in

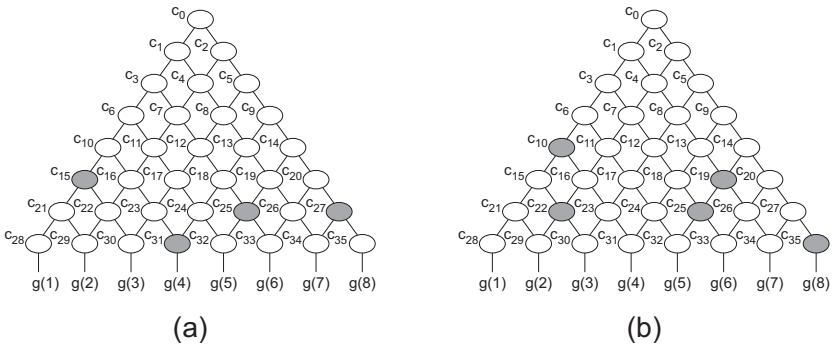


Fig. 3.8 Two lattice histograms on eight data values

Section 3.4.2. The idea is to attempt to cover the query range by one or more nodes that are as high up in the tree as possible. If, after such an attempted covering, a portion of the query range is not completely covered, then the sum over this remaining portion is estimated using the continuous-value assumption. Experiments in [35] show that the accuracy of this approach can be superior to maxdiff, v-optimal, and wavelet approaches.

The price for this improved accuracy is that, under the datacube access model, the nLT requires $O(M \log M)$ time to construct and $O(\log M)$ time to answer a query. The nLT is relatively easy to update, requiring $O(\log M)$ time complexity to handle a change in $g(i)$ for some specified value of i.

Lattice Histograms

We conclude our discussion of 1D-histograms by describing recent work by Karras and Mamoulis [199, 201] on a synopsis called a *lattice histogram* (LH), which is actually a hybrid of a histogram and a wavelet synopsis. The motivation for this work is to try and exploit both the traditional histogram's ability to flexibly and accurately capture locally smooth data, yielding buckets containing similar data values, while also exploiting any underlying hierarchies of the data in order to capture non-local interrelations, as is done effectively by hierarchical synopsis structures such as wavelets. Following [201], Figure 3.8 illustrates two LHs, each summarizing $M = 8$ g-values. The space occupied by a LH is proportional to the number of occupied nodes, each of which contains a numerical value; these correspond to the gray nodes in

the figure. Nodes cannot be occupied in an arbitrary fashion: the constraint is that, for any pair of occupied nodes, either their respective sets of leaf descendants must be disjoint, or one set must completely contain the other. A data value is reconstructed as the value of the lowest occupied ancestor node. For example, in the LH of Figure 3.8(b), the reconstructed values are $\hat{g}(1) = c_{10}$, $\hat{g}(2) = c_{22}$, $\hat{g}(5) = c_{25}$, and $\hat{g}(8) = c_{35}$. Note that the constraint on the occupation of nodes ensures that there is always a unique lowest ancestor. The LH of Figure 3.8(a) shows how the LH framework can capture traditional histograms as a special case: here the data has been partitioned into the buckets $S_1 = \{1,2,3\}$, $S_2 = \{4\}$, $S_3 = \{5,6\}$, and $S_4 = \{7,8\}$, and the values in the occupied nodes correspond to bucket averages.

For a given space budget B, Karras and Mamoulis [201] provide a dynamic programming algorithm to construct a LH that approximately minimizes a general error metric. The g-values are discretized into multiples of a constant δ, but otherwise the algorithm is exact. This discretization means that the error of the constructed LH exceeds the minimum possible error by no more than $\delta/2$. The time and (working) space complexities are $O(\delta^{-1}M^3B^2)$ and $O(\delta^{-1}MB)$, respectively. A DP algorithm specialized to the L_∞ maximum-error metric reduces these requirements to $O(\delta^{-1}M^3)$ and $O(\delta^{-1}M)$, but clearly the time and space costs are impractical. To address this problem, the authors recommend a divide-and-conquer procedure in which the g-values are first partitioned using a "primary" synopsis such as an ordinary histogram, and then a set of LHs are constructed, one per bucket. In follow-on work that focuses on the class of weighted L_∞ metrics (which include maximum-relative-error metrics), Karras [199] provides an algorithm for constructing a LH that exactly minimizes the error metric of interest, along with a much less expensive greedy algorithm that is shown empirically to produce LH's that are close to optimal. The idea is to develop exact and greedy algorithms that minimize the LH size subject to a given error bound, and then solve the original space-bounded problem by combining the error-bounded algorithms with binary search. The time and space complexities for the exact algorithm are $O(M^4)$ and $O(M^3)$, and the corresponding complexities for the greedy algorithm are $O(M+B^2)$ and $O(M)$. The space complexity even of the greedy algorithm may still be impractical but again the divide-and-conquer procedure can potentially be applied. An interesting line of research would try and further improve efficiency by incorporating

approximate DP ideas along the lines of Guha et al. [139].

3.5 Multi-Dimensional Histograms

The foregoing discussion has focused primarily on histograms for a single attribute. In practice, however, the paramount need is to capture multidimensional distributions, i.e., the statistical relationships between different attributes. In query optimization, for example, failure to capture dependencies is the key cause of poor selectivity estimates and consequent selection of bad query plans.

Given a bucketing scheme, most of the estimation schemes described for 1D-histograms generalize in a straightforward manner. The major focus of our discussion is therefore on bucketing schemes. As discussed below, multi-dimensional histograms present much greater technical challenges than 1D-histograms, because there are many more degrees of freedom for choosing bucket boundaries, and the potential for an exponential explosion in time and memory costs is great. Moreover, data tends to be sparser in higher dimensions, making efficient proximity-based compression difficult. The problem of providing accurate multi-dimensional histograms at reasonable cost is an ongoing research area, and the following results are provisional at best.

As before, we focus on the problem of estimating quantities such as $\alpha = \sum_{i \in Q} g(i)$, but now $i = (i_1, i_2, \ldots, i_d)$ for some dimension $d > 1$, and the query region Q—as well as each histogram bucket S_j—is a subset of $\mathscr{U} = [1, M_1] \times [1, M_2] \times \cdots \times [1, M_d]$; typically, both Q and S_j are hyperrectangles. Thus the total number of g-values in the dataset is $M = M_1 M_2 \cdots M_d$; e.g., $M = m^d$ if $M_1 = M_2 = \cdots = M_d = m$. (Typically, however, many of these g-values are equal to 0.) Unless specified otherwise, we assume the datacube access model throughout. Indeed, none of the techniques below can directly handle the relational access model, except for equi-width histograms, which, as discussed in Section 3.5.1 below, are impractical. Thus techniques such as sampling or sketching might need to be used; c.f. Section 3.2.3. Muralikrishna and DeWitt [235], for example, found that sampling worked well with their multi-dimensional histogramming method—see Section 3.5.1 below—for up to three dimensions.

3.5.1 Bucketing Schemes

We first discuss why the bucketing problem is extremely challenging, and then review a range of bucketing strategies that have been proposed in the literature. Even the most successful of these schemes do not perform well when the dimensionality of the data is high. For high-dimensional data, the most promising practical approach maintains a collection of low-dimensional histograms, of the type discussed in this section, and combines estimates from the different histograms into an overall approximate query answer. Section 3.5.2 describes this multi-histogram approach. In connection with this strategy, some potentially useful low-dimensional histograms include the heuristic algorithm of Wang and Sevcik [287] and the approximate algorithms for optimal 2D-histograms in Muthukrishnan et al. [238]; they each require a minimal number of passes over the data to construct. Both of these algorithms, and others, are discussed in the sequel.

Challenges

To appreciate the fundamental difficulties of the bucketing problem for a d-dimensional histogram, first observe that the number of buckets in a d-dimensional equi-width histogram grows exponentially in d; for many real-world datasets, the vast majority of the cells will be empty, and a waste of storage. Moreover, to achieve good accuracy in a region of the data domain where the data values are highly nonuniform, the cell widths for this region, and hence for the entire histogram, will need to be small, further amplifying the memory-consumption problem. Conversely, for a given (small) memory bound, the accuracy of an equi-width histogram will typically be unacceptable, unless the data is uniform.

Combatting the above accuracy problem typically incurs large space and/or time costs for constructing the histogram. Indeed, under the datacube access model, any construction algorithm that requires a complete pass over the input data—an $O(M)$ time requirement in our previous notation—will be restricted to small values of d, because M grows exponentially in d. Many of the bucketing schemes proposed in the literature, though yielding much better accuracy than the equi-width approach, require multiple passes over the data, which severely limits their practicality. The VI histogram technique of Wang

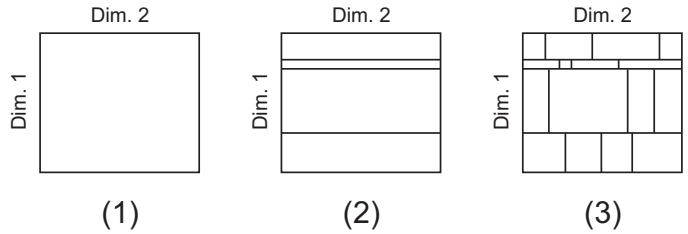


Fig. 3.9 Muralikrishna and DeWitt bucketing scheme

and Sevcik [287], described below, partially addresses this issue.

Theoretical results confirm the difficulty of the d-dimensional bucketing problem. Recall from Section 3.4.2 that an exact v-optimal 1D-histogram can be constructed in polynomial time, specifically, $O(M^2B)$ under the datacube access model. In contrast, Muthukrishnan et al. [238] show that, even when $d = 2$, this problem is NP-hard, as are a broad class of related histogram-construction problems. Even computing an ε-approximate equi-depth histogram with a specified number B of buckets in each dimension is challenging [208]. As a consequence, most of the bucketing schemes studied so far have been heuristic.

A related issue is that the number of buckets intersected by a query is likely to increase as the number of dimensions increases. This potentially slows down the process of efficiently answering a query. Carefully designed data structures are needed to ameliorate this problem.

Multi-Pass Schemes

We first describe bucketing schemes that require multiple passes over the input data. Besides being of historical interest, recent work has shown that these schemes can be incorporated into more efficient construction algorithms that require only a single pass.

The first of the multi-pass bucketing algorithms was the method of Muralikrishna and DeWitt [235]. This top-down algorithm begins with a single bucket that contains all of the d-dimensional data, and proceeds by choosing an arbitrary dimension and splitting the bucket along this dimension using an equi-depth scheme (after projecting the data points onto the splitting dimension). Each resulting bucket is then recursively split along the remaining

dimensions. The number of splits is predetermined—typically the same number in each dimension—so that the total number of buckets does not exceed the space budget. See Figure 3.9 for the case of two-dimensional data; in the figure, there are three splits (i.e., four disjoint intervals) in each dimension and, arbitrarily, dimension 1 is split before dimension 2. To answer queries efficiently, the histogram data is stored in an *R*-tree-like structure that reflects the recursive nature of the bucketing scheme. More specifically, the height of the tree corresponds to the number of dimensions, with descending levels corresponding to the successive splitting dimensions. Each internal node corresponds to an interval, demarcated by adjacent split points, along the dimension corresponding to the node's level. A leaf node contains the bounding-box information for a histogram bucket, such that the intervals defining the bucket boundaries are contained in the intervals of the bucket's ancestor nodes. This organization ensures that only buckets in the neighborhood of the query region will be visited. In the setting of 2D-histograms, Pham and Sevcik [251] performed a set of empirical experiments to systematically explore the problem of how to order the dimensions for splitting, and how many splits to use in each dimension. Given an overall budget of B buckets, they recommend partitioning each dimension into $\sqrt{B}$ buckets, validating Muralikrishna and DeWitt's original recommendation of $B^{1/d}$, at least when $d = 2$. To determine the splitting dimension at each step, they define the density of a bucket S_j as $g_j^+ / |S_j|$, and recommend choosing the currently un-split dimension that has the largest variance of density values over the buckets. (They did not, however, systematically consider alternative heuristics for choosing the splitting order, of which there are many; see, e.g., the discussions of the MHIST and HiRed algorithms immediately below.)

Poosala and Ioannidis [257] point out that, in the foregoing method, the splits in each dimension can be chosen according to any one-dimensional bucketing scheme such as maxdiff, v-optimal, and so forth. They refer to their generalization as the PHASED bucketing method, which thus includes the method of Muralikrishna and DeWitt as a special case. They also propose a more significant generalization, which they call MHIST. This top-down algorithm again starts with a single bucket for the entire dataset, and splits an existing bucket into two buckets at each step. The bucket to be split and the dimension of this bucket along which to split are chosen greedily, according to the one-dimensional bucketing scheme being used. E.g., for the v-optimal

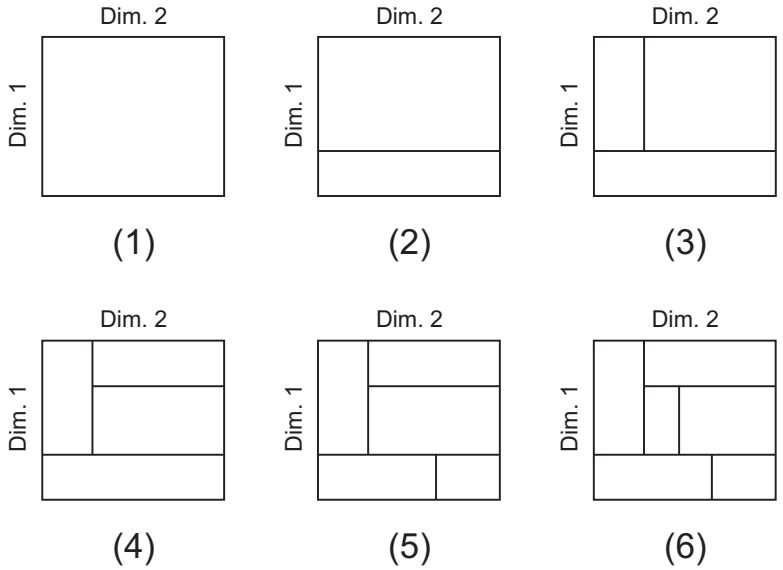


Fig. 3.10 MHIST bucketing scheme

scheme, the bucket and dimension having the largest L_2 error value are chosen for splitting. The algorithm stops when the number of buckets reaches the maximum allowed value. The MHIST approach permits a more flexible choice of bucket boundaries than the PHASED method, leading to better empirical accuracy; see Figure 3.10 for an example of a possible MHIST splitting sequence.

Baltrunas et al. [16] give a variant of the MHIST algorithm, called HiRed, where a bucket is split in half along each of its dimensions; for each split of a bucket S into buckets S_1 and S_2 along a specified dimension, a variability measure $|g_1^+ - \bar{g}| + |g_2^+ - \bar{g}|$ is computed, where g_i^+ is the sum of the g-values in bucket S_i and $\bar{g} = (g_1^+ + g_2^+)/2$. If the variability measure exceeds a user-specified threshold, then the split becomes permanent; otherwise, the split is removed. The multi-dimensional version of Wang and Sevcik's HMF histogram—see Section 3.3.4—also proceeds in a manner similar to MHIST, iteratively and greedily splitting a selected bucket into two buckets, along a selected dimension, with the additional complication that (possibly different) estimation schemes must be chosen for each of the newly created buckets. As with the one-dimensional algorithm, the various splitting possibilities are

ranked using a metric based on relative entropy and inspired by the minimum description length principle. Note that the PHASED, MHIST, and HiRed approaches, though more general than the original scheme of Muralikrishna and DeWitt [235], still explore only a subset of all possible (disjoint) bucket configurations, since some valid two-dimensional configurations cannot be achieved by hierarchical schemes.

Gunopulos at al. [143] obtain additional flexibility relative to PHASED, MHIST, and Hired by allowing buckets to overlap. Their algorithm, called GENHIST, works roughly as follows. Define the density of a bucket S as $f_S = \sum_{i \in S} g(i)/g^+$, where g^+ denotes, as usual, the sum of all g-values in the dataset. A GENHIST histogram consists of a set $\mathscr{S}$ of overlapping buckets, and the g-values are estimated as $\hat{g}(i) = g^+ \sum_{S \in \mathscr{S}_i} f_S$, where $\mathscr{S}_i$ is the set of buckets in $\mathscr{S}$ that contain i. The algorithm computes buckets by first computing a "fine-grained" equi-width partitioning of the data domain and identifying a set of "dense" buckets; a bucket is dense if its density is higher than the "average local density," i.e., the average of the densities in its neighboring buckets. For each dense bucket, a subset of the data points in the bucket is "assigned" to the bucket and removed from the dataset, and the bucket is then added to the histogram. That is, MHIST records the bucket boundaries and bucket density, where the density is defined as the number of assigned points, divided by g^+. Assigned points are chosen randomly and uniformly from the points within a bucket, and the number of assigned points is chosen such that the remaining points in the bucket region have a density equal to the local average density. Thus the reduced dataset (consisting of the remaining data points) has a more uniform distribution and can be approximated using a coarser partitioning; this process is similar to the assignment of a large g-value to its own bucket in a high-biased histogram (c.f. the discussion on singleton bucket histograms in Section 3.4.1). The process is then iterated on the remaining data points, using a coarser equi-width partition. The iterations continue until either all points have been removed or the partitioning coarsens to a single bucket.

Figure 3.11 illustrates the process for $d = 2$. In iteration 1—see Figure 3.11(a)—the domain is partitioned into 16 equi-width buckets via a 4×4 partitioning. The upper left bucket is identified as dense and added to the histogram; the density a assigned to the bucket represents the excess over the local average density. In iteration 2—see Figure 3.11(b)—the domain is par-

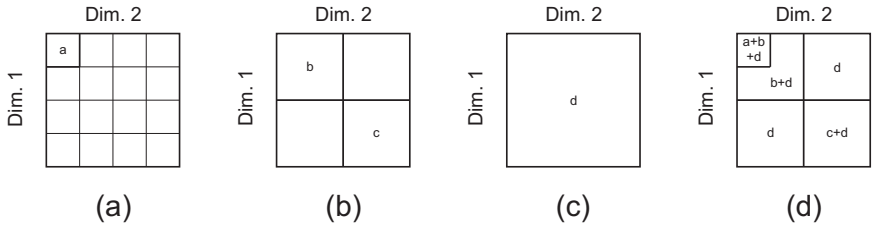


Fig. 3.11 GENHIST bucketing scheme

titioned into 4 equi-width buckets, and the upper left and lower right buckets are each identified as dense and added to the histogram, with respective densities b and c. In iteration 3—see Figure 3.11(c)—the partitioning has coarsened to a single bucket, which is added to the histogram. In this case the density d of the bucket is computed as the sum of the g-values remaining in the dataset, divided by the sum g^+ of the entire original set of g-values. The final density approximation for a given region is computed as the sum of the densities of the buckets that contain the region, as illustrated in Figure 3.11(d). Note that the bucket having density b overlaps (in fact completely contains) the bucket having density a; in general, arbitrary overlaps are possible.

Although the GENHIST method provides very accurate query-result estimates, the costs for constructing and using the histogram are quite high, as are the space requirements for computing and storing the histogram. E.g., the complexity of the initial density computations is exponential in the dimension of the data. Experiments by Baltrunas et al. [16] indicate that, given a space budget, the simple HiRed histogram can dominate GENHIST in speed and accuracy, at least on some datasets.

A related bucketing scheme is the STHoles method of Bruno et al. [30], which we discuss for the case in which $g(i)$ corresponds to the frequency of a value i. In this scheme, high-density regions are completely, not partially, removed, in a recursive fashion. Thus buckets can have "holes," which are themselves buckets that can have holes, and so forth; see Figure 3.12. Unlike the other bucketing schemes discussed so far, the method for creating buckets ("drilling holes") and assigning summary values to buckets rests not upon a scan of the data, but rather on observing query feedback, i.e., the actual numbers of tuples that satisfy selection queries issued by users. Srivastava et al. [275] improve upon the original STHoles scheme by updating the bucket

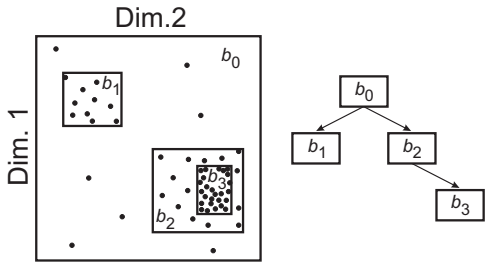


Fig. 3.12 STHoles bucketing scheme

summary values using a maximum entropy approach that ensures consistency of the bucket counts with respect to the feedback observations. Recently, Kaushik et al. [206] have extended the maximum entropy approach to handle feedback information that includes information about distinct-value counts. He et al. [164] predict the future query workload, based on past queries, and create a "proactive" STHoles histogram from this workload, which is then merged with the ordinary "reactive" STHoles histogram when the future time arrives. Luo et al. [220] provide methods for scheduling queries so as to maximize the rate at which an STHoles histogram converges to an accurate representation of the data, and also introduce an interpolation method that leads to improved accuracy for regions in a bucket that are "close" to holes. The various STHoles construction techniques are somewhat tangential to our setting, in which queries are answered approximately, and not exactly. However, one could envision a scenario where the system switches back and forth between a training regime—in which queries are processed exactly and the feedback is used to build or refine histograms—and an approximate-processing regime. The system would switch to the training regime whenever the error in the approximate answers exceeds a specified threshold.

In general, there is a trade-off between the flexibility of a bucketing scheme and the amount of data that needs to be stored in order to specify the buckets. The more information that needs to be stored per bucket, the fewer buckets that can be maintained. Fuchs et al. [100] give a scheme for compressing the bucket information in STHoles, and show that use of this technique can reduce the relative error by up to 40%. The compression technique is to quantize the each coordinate of a bucket relative to the corresponding

coordinate of the smallest enclosing bucket.

Single-Pass Schemes

As discussed previously, the multi-dimensional equi-width histogram can be constructed in a single pass (or at most two passes if the range of data values is unknown), but this histogram performs extremely poorly as the dimension d increases. Wang and Sevcik [287] propose the VI (values and intervals) histogram, a high-biased histogram that can be constructed in a single pass through the data.[7] The algorithm has the important advantage that it need only scan the positive g-values, leading to significant reductions in time complexity when only positive g-values in a dataset are explicitly stored. The idea is to place any g-value larger than $g^+/20$ into its own bucket; the threshold is chosen to limit the possible number of such buckets (to 20) and is based on empirical studies. The remaining values are stored in a relatively coarse memory-resident equi-width histogram (with about 20 buckets in each dimension), which is then compressed into a PHASED histogram. The VI histogram comprises the PHASED histogram together with the singleton buckets. The experiments in [287] indicate that this technique has the potential to strike a good balance between histogram accuracy and practicality.

In related work, Pham and Sevcik [251] study the problem of determining the number of singleton buckets to maintain in a more systematic fashion. They focus on thresholds of the form $g^+/(\alpha B)$, where the parameter α determines the priority placed on singleton buckets. Poosala et al. [258] used this type of threshold (with $\alpha = 1$) for the original maxdiff histogram. Pham and Sevcik found empirically that the optimal choice of α is highly sensitive to the particular bucketing scheme used, with, e.g., a value of $\alpha \approx 3$ being roughly optimal for PHASED histograms with equi-depth partitioning.

Optimal Histograms

Results on optimal and near-optimal multidimensional histograms are quite sparse, and indeed are found primarily in a paper by Muthukrishnan et al. [238] that contains the NP-hardness result cited earlier. Recall that the

[7] Actually, the algorithm requires that g^+ be known (or well approximated) a priori, and so might require two passes rather than one, as with the multi-dimensional equi-width histogram.

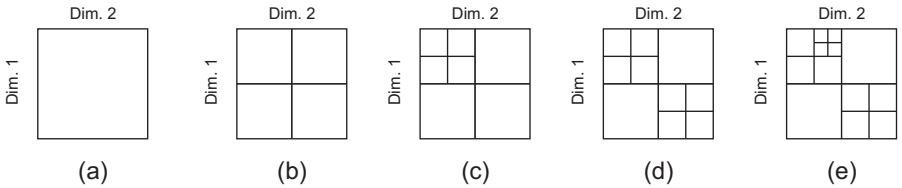


Fig. 3.13 Quad-tree bucketing scheme

thrust of this result is that there appear to be no efficient algorithms to compute optimal histograms even in the case of $d = 2$. This raises the question of whether there exist efficient algorithms for computing histograms that are near-optimal, i.e., histograms with a specified space allocation of B buckets and having an approximation error within a factor of $(1+\varepsilon)$ of the minimum possible error for a B-bucket histogram. This question is still unresolved, but Muthukrishnan et al. [238] provide several algorithms for approximately solving the dual problem in two dimensions: for a specified error bound δ, find a 2D-histogram achieving this error bound and having at most kB^* buckets for some specified k, where B^* is the minimal number of buckets required to satisfy the error bound. For example, in the case of hierarchical (i.e., recursive) bucketing, as in MHIST, the authors give an algorithm that runs in $O(M^{1+\varepsilon})$ time with $k = O(1/\varepsilon^2)$; here M is the total number of g-values in the two-dimensional dataset, as per our previous notation. The algorithm uses dynamic programming, and seems to have a prohibitive space complexity. The paper [238] contains similar results for $p \times p$ grid-like bucketing schemes and for arbitrary bucketing schemes.

Buccafuri et al. [34] obtain optimal and near-optimal algorithms that avoid the NP hardness issue by focusing on two-dimensional data and strongly restricting the allowable partitioning scheme. Specifically, the partitioning scheme starts with a single bucket. At each step, an existing bucket is split into 4 buckets by placing a bucket boundary at the midpoint of each dimension, in a manner similar to the scheme of Baltrunas et al. [16] described previously; see Figure 3.13. Given a space budget B, a dynamic programming algorithm can compute the v-optimal histogram—i.e., minimizing the L_2 error over all possible quad-tree partitionings—in time $O(BM \log M)$. This complexity is still impractical, and so the authors provide a greedy algorithm

having $O(B \log B)$ time complexity; the idea is to greedily choose a bucket to split, specifically, the bucket having the maximum L_2 error. The greedy algorithm can further be improved using the two-dimensional generalization of the techniques described in Section 3.3.3, namely, storing a subset of bucket sums explicitly and using these to derive sums for non-stored buckets, and using an analog of the 4LT to better estimate values within each bucket.

We conclude this section by briefly discussing connections to multi-dimensional indexes. We have seen that, for the PHASED-type histogram of Muralikrishna and DeWitt, the histogram is stored in a data structure that looks similar to an R-tree, except that the leaf nodes, instead of holding pointers to data items, contain count values. This close relationship between histograms and indices has been pointed out by several authors; see, for example, the survey by Ioannidis [180]. Indeed, Read et al. [263] propose augmenting existing B^+-tree indices to hold summary statistics for use in aggregation queries, and Aoki [10] proposes analogous modifications of the Generalized Search Tree (GiST) in the context of selectivity estimation. The overall hope is to leverage the highly effective technology that has been developed for indexing multi-dimensional data.

3.5.2 Collections of Histograms

Even the best of the foregoing bucketing methods has difficulty dealing with data of very high dimension. One possibility for dealing with this problem is to approximate the full datacube via a collection of histograms, where each histogram is constructed over a small subset of the total set of dimensions. A graphical statistical model captures the global correlation structure of the data and is used to combine results from the different histograms into an overall approximation. We describe a couple of such methods below.

To simplify the exposition, we focus on point-count queries and assume that the g function coincides with the frequency function f. Thus our data is a multi-dimensional array of frequencies with entries of the form $f(v_1, v_2, \ldots, v_d)$, where $1 \leq v_k \leq M_k$ for $k \in [1,d]$. I.e., $f(v_1, v_2, \ldots, v_d)$ is the number of data points $x = (x_1, x_2, \ldots, x_d)$ with $x_i = v_i$ for $i \in [1,d]$. We denote by N the total number of points and by h the relative frequency function: $N = \sum_{v_1, v_2, \ldots, v_d} f(v_1, v_2, \ldots, v_d)$ and $h(v_1, v_2, \ldots, v_d) = f(v_1, v_2, \ldots, v_d)/N$. We denote marginal relative frequen-

cies over a collection of dimensions $A \subseteq \{1,2,\ldots,d\}$ using subscripts; e.g., $h_{2,4}(u,w) = \sum_{v_1,v_3,v_5,\ldots,v_d} h(v_1,u,v_3,w,v_5,\ldots,v_d)$. Conditional relative frequencies are defined analogously to conditional probabilities, e.g., $h_{2|3}(u \mid v) = h_{2,3}(u,v)/h_3(v)$. Thus if W is the collection of points whose third dimension has value v, then $h_{2|3}(u \mid v)$ is the fraction of points in W having second dimension equal to u. Two dimensions i and j are *independent* if $h_{i,j}(u,v) = h_i(u)h_j(v)$ for all $i \in [1,M_i]$ and $j \in [1,M_j]$. This notion corresponds to the usual notion of statistical independence if we select a point X at random from the dataset; the quantity $h_i(u)$ is then the probability that $X_i = u$, the quantity $h_{i,j}(u,v)$ is the probability that $X_i = u$ and $X_j = v$, and so forth. Dimensions i and k are *conditionally independent*, given dimension j, if $h_{i,k|j}(u,w \mid v) = h_{i|j}(u \mid v)h_{k|j}(w \mid v)$ for all u,v,w.

Decompositions Based on Graphical Models

Deshpande et al. [84] capture the correlation structure of the data dimensions by means of a "decomposable interaction model." Such a model can be represented graphically as a *Markov network*, i.e., an undirected graph in which the nodes represent dimensions and an edge represents a direct correlation between the dimensions that it connects. Two dimensions i and k that are separated by a dimension j are conditionally independent, given dimension j. The Markov network serves to define a collection of (possibly non-disjoint) groups of dimensions, which correspond to the *cliques* (maximal completely connected subgraphs) of the network. A joint histogram is maintained for each clique, and desired joint distributions are computed by combining marginal relative frequencies for the various histograms according to rules that are determined by the interaction model. In one simple scenario, for example, the cliques correspond to disjoint groups of correlated dimensions, and dimensions in different cliques are considered to be independent. Thus, if a dataset contains dimensions i, j, k, and l, and there are two cliques $[i,j]$ and $[k,l]$, then we estimate $h_{i,l}(t,w)$ as $\hat{h}_{i,l}(t,w) = \hat{h}_i^{i,j}(t)\hat{h}_l^{k,l}(w)$, where $\hat{h}^{i,j}$ is the approximate relative frequency function for dimensions i and j, based on the 2D-histogram over these dimensions, $\hat{h}_i^{i,j}$ is the approximate marginal relative frequency function for dimension i that is computed from $\hat{h}^{i,j}$, and similarly for dimensions k and l. More generally, the model can deal with conditional independence relationships, e.g.,

in which dimensions i and k are independent, given the value of a third dimension j. For this example, the cliques would be $[i,j]$ and $[j,k]$, and we estimate $h_{i,j,k}(u,v,w)$ as $\hat{h}_{i,j,k}(u,v,w) = \hat{h}^{i,j}(u,v)\hat{h}^{j,k}(v,w)/\hat{h}^{j,k}_{j}(v)$ or, equivalently, $\hat{h}_{i,j,k}(u,v,w) = \hat{h}^{i,j}(u,v)\hat{h}^{j,k}(v,w)/\hat{h}^{i,j}_{j}(v)$. To limit computational complexity, cliques are constrained to contain at most three or four dimensions. The interaction model is fitted using a heuristic "forward selection" search process, in which full independence is assumed initially—so that there is exactly one singleton clique per dimension—and cliques are built up incrementally based on improvements in the approximation, as measured by decreases in Kullback-Liebler distance between h and $\hat{h}$. For a given proposed interaction model, the number of buckets per histogram is allocated using a greedy algorithm, and bucket boundaries within each histogram are also selected in a greedy fashion. The resulting overall candidate model is then scored. Lim et al. [215] have developed a feedback-based variant of the foregoing algorithm, which they call SASH.

Getoor et al. [120] explore an approach somewhat similar to the one above, but based on a *Bayesian network* representation. Here the network is represented as a directed acyclic graph, and a given dimension is independent of its non-descendants, given its parents. In this framework, the relative frequency distribution is represented via a set of conditional univariate probability distributions, where the distribution of a given dimension is conditioned on the values of its parents. As in [84], the statistical graphical model is fitted to the data using a heuristic search through possible correlation structures. For a given candidate structure, the needed conditional probabilities are simply estimated via sample frequencies when the number of frequency values is not too large; otherwise, bucketing techniques analogous to those in [84] must be used.

Both of the foregoing techniques can be modified to use more sophisticated types of histograms of the type discussed previously, but the construction costs will increase accordingly. In general, the construction costs for these algorithms are fairly heavy, but can potentially be alleviated using, e.g., sampling techniques. In any case, the decomposition of the entire joint distribution into a family of related distributions seems necessary in order to summarize high-dimensional data using histograms, so this line of research deserves further exploration.

Approximating a Hierarchical Datacube

In a somewhat different setting, Poosala and Ganti [255] consider the problem of identifying an optimal configuration of histograms—MHIST histograms in particular—that approximate the different subcubes of a *hierarchical datacube*. A hierarchical datacube in d dimensions comprises a collection of $2^d - 1$ subcubes, where a subcube corresponding to the dimensions in the set $\mathscr{D}' \subseteq \mathscr{D} = \{1, 2, \ldots, d\}$ is obtained by aggregating over the dimensions in $\mathscr{D} \setminus \mathscr{D}'$. Thus a given subcube corresponding to a set $\mathscr{D}'$ of dimensions can be approximated directly by a $|\mathscr{D}'|$-dimensional histogram, or may be approximated indirectly by first approximating a subcube with dimension set $\mathscr{D}'' \supset \mathscr{D}'$, and then aggregating over the dimensions in $\mathscr{D}'' \setminus \mathscr{D}'$. The problem is to identify a collection of subcube histograms that occupy minimal space, subject to an upper bound on the allowable maximum approximation error over all subcubes. To approximately answer a given OLAP query once a solution has been obtained, all approximated subcubes whose dimensions cover the query dimensions are considered, and the subcube having minimum average relative approximation error—as computed during histogram creation—is used. (This technique contrasts with the other techniques in this section, in which multiple partial histograms are combined together, via assumptions of independence or conditional independence, to approximately answer a query.) The authors show that a solution to the minimum weighted set cover problem (MWSC) yields a set of histograms that satisfy the error bound and whose space requirement is within a constant factor of the space required for the optimal configuration. A standard greedy algorithm for solving MWSC is employed to compute the near-optimal histogram configuration.

3.6 Approximate Processing of General Queries

The discussion so far has concentrated on the use of histograms for approximate answering of range-sum queries, which are central to OLAP applications. As we have seen, there has been an enormous amount of research in this area. A smaller but perhaps even more important line of work has attempted to extend histogram methodology to the more general setting of SQL queries that involve multi-table operations such as joins. Work in this area has fo-

cused on the estimation of COUNT queries over the result of a select-project-join (SPJ) query over a set of tables (typically in the context of selectivity estimation for query optimization), as well as approximating the results of set-valued queries. In this section, we describe the key results in this area, which still poses a tremendous challenge to histogram technology.

3.6.1 Aggregation Queries Over Joins

In this section, we consider the problem of approximating the answer to a simple `COUNT(*)` query over a table that is itself obtained via a join operation over a set of base tables. (In the query optimization setting, this task corresponds to estimating the size of a multi-table join.) As pointed out by Dobra [87], this framework extends straightforwardly to queries of the form

`SELECT AGG(`$h_1(A_1)*h_2(A_2)*\cdots*h_k(A_k)$`)`
`FROM` $R_1, R_2, \ldots, R_k$
`WHERE ...`

where each h_i is a real-valued function on a set A_i of attributes from R_i, the `WHERE` clause contains local and join predicates, and `AGG` is an aggregation function such as `SUM`, `COUNT`, `AVG`, or `STDDEV`. As discussed below, there have been a number of theoretical results that characterize the difficulty of this problem, that compare the relative effectiveness of histograms and other types of synopses, and that provide some guidance in configuring a histogram to answer such queries.

A Negative Result

The database community first contributed to the theory of histograms via the work of Ioannidis and Christodoulakis [181]. The study in [181] focuses on a setting in which all joins are equality joins, but need not be key to foreign-key joins. Exactly one column in each table serves as the join column for all joins in which the table participates ("t-clique" joins). Individual single-bucket histograms are maintained on the join columns, using the continuous-values estimation scheme within the bucket. The analysis in [181] shows that the estimation error grows exponentially as the number of joins increases, indicating that this approach, in general, will work poorly, unless each histogram is exceedingly accurate or the estimation problem (data and or query)

has special properties that can be exploited (see below).

Refinements: Relative Effectiveness of Histograms

Dobra [87], refines the above theoretical results by identifying some special conditions under which accurate results can be obtained, and compares histograms to samples and sketches in several scenarios. To highlight the main ideas, we focus on the case of size estimation for a simple two-table equality join, where, for each table R and S, a single-bucket uniform histogram is maintained on the join column. Suppose that the join column values have domain $\mathscr{U} = [1,M]$, and, for $i \in \mathscr{U}$, denote by r_i and s_i the frequency of the ith join value in R and S. The histogram-based estimate of the join size is simply $M\bar{r}\bar{s}$, where $\bar{r}$ is the average of $r_1, r_2, \ldots, r_M$, and similarly for $\bar{s}$. Clearly, this estimate will be accurate if $r_i \approx \bar{r}$ and $s_i \approx \bar{s}$ for all $i \in \mathscr{U}$, but such a uniformity condition is usually violated in practice. A more general condition allows both the r_i's and the s_i's to be heterogeneous, but requires that there be no "systematic relationship" between r_i and s_i as i varies. Conceptually, one can envision specifying the r_i's and the s_i's, but then randomly permuting the s_i's to break any relationship, i.e., to replace each s_i by $s_{\sigma(i)}$, where σ is a random permutation of the elements of $\mathscr{U}$. This model is called the *random-shuffling assumption*, and it can be shown that, under this probabilistic model, the histogram estimate is unbiased for the true join size, and is close to the true join size with high probability.[8]

Dobra extends this result to multi-bucket histograms which are "aligned", in the sense that the R-histogram and S-histogram use the same set of buckets. He then shows that, under the random shuffling assumption, aligned histograms are much more accurate, on average, than samples and sketches. On the other hand, when the random-shuffling assumption does not hold, aligned histogram estimates are poor, on average, unless the histogram sizes are close to the column sizes. In particular, for estimating the size of a self join, aligned histograms are shown to perform fundamentally worse, on average, than samples or sketches. As pointed out in [87], however, non-aligned histograms, such as high-biased histograms, may do substantially better than aligned histograms, so no general conclusions can be drawn in this case. The foregoing

[8] The unbiasedness result was originally proved by Ioannidis [178], but for a different purpose.

results are theoretical in that they cannot be applied a priori to choose between synopses for a specific query workload and set of tables.

Kaushik et al. [205] provide a number of fundamental theoretical results pertaining to the relative effectiveness of histogram synopses for estimating the size of general SPJ queries. These results analyze the space complexity of histograms and other types of synopses. Under a general estimation model that includes both deterministic and probabilistic estimation methods, they establish the following results in a precise, information-theoretic sense:

- Under the usual absolute, ratio, and relative error metrics, and with no assumptions on the data distribution, efficient estimation is impossible for any synopsis, in that it is impossible to simultaneously guarantee "small" errors (constant or polylogarithmic in the data size) and a small synopsis size, even for single-column equality selection queries on a single table; c.f. the informal example in Section 3.1.1.
- Under a looser "threshold" error criterion, which only tries to estimate whether the result size is smaller or larger than a specified threshold, efficient estimation is possible for single-column selections as above but not for range selection queries.
- For single-table selections, histograms are the most space-efficient type of synopsis; the resulting errors, although "large" in the above sense, are typically acceptable in practice, validating the observed efficacy of histograms in this setting.
- For specific, skewed, data distributions and a given space budget, the worst case error guarantees can be substantially improved, again supporting practical experience.
- The space complexity for handling COUNT queries over arbitrary joins is substantially higher that the space complexity for single-table queries; this result complements and strengthens the results in [181].
- The above negative result does not hold for the special case of key-foreign key joins, and an effective technique for handling such queries is to use as a synopsis a small sample from the precomputed join, in an extension of the "join synopsis" technique of Acharya et al. [3]. Kaushik et al. [205] show that their proposed

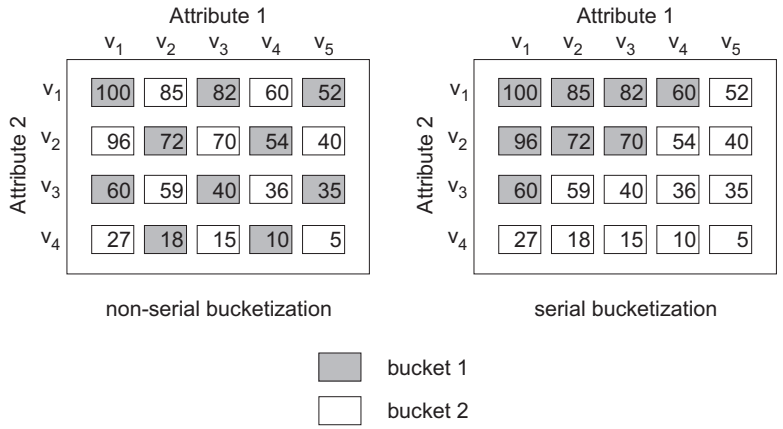


Fig. 3.14 A non-serial and a serial bucketing scheme

synopsis has near-optimal space complexity.

A histogram-oriented analogue to the above synopsis of Kaushik et al. is the notion of Statistics on Intermediate Tables (SITs) introduced for use in the query optimizer for the SQL Server product by Bruno and Chaudhuri [29]; In DB2 for LUW [52], SITs are called "statistical views." The idea is to create histograms, rather than samples, on intermediate query results. The major challenges of this approach (and to that of Kaushik et al.) are: (a) determining for which of the many sub-expressions of an SQL workload the system should collect SITs, and (b) ensuring that the system can effectively exploit the SITs. Standard view-matching algorithms can potentially be used to handle challenge (b); challenge (a) has not received attention in the general setting of approximate query answering.

Guidance on Histogram Configuration

As can be seen, using a histogram to estimate even a simple COUNT query over joins is a challenging problem. The foregoing discussion raises the question of whether there is any theoretical guidance on classes of histograms that will minimize overall estimation error. The results on this problem are rather limited.

Ioannidis and colleagues [178, 182, 183] consider a class of general

equality-join queries (not necessarily t-clique), where, for each table, a multi-dimensional histogram is maintained on the set of all join attributes. For a given set of distinct-value frequencies per table, it is shown that, when these frequencies are assigned to values such that the join-result size is maximized—which corresponds to the worst-case absolute estimation error for the histograms—this worst-case error is minimized by using serial histograms. Such histograms generalize the definition in Section 3.4 for the one-dimensional case, and group values into buckets based on similarity of frequencies; Figure 3.14, adapted from [183], illustrates a frequency distribution for two-dimensional data, along with a non-serial and a serial bucketing scheme (for two buckets).[9] For t-clique queries with a large number of joins, estimation error is minimized using high-biased histograms—in which, as discussed in Section 3.4.1, the values with the highest frequencies are assigned to singleton buckets and the remaining values are assigned to a common bucket.

Some results on choosing the optimal serial histogram for a given join attribute are presented in [178, 182]. These results only apply to the worst-case-error scenario described above, and depend on the both specific query of interest and the precise contents of all of the tables; thus the optimal histograms are sensitive to changes in the data. It is shown in [183] that, using a v-optimality criterion, optimal histograms can be determined separately for each table, and independent of a particular query. Specifically, it suffices to determine a histogram for a join attribute A on a table R that is optimal with respect to the join of R to itself. This optimal histogram will be a serial histogram in general, which can be extremely expensive to construct and use. Consequently, based on empirical evidence, the authors recommend the use of end-biased histograms. Of course, practically speaking, any of the histograms discussed previously can be considered, once optimality guarantees are no longer being provided.

3.6.2 Set-Valued Queries

Ioannidis and Poosala [184] initiated the study of how to use histogram techniques to approximate the answers to set-valued queries over discrete

[9] Note that, in the figure, the frequencies decrease from top to bottom and from left to right. This arrangement, when applied to each table, maximizes the join size as discussed above.

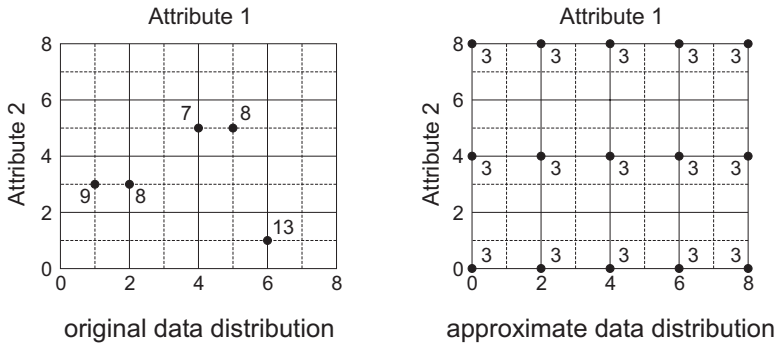


Fig. 3.15 Uniform-spread assumption in two dimensions

data. They assume that a multi-dimensional bucketing scheme has been specified for each table in the database (Section 3.5.1), and use the multi-dimensional version of the uniform-spread assumption (Section 3.3.1) to approximate the data values. That is, given that a bucket—i.e., a hyper-rectangle in d-dimensional space—contains m_i distinct values in the ith dimension ($1 \leq i \leq d$), the set of (multivariate) distinct values present in the bucket is approximated as a set of $m_1 m_2 \cdots m_d$ distinct values, uniformly spread out over the bucket. The frequency for each of these distinct value is the average frequency for the bucket, i.e., the total number of data points in the bucket divided by the quantity $m_1 m_2 \cdots m_d$. See Figure 3.15 for an example in two dimensions, involving an 8×8 bucket; in the figure, black dots correspond to distinct data values, with the corresponding frequency displayed next to the value. Thus the original data in the bucket comprises 45 tuples having five distinct tuple values $(1,3)$, $(2,3)$, $(4,5)$, $(5,5)$, and $(6,1)$, with respective frequencies 9, 8, 7, 8, and 13; here there are $m_1 = 5$ distinct values (1, 2, 4, 5, and 6) in dimension 1 and $m_2 = 3$ values (1, 3, and 5) in dimension 2. The approximated data comprises $m_1 m_2 = 5 \times 3 = 15$ distinct tuple values, each with frequency $45/15 = 3$ and spaced evenly throughout the bucket.

The authors perform approximate query processing by, in effect, applying the query Q of interest to the approximate tables $\hat{R}_1, \hat{R}_2, \ldots, \hat{R}_k$ corresponding to the histogram synopses $H_1, H_2, \ldots, H_k$ of the original tables $R_1, R_2, \ldots, R_k$, thereby producing an output table $\hat{S}$ that approximates the true result table S. To achieve acceptable performance, the system actually proceeds by storing each histogram H_i in relational form and transforming Q into an equivalent

query Q' that is applied directly over $H_1, H_2, \ldots, H_k$ to produce a result histogram H', which is then expanded to the (approximate) result relation $\hat{S}$. The query Q' is generated so as to ensure that H' is indeed a synopsis of $\hat{S}$.

To illustrate the process, consider a table `R` having a single attribute `A` and let H be a histogram synopsis of `R`. Assume that H is stored as a relation `H` with schema `H(lo, hi, dv, avg, sp)`. Each row of `H` represents a histogram bucket, having bucket boundaries `lo` and `hi`, and containing `dv` distinct values. The columns `avg` and `sp` represent the average frequency and spread of the values in the bucket, and are used to approximate the data distribution in the bucket (using the uniform-spread assumption as previously described). All 1D-histograms in the system share this schema. Consider the selection query Q: `SELECT A FROM R WHERE A = c`. We rewrite this query to a new query Q' over `H`:

```
SELECT c, c, 1, avg, 0
FROM H
WHERE (c >= lo) AND (c <= hi) AND (MOD(c-lo,sp) == 0)
```

If the selection is nonempty, then this query creates an output histogram `H'` having exactly one singleton bucket containing the value `c`, i.e., a bucket for which the lower and upper boundaries both equal `c`, the number of distinct values is exactly 1, and the average spread is trivially 0. To create this output bucket, the query finds the bucket in `H` that potentially contains the value `c` (via the first two clauses of the `WHERE` expression). Next, the query checks whether `c` is a multiple of the spread (via the final clause of the `WHERE` expression); under the uniform-spread assumption, such multiples, and only such multiples, have positive frequency, namely `avg`, and this `avg` value is returned as the frequency of `c` in the singleton output bucket. A (slightly tricky) SQL query can then be applied to transform H' into an approximate result relation $\hat{S}$ consisting of `avg` rows, each equal to `c`. Observe that, with this approach, we never have to expand H into an approximate relation—which can be as large as the original relation `R`—thereby reducing processing costs. These techniques extend to multi-dimensional histograms and other relational operators besides selection.

In related work, Poosala et al. [256] observe that the foregoing technique can also be applied to aggregation queries over relational expressions. The idea is to first proceed as above in computing an output histogram H' which is

a synopsis of an approximation to the relational expression. Then, instead of expanding H' into an approximate output relation as above, we simply apply the original aggregation operator to H'—after rewriting it appropriately—to produce an approximate answer to the original aggregation query. E.g., consider a histogram H' as above for a table $\hat{S}$ that approximates a single-column table S of sales data, where S is computed via join queries over multiple base tables and $\hat{S}$ is created by executing analogous queries over histogram synopses of the base tables. If the original aggregation query is `SELECT SUM(SALES) FROM S`, then the answer to this query can be approximated as the answer to the query

```
SELECT SUM(avg*(dv*lo+0.5*dv*(dv-1)*sp))
FROM H'
```

This query, in effect, computes the answer to the query `SELECT SUM(SALES) FROM` $\hat{S}$. Observe that, for a given bucket of the histogram H', the transaction amounts that occur with positive frequency are `lo`, `lo+sp`, `lo+2*sp`, ..., `lo+(dv-1)sp` under the uniform-spread assumption; these numbers sum to `dv*lo+0.5*dv*(dv-1)*sp`. Because each of these values occurs with frequency `avg`, the total contribution to the sum of sales from the bucket is `avg*(dv*lo+0.5*dv*(dv-1)*sp)`. The above query simply sums up the contributions from the different buckets.

3.7 Additional Topics

We conclude this chapter with a discussion of histograms over streaming data, as well as techniques targeted specifically toward real-valued data.

3.7.1 Histograms Over Streaming Data

As mentioned at the end of Section 3.4.2, histograms over streaming data are needed to handle the relational data access scenario. Such histograms are becoming ever more important in their own right, due to the increasing importance of streaming data, such as the data from sensor and IP networks. In this section we describe algorithms for maintaining histograms over streaming data. We first discuss methods for one-dimensional data, and then briefly describe extensions to multidimensional data.

We consider *turnstile* and *sliding-window* streaming-data models. The

turnstile model was described at the end of Section 3.4.2: each new transaction is of the form "increment the current value of $g(i)$ by v." When v is constrained to be a positive integer, one obtains the *cash-register* model as a special case. Sliding-window models are described later in the section. We note that the term "streaming data" is sometimes used to describe the scenario where a histogram construction algorithm takes a single pass over a finite set of N items and uses $o(N)$ memory—and often $O(1)$ memory in practice—to construct the histogram. We would maintain that this level of performance is requisite for an algorithm to be practically useful, and do not reserve the term "streaming data" for this scenario.

Early work by Donjerkovic et al. [90] proposed a histogram maintenance algorithm that supports the cash-register model. Their approach, however, does not control the approximation error over time.

Gilbert et al. [128] provide a histogram maintenance algorithm that supports the full turnstile model. The algorithm maintains a sketch based on distance-preserving random projections—see Chapter 5—and in turn uses the sketch to maintain an accurate wavelet representation of the data—see Chapter 4. On demand, the algorithm recovers, with high probability, a near-optimal histogram; i.e., the error is optimal within a factor of $(1+\varepsilon)$, with respect to an L_1 or L_2 error metric. The algorithm, which is rather complicated, has space complexity that is polynomial in $\tau = B\varepsilon^{-1}\log M$, and requires $O(\tau)$ time to process each transaction. Muthukrishnan et al. [240] extend these results to a workload-dependent L_2 metric, i.e., a metric of the form $\sum_{i=1}^{M} w_i\big(g(i)-\hat{g}(i)\big)^2$, where $w=(w_1,w_2,\ldots,w_M)$ represents a weight or probability distribution over possible point queries. Results are given for both the datacube and turnstile models, and for cases where the workload vector w is provided either in compressed form (e.g., using the Ziv-Lempel encoding) or in uncompressed form.

Qiao et al. [259] provide a heuristic adaptive scheme for histogram maintenance (no error guarantees) that splits and merges buckets to try and ensure narrower buckets in regions that are near the boundaries of range queries or values that are queried more frequently. An exponential smoothing approach is used to summarize a changing workload.

The problem of dynamically maintaining an (approximate) equi-depth histogram under the turnstile model is equivalent to the problem of dynamically maintaining (approximate) quantiles under this model. Sketches have

been successfully employed for this problem; see Section 5.3.4.3.

A related approach, also applicable to the turnstile model, maintains a random sample of data values, called a *backing sample*, and constructs an approximate histogram from the sample. Gibbons et al. [127] propose such a scheme for maintaining an approximate equi-depth or compressed histogram; we focus here on the simpler equi-depth case. The algorithm takes an initial sample and then computes a corresponding initial equi-depth histogram. As records are inserted and deleted into the database, the algorithm maintains the backing sample and increments or decrements the bucket counts, as appropriate. Recall that, ideally, the bucket counts for an equi-depth histogram with B buckets should all be equal to $|R|/B$, where $|R|$ is the number of records in the database; if the count for a bucket becomes very large or very small relative to the other bucket counts, then the algorithm recomputes the histogram from scratch, using the current backing sample. The algorithm actually tries to reduce the number of such recomputations by splitting and merging adjacent buckets to even out the counts. The method for maintaining a backing sample handles database insertions by using a standard reservoir sampling algorithm, and handles deletions by removing the tuple from the reservoir, if present. Whenever a tuple is removed from the reservoir, the method decreases the reservoir size by 1; when the reservoir size reaches a specified lower threshold, the method recomputes the sample from scratch. Gemulla et al. [117, 118] give improved algorithms for maintaining a backing sample, which can be used in conjunction with the foregoing histogram maintenance algorithm.

Buragohain et al. [38] show that their Min-Increment algorithm—see the discussion of maximum-error metrics in Section 3.4.2—can be extended to a *sliding-window* model. In this model, the goal is to maintain, after a total of n g-values have been observed, a histogram of the most recent w of these n values, i.e., of the values $\{g(n-w+1), g(n-w+2), \ldots, g(n)\}$. We assume here that each $g(i)$ value belongs to some finite subset of the integers that contains R elements. Such an algorithm must take only one pass over the data, have low per-item processing cost, and have a $o(w)$ space complexity. The extended version of Min-Increment is the first such algorithm having optimality guarantees. Specifically, the L_∞ error is within a $(1+\varepsilon)$ factor of the optimal error for a B-bucket histogram, while using at most $B+1$ buckets; the memory requirement is $O(\varepsilon^{-1} B \log R)$, which is indeed $o(w)$, and a processing time of

$O(\varepsilon^{-1}B\log R)$ per arriving data item. The algorithm is similar to the original Min-Increment algorithm in that it maintains a set of histograms as in the Greedy-Insert algorithm, one for each error bound e_i as defined previously. The difference is that a histogram bucket is eliminated if and only if either (1) all of the data points in the bucket fall outside the window or (2) the number of buckets exceeds $B+1$, in which case the oldest bucket is deleted. Observe that the number of buckets may exceed B; it might be hoped that, by increasing the number of buckets by some factor β, the L_∞ error bound could be reduced to match that of the optimal B-bucket histogram, thereby eliminating the $(1+\varepsilon)$ approximation factor. The authors prove, however, that this is impossible. Using a dynamic-programming approach, Guha et al. [139] provide a sliding-window algorithm called AHIST-L-Δ that maintains a data structure of size $O(w+B^2\varepsilon^{-1})$ with $O(1)$ processing cost per arriving data item; an ε-approximate histogram can be produced on demand for a time complexity of $O(B^3\log^2 w+\tau B^2\varepsilon^{-1})$, where $\tau=\min(B\varepsilon^{-1}\log w, w)$. Although the space requirements are heavy relative to the method of Buragohain et al., there is no restriction to L_∞ error.

Buccafurri and Lax [36] provide a variant of the n-level tree—see Section 3.4.3—suitable for approximately answering range-sum queries over a sliding window. The key difference from the original nLT is that a running range sum for the r most recent g-values is maintained in a buffer, where $r=w/2^{n-1}$ with $w=$ the window size and $n=$ the number of levels in the tree. When the buffer fills up, the sum is added to the tree by overwriting the oldest leaf node and propagating the update to all of the nodes on the path connecting the leaf to the root. In this way, the per-item update cost, as well as the time to approximately answer a range-sum query, is $O(\log w)$.

The above sliding-window model might more precisely be called a "datacube" sliding-window model, since it is closely related to the datacube model of data access (Section 3.2.2). Note that the $g(i)$ values are partitioned according to time, i.e., according to the i values. A sliding-window setting that corresponds to the relational model of data access assumes a sequence of values $d_1=(j_1,v_1), d_2=(j_2,v_2),\ldots$ as in Section 3.2.2—so that each j_n is an element of $[1,M]$—together with a specified partition of $[1,M]$ into B buckets. Thus data items are assigned to buckets based on the j_n values, which do not correspond, in general, to the arrival order of the items.

Suppose that each $g(i)$ is obtained applying the SUM aggregate to the set $A_i = \{ v_n : j_n = i \}$. This special case can be viewed as a modified turnstile model in which, at the arrival of data item d_i, the value $g(j_i)$ is incremented by v_i, and the value of $g(j_m)$ is decremented by v_m; here the index $m = i - w$ corresponds to the data item that is dropping out of the window.[10] Now further assume that each v_i is an integer lying in the range $[1, R]$ for some $R \geq 1$. Then the sliding-window problem is to maintain a histogram having the specified set of buckets, where the g-values are computed from the last w data items $\{ d_{n-w+1}, d_{n-w+2}, \ldots, d_n \}$. As usual, the most important special case occurs when each v_i equal 1, so that $g(i)$ is the number of the last w data items having value i. Datar et al. [78] provide a data structure called an "exponential histogram" that can be used to maintain, for each bucket, the sum of the g-values to within a factor of $1 + \varepsilon$, thereby approximating the true histogram over the sliding window (where we use the continuous-value assumption within each bucket). The total space requirement is $O\big(B\varepsilon^{-1} \log w (\log R + \log w)\big)$, and the worst-case per-item processing time is $O(\log w + \log R)$.

We now consider multi-dimensional data. Several of the foregoing methods for maintaining 1D-histograms under the general turnstile model can be extended to the multi-dimensional case. In particular, the backing-sample approach of Gibbons et al. can be adapted for use with the multi-dimensional equi-depth histogram of Muralikrishna and Dewitt.

Moreover, Thaper et al. [278] have extended the sketch-based technique of Gilbert et al. [128] to a multi-dimensional setting; see also Section 5.3.5.1. Recall that this algorithm maintains a sketch of the data stream based on random projections, and then produces a near-optimal histogram on demand. For the multidimensional algorithm, the time to produce a histogram having error at most $(1 + \varepsilon)$ times the minimum possible error is roughly $O\big(M \log(MB/\varepsilon^2)\big)$. Muthukrishnan and Strauss [239] improve on this result using advanced techniques based on tensor products of Haar wavelets.

3.7.2 Techniques for Real-Valued Data

The discussion so far has focused on discrete data. Some experimental evidence in Blohsfeld et al. [23] and Korn et al. [210] indicates that, for COUNT

[10] Note that turnstile algorithms cannot be applied to this problem, since the value v_m is not available when d_i arrives.

queries on real-valued data, superior performance might be obtained by using (and extending) classical statistical methods, which were originally developed under the assumption that the data are i.i.d. samples from a continuous probability density function (see Section 3.1.3). These methods are also potentially desirable for the case of discrete data in which the domain size M is large and the multiplicity of each distinct value in the dataset is close to 1. We loosely describe such data also as "real-valued."

The basic tool used is *kernel density (KD)* estimation [86, 163, 270, 273, 285], which has been extensively studied in the statistical literature. We briefly describe this approach below, and the discuss its application in the database setting, first to one-dimensional data and then to multi-dimensional data.

One motivation behind KD methods that the bucket values—and hence the estimates—of a classical equi-width histogram can be very sensitive to the placement of the bucket boundaries. This suggests that multiple histograms be obtained from a "base" equi-width histogram by systematically shifting the histogram (i.e., uniformly shifting the bucket boundaries) by various amounts. Approximate COUNT values are then obtained by averaging the results from these histograms. Letting the number of shifted histograms increase to infinity, we obtain a special case of a KD estimator having a "rectangular kernel".

The general form of a one-dimensional KD estimator based on data values $x_1, x_2, \ldots, x_n$ is

$$f_h(u) = \frac{1}{nh} \sum_{i=1}^{n} K\Big(\frac{u - x_i}{h}\Big),$$

where the function K is called the *kernel* and the positive, real-valued smoothing parameter h is called the *bandwidth*. In general, h can be a function of the data. One theoretical motivation for KD estimators is the fact that, when the observed data are indeed i.i.d. samples from a density function f, then, under very mild conditions, $E\big[\int |f - f_h|\big] \to 0$ as $h \to 0$ and $nh \to \infty$. That is, the function f_h is a consistent estimator of f in an L_1 sense.

Intuitively, each data point x_i influences the value of f_h at a given point x. The kernel K determines the shape of the "influence function" and the bandwidth determines how quickly the influence of x_i dies off with increasing distance; the smaller the value of h, the more rapidly the influence dies off.

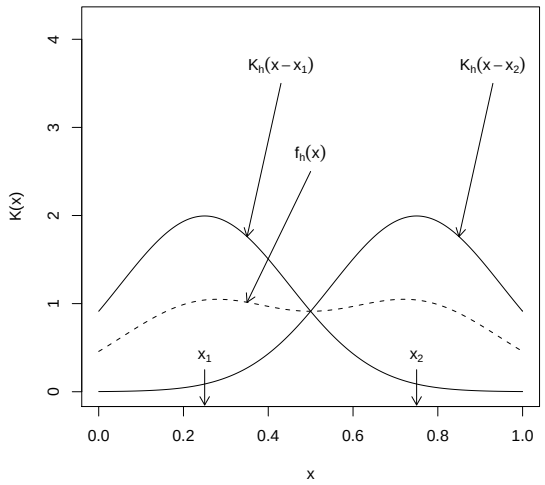


Fig. 3.16 Univariate KD estimation (two training points, bandwidth $h = 0.1$)

Some common one-dimensional kernels include the Gaussian kernel

$$K_G(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

and the Epanechnikov kernel

$$K_E(x) = \frac{3}{4} \max(1 - x^2, 0)$$

both defined for all real x. See Figure 3.16 for an illustration of these ideas, using the Gaussian kernel. In the figure and in the sequel, we use the notation $K_h(x) = h^{-1} K_G(x/h)$. Experience in a variety of applications has shown that the overall estimation quality is more sensitive to the choice of bandwidth than to the form of the kernel [270]. Indeed, the problem of selecting an optimal or near-optimal bandwidth for a given kernel and set of training data is very challenging, and the subject of much research. There are a variety of rules-of-thumb, as well as more elaborate schemes, such as the cross-validation techniques in [86] or the Markov chain Monte Carlo techniques in [297].

In the database setting, KD estimators have been used to approximately answer range-count queries. Such a query estimates the fraction of data points that fall in a specified range $[a,b]$. The estimated answer is $\int_a^b f_h(u)\,du = (1/n)\sum_{i=1}^n \int_a^b K_h(u-x_i)\,du$. This expression can be simplified further, based on the explicit form of K_h. Note that the space and time complexity of using the KD estimator to answer queries is $O(n)$, which is unacceptable. Therefore, the KD estimator is based on a sample of training points.

Blohsfeld et al. [23] investigate a simple one-dimensional KD estimator that uses the Epanechnikov kernel together with a well known rule-of-thumb for the bandwidth: $h = 2.345sn^{-1/5}$, where s is the minimum of (1) the sample standard deviation of the training points and (2) the interquartile range[11] divided by 1.348. They also modify the basic estimator to provide better estimates near the boundary of the data domain. The authors also consider a hybrid approach in which the data is partitioned into buckets, and then a KD estimator is computed for each bucket.

Korn et al. [210] develop a *KernelSpline* estimator. The estimator is based on a standard KD estimator that uses a Gaussian kernel, together with a bandwidth formula given by $h = 0.25(\log_2 n + 1)$. KernelSpline uses the KD estimator to estimate the value of f_h at a fixed set of p grid points during a single pass over the data, incurring a time complexity of $O(n)$ and a space complexity of $O(p)$. The algorithm then uses a cubic-spline interpolation scheme to estimate $f_h(x)$ for arbitrary values of x, thus approximating the range-query result $\int_a^b f_h(u)\,du$ by a simple polynomial expression.

The experiments in [23, 210] both show that the KD based methods provide better accuracy than maxdiff histograms for real-valued data, reducing the error of some estimates by almost a factor of 6. Thus there is an argument for using specialized techniques when dealing with real-valued data.

The foregoing KD techniques for one-dimensional data can be extended to multidimensional data; this fact was explicitly recognized in the database setting in [143]. The general form of a D-dimensional estimator is

$$f_h(u) = \frac{1}{nh_1h_2\cdots h_D}\sum_{i=1}^{n} K\Big(\frac{u_1-x_{i1}}{h_1}, \frac{u_2-x_{i2}}{h_2}, \ldots, \frac{u_d-x_{iD}}{h_D}\Big),$$

where now $h = (h_1, h_2, \ldots, h_D)$ and $x_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$. For example, K

[11] The IQR is defined as the 75th percentile minus the 25th percentile.

might be a multivariate Gaussian density or a product of one-dimensional kernels. To simplify computations, the above model is often specialized to an axis-aligned, product-form estimator:

$$f_h(u) = \frac{1}{nh_1h_2\cdots h_D}\sum_{i=1}^{n} K_1\Big(\frac{u_1 - x_{i1}}{h_1}\Big)K_2\Big(\frac{u_2 - x_{i2}}{h_2}\Big)\cdots, K_d\Big(\frac{u_d - x_{iD}}{h_D}\Big),$$

where each K_i is a one-dimensional KD estimator. This form is used in [143], with each K_i equal to the Epanechnikov kernel. Only one pass through the data (or a sample of the data) is needed to construct a KD estimator, and so there is a significant cost advantage. The downside, indicated by experiments in [143], is an increase in relative error relative to techniques such as GENHIST (up to 250% in some experiments).

4

Wavelets

4.1 Introduction

Wavelets are a useful mathematical tool for hierarchically decomposing datasets in ways that are both efficient and theoretically sound. Broadly speaking, the wavelet transform of a dataset consists of a coarse overall approximation together with detail coefficients that influence the dataset at various scales. The wavelet transform has a long history of successful applications in signal and image processing [187, 222, 276]. More recently, several studies have demonstrated the effectiveness of the wavelet transform as a basic tool for approximate query processing over massive relational tables and continuous data streams. Briefly, the idea is to apply the wavelet transform to the input relation to obtain a compact data synopsis that comprises a select small collection of *wavelet coefficients*. The excellent energy compaction and decorrelation properties of the wavelet transform allow for concise and effective approximate representations that exploit the structure of the data. Furthermore, wavelet transforms are generally simple linear transformations, thus allowing for efficient synopsis-construction algorithms.

Compared to sampling and histograms, wavelets are a relative newcomer to the field of approximate query processing, and they have not yet been

adopted for use in commercial database systems. Still, the area of wavelet-based approximation has seen intense interest from database and algorithms researchers in the last few years, and several novel tools and techniques with potential for practical use have been developed. In the remainder of this chapter, we first introduce the key concepts of the wavelet transform and wavelet-based data synopses, focusing on the one-dimensional Haar wavelet decomposition. Our initial discussion concentrates on the construction and properties of wavelet synopses optimized for L_2 (i.e., sum-squared) error metrics. We then proceed to cover more recent results on the challenging problem of optimizing wavelet synopses for *non-*L_2 error measures. As in the case of histograms, perhaps due to the inherent limitations of the infamous "curse of dimensionality", the vast majority of algorithmic and theoretical work on wavelet synopses has concentrated on the one-dimensional case. Later in the chapter, we discuss multi-dimensional wavelets and techniques for efficient approximate query processing over wavelet synopses. Finally, we conclude the chapter by discussing wavelets over streaming data, techniques that optimize synopsis storage (e.g., for multi-measure data or better compression), and "hybrid" synopses that combine ideas from both wavelets and histograms.

Following the bulk of the database literature on wavelets, our discussion in this chapter focuses primarily on the *Haar Wavelet Transform (HWT)*. Conceptually, Haar is probably the simplest wavelet basis (based on recursive pairwise averaging and differencing), and the resulting wavelets are easy to compute and have been found to perform well in practice for a wide variety of applications ranging from image editing and querying to OLAP and streaming-data approximations. Furthermore, many of the ideas and techniques developed for Haar wavelets naturally carry over to more sophisticated wavelet bases.

4.2 One-Dimensional Wavelets and Wavelet Synopses: Overview

We start by defining one-dimensional Haar wavelets and exploring some of their key properties as a data-reduction tool. We then describe a simple, one-pass algorithm for building wavelet synopses optimized for L_2-error, as well as the use of such synopses for range-query estimation. Finally, we draw some parallels between wavelets and histograms as data-reduction mechanisms.

4.2.1 Haar Wavelets: Definitions and Key Properties

Suppose we are given the one-dimensional data vector A containing the $M = 8$ data values $A = [2,2,0,2,3,5,4,4]$. The HWT of A can be computed as follows. We first average the values together pairwise to get a new "lower-resolution" representation of the data with the following average values $[2,1,4,4]$. In other words, the average of the first two values (that is, 2 and 2) is 2, that of the next two values (that is, 0 and 2) is 1, and so on. Obviously, some information has been lost in this averaging process. To be able to restore the original values of the data (treated as a data array or vector), we need to store some *detail coefficients*, that capture the missing information. In Haar wavelets, these detail coefficients are simply the differences of the (second of the) averaged values from the computed pairwise average. Thus, in our simple example, for the first pair of averaged values, the detail coefficient is 0 since $2-2=0$, for the second we again need to store -1 since $1-2=-1$. Note that no information has been lost in this process — it is fairly simple to reconstruct the eight values of the original data array from the lower-resolution array containing the four averages and the four detail coefficients. Recursively applying the above pairwise averaging and differencing process on the lower-resolution array containing the averages, we get the following full decomposition:

Resolution	Averages	Detail Coefficients
3	[2, 2, 0, 2, 3, 5, 4, 4]	—
2	[2, 1, 4, 4]	[0, -1, -1, 0]
1	[3/2, 4]	[1/2, 0]
0	[11/4]	[-5/4]

The *wavelet transform* (also known as the *wavelet decomposition*) of A is the single coefficient representing the overall average of the data values followed by the detail coefficients in the order of increasing resolution. Thus, the one-dimensional HWT of A is given by $W_A = [11/4, -5/4, 1/2, 0, 0, -1, -1, 0]$. Each entry in W_A is called a *wavelet coefficient*. The main advantage of using W_A instead of the original data vector A is that for vectors containing similar values most of the detail coefficients tend to have very small values. Thus, eliminating such small coefficients from the wavelet transform (i.e.,

treating them as zeros) introduces only small errors when reconstructing the original data, resulting in a very effective form of lossy data compression.

Note that, intuitively, wavelet coefficients carry different weights with respect to their importance in rebuilding the original data values. For example, the overall average is obviously more important than any detail coefficient since it affects the reconstruction of all entries in the data. In order to equalize the importance of all wavelet coefficients, we need to *normalize* the final entries of W_A appropriately. A common normalization scheme is to scale each wavelet coefficient by $\sqrt{M/2^l} = 2^{(\log M - l)/2}$, where M is the number of input data values and l denotes the *level of resolution* at which the coefficient appears (with $l = 0$ corresponding to the "coarsest" resolution level). Thus, the normalized i^{th} HWT coefficient, c_i^*, is simply $c_i^* = c_i \sqrt{\frac{M}{2^{\texttt{level}(c_i)}}}$. (As we discuss later, this normalization scheme ensures the *orthonormality* of the underlying Haar wavelet basis.)

The Haar-Tree Representation. A helpful tool for exploring and understanding the multi-resolution nature and key properties of the HWT is the *Haar-tree* structure [225]. The Haar tree is a hierarchical structure built based on the wavelet transform process (even though it is primarily used as a conceptual tool, a Haar tree can be easily constructed in linear $O(M)$ time). Figure 4.1 depicts the Haar tree for our simple example data vector A. Each internal node c_i ($i = 0, \ldots, 7$) is associated with a wavelet coefficient value, and each leaf $A[i]$ ($i = 0, \ldots, 7$) is associated with a value in the original data array; in both cases, the index i denotes the positions in the (data or wavelet transform) array. For example, c_0 corresponds to the overall average of A. Note that the values associated with the Haar tree nodes c_j are the *unnormalized* coefficient values; the resolution levels l for the coefficients (corresponding to levels in the tree) are also depicted. (We use the terms "node", "coefficient", and "node/coefficient value" interchangeably in what follows.)

Given a Haar tree T and an internal node t of T, $t \neq c_0$, $\texttt{leftleaves}(t)$ ($\texttt{rightleaves}(t)$) denotes the set of leaf (i.e., data) nodes in the subtree rooted at t's left (resp., right) child; furthermore, we let $\texttt{leaves}(t) = \texttt{rightleaves}(t) \cup \texttt{leftleaves}(t)$, i.e., the set of all leaf nodes in t's subtree. Also, given any (internal or leaf) node u, $\texttt{path}(u)$ is the set of all (internal) nodes in T that are proper ancestors of u (i.e., the nodes on the path from u to

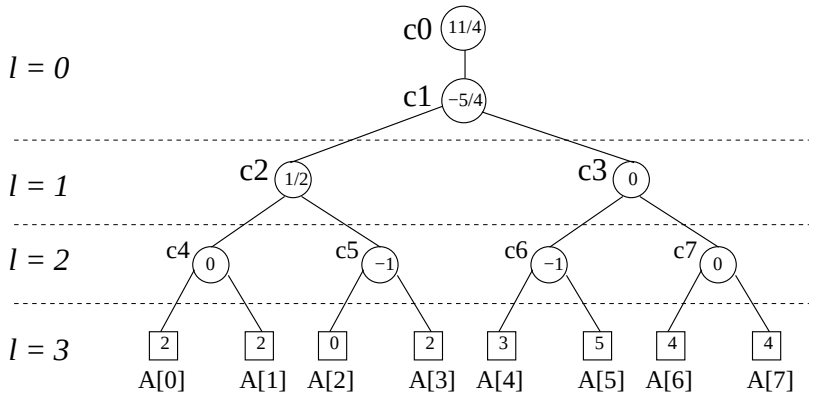


Fig. 4.1 Haar-tree structure for our example data array A ($M = 8$).

the root of T, including the root but not u) with non-zero coefficients. A key property of the HWT is that the reconstruction of any data value $A[i]$ depends only on the values of coefficients on $\texttt{path}(A[i])$; more specifically, we have

$$A[i] = \sum_{c_j \in \texttt{path}(A[i])} \mathsf{sign}_{i,j} \cdot c_j, \tag{4.1}$$

where $\mathsf{sign}_{i,j} = +1$ if $A[i] \in \texttt{leftleaves}(c_j)$ or $j = 0$, and $\mathsf{sign}_{i,j} = -1$ otherwise. Thus, reconstructing any data value involves summing at most $\log M + 1$ coefficients. For example, in Figure 4.1, $A[4] = c_0 - c_1 + c_6 = \frac{11}{4} - (-\frac{5}{4}) + (-1) = 3$.

The *support* of a coefficient c_i (denoted $\texttt{support}(c_i)$) is defined as the region of (contiguous) data values that c_i is used to reconstruct (i.e., the range of data/leaf nodes in the subtree rooted at c_i); the support of a coefficient c_i is uniquely identified by its coordinate i. Note that the supports of all coefficients at resolution level l of the HWT are exactly the 2^l (disjoint) *dyadic ranges* of size $M/2^l = 2^{\log M - l}$ over the domain $\mathscr{U} = [0, M-1]$, defined as

$$R_{l,k} = [k \cdot 2^{\log M - l}, \ldots, (k+1) \cdot 2^{\log M - l} - 1] \qquad \text{for } k = 0, \ldots, 2^l - 1,$$

for each resolution level $l = 0, \ldots, \log M$. [1] Furthermore, the supports of the Haar wavelet coefficients are naturally *nested* across levels: Given any pair

[1] To simplify the exposition in this chapter, we assume that the domain $\mathscr{U}$ is indexed starting from 0 and that the domain size M is a power of 2 (e.g., by padding the array with zero entries).

of (distinct) coefficients c_i and c_j, their support sets are either completely disjoint (i.e., $\texttt{support}(c_i) \cap \texttt{support}(c_j) = \emptyset$) or one is completely contained within the other (i.e., $\texttt{support}(c_i) \subset \texttt{support}(c_j)$ or $\texttt{support}(c_j) \subset \texttt{support}(c_i)$).

The Haar Wavelet Basis for $\mathbb{R}^M$. The mathematical foundation of the HWT relies on vector inner-product computations over the vector space $\mathbb{R}^M$ using the Haar wavelet basis. In general, a wavelet basis $\{\phi_i\}_{i=0}^{M-1}$ for $\mathbb{R}^M$ is a basis where each vector is constructed by dilating a single function, referred to as the *mother wavelet* ϕ. The Haar mother wavelet is defined as:

$$\phi_H(t) = \begin{cases} 1 & \text{if } 0 \le t < 1/2 \\ -1 & \text{if } 1/2 \le t < 1 \\ 0 & \text{otherwise.} \end{cases}$$

The Haar wavelet basis for $\mathbb{R}^M$ is composed of the vectors

$$\phi_{l,k}[i] \;=\; \sqrt{\frac{2^l}{M}} \cdot \phi_H\left(\frac{i - k \cdot 2^{\log M - l}}{2^{\log M - l}}\right) \;=\; \sqrt{\frac{2^l}{M}} \cdot \phi_H\left(\frac{i \cdot 2^l - kM}{M}\right), \quad (4.2)$$

where $i \in [0, M-1]$, $l = 0, \ldots, \log M - 1$ and $k = 0, \ldots, 2^l - 1$, plus their orthonormal complement vector $\psi_M = \frac{1}{\sqrt{M}}\mathbf{1}^M$. (Here $\mathbf{1}^M$ denotes the M-vector whose entries are all equal to 1.) Note that the $\phi_{l,k}$ vectors are essentially dilated and translated versions of the mother wavelet function ϕ_H over the corresponding $R_{l,k}$ dyadic support intervals. To simplify notation, we denote the Haar wavelet basis of $\mathbb{R}^M$ as the collection of vectors $\{\phi_i : i = 0, \ldots, M-1\}$, where $\phi_0 = \psi_M$ and $\phi_i = \phi_{l,k}$ with $l = \lfloor \log i \rfloor$ and $k = i - 2^{\lfloor \log i \rfloor}$ for $i = 0, \ldots, M-1$. It is then not difficult to see that each of the (normalized) coefficients c_i^* $i = 0, \ldots, M-1$ in the HWT of A can be expressed as the inner product of A with the corresponding Haar basis vector ϕ_i; more formally,

$$c_i^* \;=\; \langle A, \phi_i \rangle \;=\; \sum_{j=0}^{M-1} A[j] \cdot \phi_i[j].$$

It can be shown that the above Haar vector basis $\{\phi\}_{i=0,\ldots,M-1}$ is an *orthonormal* basis of $\mathbb{R}^M$; that is, for any pair of vectors ϕ_k, ϕ_l in the basis, $\langle \phi_k, \phi_l \rangle = 1$ if $k = l$ and 0 otherwise. The original data array $A \in \mathbb{R}^M$ can then be expressed in the Haar wavelet basis as a linear combination of the Haar basis vectors using the corresponding HWT coefficients: $A \;=\; \sum_{i=0}^{M-1} c_i^* \phi_i$. Haar wavelets are

also an example of a wavelet system with *compact support*; that is, for any basis vector ϕ_k there exists a closed interval $I = [a,b]$ such that $\phi_k[x] = 0$ for any $x \notin I$. Interestingly, Haar wavelets (discovered in 1910) were the only known wavelets of compact support until the discovery of the Daubechies wavelet families in 1988 [79].

4.2.2 Wavelet-based Data Reduction: Minimizing L_2 Error

Given a limited amount of storage for approximating a large data array A, our goal is to build a *sparse* representation of A in the Haar wavelet basis. Such a sparse representation (also known as *wavelet synopsis*) of A is constrained to use a number of basis elements that is much smaller than the size M of the target data array; of course, the data-reduction benefits of sparsity are counteracted by a loss in the fidelity of the representation and its ability to accurately capture the data. More specifically, a wavelet synopsis of a data array A retains a certain number $B \ll M$ of the coefficients in the wavelet transform of A as a highly-compressed approximate representation of the original data (the remaining coefficients are implicitly set to 0).

A key step here is a *coefficient thresholding* process in order to determine the "best" subset of B coefficients to retain, so that some overall error measure in the approximation is minimized. The thresholding method of choice for most early work on wavelet-based data reduction and approximation [41, 42, 225, 226, 284] is *conventional* coefficient thresholding that greedily retains the *B largest HWT coefficients in absolute normalized value*. Let $\hat{A}$ denote the approximate data array reconstructed by using a subset $\mathscr{B}$ of only $B \ll M$ Haar coefficients in a wavelet synopsis of the original data array A. By the orthonormality of the Haar wavelet basis, the HWT preserves the Euclidean length or L_2-norm of any vector *(Parseval's Theorem)* [222, 276]; then, for the error vector $A - \hat{A}$, we have

$$||A - \hat{A}||_2 \;=\; \sqrt{\sum_{i=0}^{M-1} (A[i] - \hat{A}[i])^2} \;=\; \sqrt{\sum_{c_i^* \notin \mathscr{B}} (c_i^*)^2}. \qquad (4.3)$$

Thus, the conventional thresholding method of retaining the B largest coefficients in absolute normalized value is in fact *provably optimal* with respect to minimizing the overall L_2-norm (or, sum-squared) error in the data approximation for the given amount of space B. Despite its simplicity, L_2-error

thresholding may not always be the best choice for approximate query processing systems — we discuss wavelet thresholding schemes for other error metrics later in this chapter.

Similar to histogram-based summaries, the construction of wavelet synopses can take place during an offline statistics-collection process, whose goal is to create concise statistical approximations for the value distributions of either individual attributes or combinations of attributes in the relations of a DBMS [3, 108, 179]. (Statistics collection is usually an off-line process, carried out during night-time or other system-idling periods.) Once created, such statistical summaries are typically stored as part of the *DBMS-catalog information*. More specifically, a wavelet synopsis comprising B wavelet coefficients can be stored as a collection of B pairs $\{(i_j, c_{i_j}) : j = 1, \ldots, B\}$, where i_j and c_{i_j} denote the index and value (respectively) of the j^{th} retained synopsis coefficient. Wavelet-synopsis information in the DBMS catalog can then be exploited for several different purposes, including result-size estimation for cost-based query optimization and approximate query processing.

4.2.3 Efficient Wavelet Decomposition

As in the case of histograms, we consider two primary data-access models: In the *datacube* model, the data is accessed as described previously, that is, in the form of an M-component data array A. Here, i denotes the cell index in a (one-dimensional) datacube and $A[i]$ is the corresponding "measure value". In contrast, the *relational* model assumes that the data is available simply as an (unordered) unaggregated list of N (index, value) pairs $(i_1, v_1), (i_2, v_2), \ldots, (i_N, v_N)$, where each i_k is an index value in $\mathscr{U} = [0, M-1]$ and v_k is an associated value. The i^{th} entry $A[i]$ of the data array then corresponds to the aggregate of the v_k values appearing with index i in a relational representation. For instance, in the important special case of a frequency distribution array A, we have $v_k = 1$ for all k, and $A[i]$ is the total frequency $f(i)$ of value i in the relational list. Recall that conversion of relational data to the datacube format requires at least one pass over the data, and can be impractical, especially in the case of sparse, multi-dimensional data (considered later in this chapter). Unless otherwise specified, we assume the datacube model in the ensuing discussion.

One of the key benefits of wavelets is that they generally represent a lin-

ear transformation of a data array, and, thus, they are simple and efficient to compute (typically, requiring a single pass over the array). In the case of a one-dimensional data array A, there is an easy one-pass algorithm based on the Haar-tree structure (Figure 4.1) for computing a synopsis comprising the B largest (normalized) Haar coefficients [131]. (Our discussion in this section focuses on L_2 coefficient thresholding.) The algorithm steps through the array entries $A[i]$ (for $i = 0, \ldots, M-1$) maintaining the following two sets of (partial) coefficients:

(1) The B highest HWT coefficient values for the portion of A seen thus far; and,
(2) The current $\log M + 1$ *straddling* partial HWT coefficients, one for each level of the Haar tree. At level l, the kth detail coefficient *straddles* a given data element $A[i]$ if and only if i belongs to the coefficient's dyadic support region, that is, if and only if $i \in [k \cdot 2^{\log M - l}, (k+1) \cdot 2^{\log M - l} - 1]$. (Note that there is at most one such coefficient per level.)

When processing $A[i]$, the value for each of the affected straddling coefficients is updated. Following these updates, some coefficients may no longer be straddling (i.e., their computation is now complete). In that case, the value of these newly-completed coefficients is compared against the current set of B highest coefficients, and only the B largest coefficient values in the combined set are retained. Also, for levels where a straddling coefficient has been completed, a new straddling coefficient is initiated (with an initial value of 0). Thus, using a max-heap for maintaining the top B HWT coefficients, we can compute a B-term wavelet synopsis in a single pass over a one-dimensional data array using $O(B + \log M)$ space and $O(\log M)$ processing time per data item [131].

4.2.4 Range-Aggregate Query Estimation

Typically, concise data synopses are used as a tool for enabling effective (compile-time) estimates of the result sizes of relational operators for the purpose of *cost-based query optimization*. (Accurate estimates of such result sizes play a critical role in choosing an effective physical execution plan for an input SQL query.) Probably the most fundamental such estimation task

is estimating the *selectivity* of (i.e., the number of data tuples satisfying) a range-predicate selection like $l \leq X \leq h$. Assuming our input data array A holds the (one-dimensional) frequency distribution for the target relational attribute X, this estimation task is equivalent to estimating the result of the range-`COUNT` query $A(l:h) = \sum_{i=l}^{h} A[i]$. Based on the rule for data-value reconstruction using Haar wavelets, it is not difficult to see that an internal node c_j contributes to the range-`COUNT` $A(l:h)$ *only if* $c_j \in \texttt{path}(A[l]) \cup \texttt{path}(A[h])$. More specifically, $A(l:h) = \sum_{c_j \in \texttt{path}(A[l]) \cup \texttt{path}(A[h])} x_j$, where

$$x_j = \begin{cases} (h-l+1)\cdot c_j, \text{ if } j = 0 \\ (|\texttt{leftleaves}(c_j, l:h)| - |\texttt{rightleaves}(c_j, l:h)|)\cdot c_j, \text{ otherwise.} \end{cases}$$

where $\texttt{leftleaves}(c_j, l:h) = \texttt{leftleaves}(c_j) \cap \{A[l], A[l+1], \ldots, A[h]\}$ (i.e., the intersection of $\texttt{leftleaves}(c_j)$ with the summation range) and $\texttt{rightleaves}(c_j, l:h)$ is defined similarly [284]. (Note that coefficients whose subtree is completely contained within the summation range have a net contribution of zero, and can be safely ignored.) For instance, in the example array of Figure 4.1, $A(2:6) = 5c_0 + (2-3)c_1 - 2c_2 = 5 \times \frac{11}{4} - (-\frac{5}{4}) - 1 = 14$. In other words, given a B-coefficient synopsis of the d array, computing $A(l:h)$ (for any boundaries l, h) only involves retained coefficients in $\texttt{path}(A[l]) \cup \texttt{path}(A[h])$ and, thus, can be estimated by summing only $\min\{B, 2\log M + 1\}$ synopsis coefficients [284].

4.2.5 The Connection between Histograms and Wavelets

At this point, it is interesting to consider the key similarities and differences between wavelets and histograms as data-reduction tools. Clearly, both wavelets and histograms rely on partitioning the underlying data domain into continuous ranges (i.e., sub-intervals of $[0, M-1]$) and using a piecewise-constant approximation for each data range.[2] A B-bucket histogram defines exactly B piecewise-constant ranges; in contrast, a B-coefficient Haar wavelet synopsis can define anywhere between B and $3B+1$ such ranges. To see this, note that each Haar basis function is a piecewise-constant function of 4 pieces (3 interval boundaries). Thus, a B-coefficient Haar wavelet synopsis can equivalently be seen as a histogram of up to $3B+1$ buckets. This observation seems to suggest that, for a given space budget, Haar wavelets are

[2] In this discussion we focus on basic histograms as in Section 3.3.1.

better than histograms in accurately modeling data sets with a larger number of discontinuities [203]. It is also important to note, however, that while histogram ranges can form any arbitrary partitioning of $[0, M-1]$, the wavelet ranges are (by definition) constrained to a subset of the collection of dyadic ranges $R_{l,k}$ over $[0, M-1]$.

In the other direction, any piecewise-constant function f_I over an arbitrary interval $I \subseteq [0, M-1]$ can be expressed in the Haar wavelet basis as $f_I = \sum_j \langle f_I, \phi_j \rangle \phi_j$, and there are only $2\log M + 1$ basis functions ϕ_j for which the corresponding coefficient (inner product) is non-zero — these are exactly the coefficients whose support intersects an endpoint of I. Thus, a Haar wavelet representation can simulate a histogram over $[0, M-1]$ with at most a $2\log M + 1 = O(\log M)$ blowup in space (i.e., number of required terms). This also naturally implies that an L_2-error optimal Haar wavelet synopsis with $(2\log M + 1)B$ coefficients approximates a given data array $A \in \mathbb{R}^M$ as well as the L_2-optimal (or, *v-optimal*) B-bucket histogram (Section 3.4.2). Guha et al. [137] exploit this observation for the efficient construction of near-optimal histogram representations of streaming data by first building a *robust* approximate representation based on Haar wavelets.

4.3 Wavelet Synopses for Non-L_2 Error Metrics

Conventional wavelet synopses optimized for overall L_2 error may not always be the best choice for approximate query processing systems. The quality of the approximate answers such synopses provide can vary widely, and users have no way of knowing the accuracy of any particular answer. Even for the simplest case of approximating a value in the original data set, the absolute and relative errors can show wide variation. Consider the example depicted in Table 4.1. The first two lines show the 16 original data values (the exact answer), whereas lines 3–4 show the 16 approximate answers returned when using conventional wavelet synopses and storing the 8 largest (normalized) coefficients. Although the first half of the values is basically a mirror image of the second half, all the approximate answers for the first half are 65, whereas all the approximate answers for the second half are exact! [3] Similar data values have widely different approximations, e.g., 30 and 31 have

[3] In this (carefully crafted) example, conventional L_2 thresholding ends up retaining all 8 coefficients in the left part of the Haar tree, and no coefficients from the right subtree [110].

approximations 30 and 65, respectively. The approximate answers make the first half appear as a uniform distribution, with widely different values, e.g., 3 and 127, having the same approximate answer 65. Moreover, the results do not improve when one considers the presumably easier problem of approximating the sum over a range of values: for *all possible* ranges within the first half involving $x = 2$ to 7 of the values, the approximate answer will be $65 \cdot x$, while the actual answers vary widely. For example, for both the range $A[0]$ to $A[2]$ and the range $A[3]$ to $A[5]$, the approximate answer is 195, while the actual answer is 285 and 93, respectively. On the other hand, *exact* answers are provided for all possible ranges within the second half.

Original data values	127	71	87	31	59	3	43	99
	100	42	0	58	30	88	72	130
Wavelet answers	65	65	65	65	65	65	65	65
	100	42	0	58	30	88	72	130

Table 4.1 Errors with Conventional Wavelet Synopses.

The simple example above illustrates that conventional wavelet synopses suffer from several important problems, including the introduction of severe bias in the data reconstruction and wide variance in the quality of the data approximation, as well as the lack of non-trivial guarantees for individual approximate answers. To address these shortcomings, recent work has proposed novel thresholding schemes for building wavelet synopses that try to minimize different, non-L_2 approximation-error metrics, that might be better suited for various approximate query processing scenarios. For instance, to provide error guarantees in the approximation of individual data values, a wavelet synopsis could be constructed to minimize error metrics such as the *maximum absolute error* or the *maximum relative error* (with an appropriate *sanity bound* s) in the data reconstruction; [4] that is, minimize $\max_i\{\mathsf{absErr}_i\}$ or $\max_i\{\mathsf{relErr}_i\}$, where

$$\mathsf{absErr}_i = |A[i] - \hat{A}[i]| \quad \text{and} \quad \mathsf{relErr}_i = \frac{|A[i] - \hat{A}[i]|}{\max\{|A[i]|, \mathsf{s}\}}.$$

[4] The role of the sanity bound is to ensure that relative-error numbers are not unduly dominated by small data values [157, 284].

Another important class of error metrics is the *L_p-norm error*

$$||A-\hat{A}||_p \;=\; \left(\sum_i |A[i]-\hat{A}[i]|^p\right)^{\frac{1}{p}}$$

in the data reconstruction (for any $p \geq 0$), as well as the (more general) *weighted L_p-norm error*

$$||A-\hat{A}||_{p,\mathbf{w}} \;=\; \left(\sum_i w_i \cdot |A[i]-\hat{A}[i]|^p\right)^{\frac{1}{p}},$$

where a weight vector $\mathbf{w} = (w_0, w_2, \ldots, w_{M-1})$ is used to associate different "significance" to the errors for different values in the underlying data domain [112]. Such weights are an important tool for capturing the importance of individual data values, for example, based on the non-uniformities of the observed query workload (see also Section 4.3.3). Note that maximum-error metrics correspond to L_∞-norm error (i.e., $p = \infty$), whereas relative-error metrics are special cases of weighted L_p-norm error (with $w_i = 1/\max\{|A[i]|, \mathtt{s}\}$).

Wavelet thresholding algorithms for optimizing such non-L_2 error metrics can be classified into two broad categories. Most early work on the problem (e.g., [109, 110, 112]) focused solely on the case of *restricted wavelets*, where the algorithm selects values retained for the synopsis from the standard HWT coefficient values (i.e., computed by standard pairwise averaging and differencing). However, as observed by Guha and Harb [135], such a restriction makes little sense when optimizing for non-L_2 error, and can, in fact, lead to sub-optimal synopses. Their work considers *unrestricted* Haar wavelets, where the values retained in the synopsis are specifically chosen to optimize a general (weighted) L_p-norm error metric. In what follows, we survey these two related lines of research. We also discuss the concurrent efforts of Matias and Urieli [223, 224] on building workload-optimal wavelet synopses through the design of appropriate *weighted* Haar wavelet bases that allow for optimal greedy thresholding based on Parseval's theorem.

4.3.1 Restricted Haar Wavelets for Non-L_2 Error

We start with a brief description of early work on *probabilistic* thresholding schemes that implicitly try to optimize for maximum error through the minimization of appropriate probabilistic metrics. We then discuss techniques for

deterministic wavelet thresholding that select HWT values to directly optimize various (non-L_2) error metrics.

4.3.1.1 Probabilistic Wavelet Synopses

The problems and biases of conventional, greedy wavelet thresholding were first observed by Garofalakis and Gibbons [109, 110]. Their work introduces *probabilistic wavelet synopses* that rely on probabilistic thresholding algorithms based on ideas from randomized rounding [233]: The key idea is to deterministically retain the most important coefficients while randomly rounding the other coefficients either up to a larger value (called a *rounding value*) or down to zero. The probability of rounding up vs. down is selected so that the expected value of the rounded coefficient equals the original coefficient. By carefully selecting the rounding values (through appropriate optimization procedures), the thresholding algorithm ensures that (1) the expected total number of coefficients in the synopsis is B, and (2) a desired maximum-error metric (e.g., maximum relative error) in the reconstruction of the data is minimized [110].

More specifically, each non-zero coefficient c_i in the wavelet transform of the data array A is associated with a random variable C_i such that (1) C_i takes the value zero (i.e., c_i is discarded from the synopsis) with some (possibly zero) probability, and (2) $E[C_i] = c_i$. The *probabilistic wavelet synopsis* for A, comprises the values for those random variables C_i with non-zero values.[5] The general form of these random variables is determined using a randomized rounding scheme, with a *rounding value*, λ_i, for each non-zero c_i such that $C_i \in \{0, \lambda_i\}$, $0 < \frac{c_i}{\lambda_i} \leq 1$, and

$$C_i = \begin{cases} \lambda_i & \text{with probability } \frac{c_i}{\lambda_i} \\ 0 & \text{with probability } 1 - \frac{c_i}{\lambda_i} \end{cases}$$

Thus, the thresholding scheme essentially "rounds" each non-zero wavelet coefficient c_i *independently* to either λ_i or zero by flipping a biased coin with success probability $\frac{c_i}{\lambda_i}$. For this rounding process, the expected value of each rounded coefficient is $E[C_i] = \lambda_i \cdot \frac{c_i}{\lambda_i} + 0 \cdot (1 - \frac{c_i}{\lambda_i}) = c_i$ (i.e., the actual

[5] Note that the values stored in the synopsis can be different from those of the corresponding HWT coefficients. Still, we chose to classify this as a "restricted" approach since it insists on maintaining the values of individual HWT coefficients *on expectation*.

coefficient value), and its variance is simply $\mathrm{Var}(C_i) = E[C_i^2] - (E[C_i])^2 = \lambda_i^2 \cdot \frac{c_i}{\lambda_i} - c_i^2 = (\lambda_i - c_i) \cdot c_i$. (For the special case where a coefficient is deterministically retained, $\lambda_i = c_i$, and indeed $\mathrm{Var}(C_i) = 0$.) Let $\hat{A}[i]$ ($\hat{A}(l:h)$) denote the estimator for the data value $A[i]$ (resp., the range sum/count $A(l:h)$), as calculated based on the coefficient values retained in a probabilistic wavelet synopsis. By the linearity of point and range-sum reconstruction (Section 4.2) and linearity of expectation, it is not difficult to prove that these estimators are *unbiased*; that is, $E[\hat{A}[i]] = A[i]$ and $E[\hat{A}(l:h)] = A(l:h)$ [110].

The above development holds for *any* choice of rounding values λ_i, as long as $0 < \frac{c_i}{\lambda_i} \leq 1$. The choice of the λ_i's is crucial, however, because it determines the variances of probabilistic estimators as well as the expected number of coefficients retained. Indeed, the key to providing "good" error guarantees for individual data values and range sums lies in selecting the λ_i's to ensure small variances for data-value reconstruction while not exceeding the prescribed space limit for the synopsis. (Note that, essentially, the retention probabilities $\frac{c_i}{\lambda_i}$ can be seen as the amount of *fractional storage* assigned to individual coefficients c_i.) Based on these observations, [109, 110] propose novel thresholding algorithms based on dynamic-programming (DP) formulations over the Haar-tree structure that explicitly minimize appropriate probabilistic metrics (such as the maximum normalized standard error or the maximum normalized bias) in the randomized synopsis construction; these formulations are then combined with a *quantization* of the potential fractional-storage allotments to give polynomial-time combinatorial techniques [110]. More specifically, employing a quantization that allocates fractional storage to coefficients at multiples of $1/\mathtt{q}$ (where $\mathtt{q} > 1$ is an integer input parameter), the DP algorithms of [110] construct a B-coefficient probabilistic wavelet synopsis in time $O(M\mathtt{q}^2 B \log(\mathtt{q}B))$. Since this complexity can be problematic, especially for large synopsis sizes B, Deligiannakis et al. [80] propose an *approximation scheme* that, given a desired approximation factor ε, builds a B-coefficient probabilistic wavelet synopsis in $O(M\mathtt{q} \log \mathtt{q} \min\{\log M \log R/\varepsilon, B\mathtt{q}\})$ (where R is roughly proportional to the maximum absolute Haar-coefficient value in the decomposition), while guaranteeing that the quality of the final solution is within a factor of $(1+\varepsilon)$ of that obtained by the (exact) techniques of [110] for the same problem instance.

4.3.1.2 Deterministic Thresholding Techniques

The probabilistic synopses of [109, 110] try to probabilistically control data-value reconstruction error through the optimization of appropriate probabilistic measures (like normalized standard error or bias [109, 110]). Such schemes can suffer from important shortcomings, including the potential pitfalls of randomized techniques (e.g., a "bad" sequence of coin flips resulting in a poor synopsis), and the ad-hoc nature of the space-quantization requirement of [109, 110] whose impact on the quality of the final solution is not entirely clear. Furthermore, the synopsis space bound is only preserved on expectation, and the estimated variance in the space usage can be large [135].

To address these issues, Garofalakis and Kumar [111, 112] design a more direct, *deterministic* solution that explicitly minimizes the error metric of interest. Their DP-based, deterministic thresholding algorithms for building coefficient synopses provably optimize the *maximum relative error or absolute error* in the data-value reconstruction (the key metrics considered in [110]). Furthermore, their algorithmic techniques are directly applicable to a much broader class of *distributive* approximation-error metrics (which includes, for instance, weighted L_p-norm error) [112]. We now briefly overview some of the key ideas of their solution.

Given a target error metric $\mathsf{err} \in \{\mathsf{relErr}, \mathsf{absErr}\}$, the goal is to efficiently select the B Haar coefficients that minimize $\max_{i\in\{1,\ldots,M\}} \mathsf{err}_i$, where err_i is the reconstruction error for $A[i]$. The challenge here is that, since each coefficient contributes with different signs to each half of its support range, these error metrics do not have a simple monotonic structure along the Haar-tree structure. To formulate a valid DP recurrence for maximum error metrics, the key idea is to condition the optimal error value for an error subtree not only on the root node c_j of the subtree and the amount B of synopsis storage allotted, but also on *the error that "enters" that subtree* through the coefficient selections made on the path from the root to node c_j (excluding c_j itself), i.e., coefficient selections on $\texttt{path}(c_j)$. The basic observation here is that, since the depth of the Haar tree is $O(\log M)$, we can afford to tabulate all such possible selections while keeping the running-time of the algorithm in the low-polynomial range [111, 112].

More formally, let B denote the total space budget for the synopsis, and let T_j be the subtree of the Haar-tree rooted at node c_j, with $\texttt{coeff}(T_j)$

($\texttt{data}(T_j)$) denoting the set of coefficient (respectively, data) values in T_j. Finally, let $E[j,b,S]$ denote the optimal (i.e., minimum) value of the maximum error (relative or absolute) among all data values in T_j assuming a synopsis space budget of b coefficients for the T_j subtree, *and* that a subset $S \subseteq \texttt{path}(c_j)$ (of size at most $\min\{B-b, \log M+1\}$) of proper ancestors of c_j have been selected for the synopsis; that is, assuming a relative-error metric (i.e., $\mathsf{err} = \mathsf{relErr}$),

$$E[j,b,S] = \min_{S_j \subseteq \texttt{coeff}(T_j), |S_j| \leq b} \left\{ \max_{A[i] \in \texttt{data}(T_j)} \mathsf{relErr}_i \right\},$$

where

$$\mathsf{relErr}_i = \frac{|A[i] - \sum_{c_k \in \texttt{path}(A[i]) \cap (S_j \cup S)} \mathsf{sign}_{i,k} \cdot c_k|}{\max\{|A[i]|, \mathtt{s}\}}.$$

(The case for absolute error (i.e., $\mathsf{err} = \mathsf{absErr}$) is defined similarly.) The $E[j,b,S]$ entries can be efficiently computed through a DP recurrence [112] (discussed below); clearly, $E[0,B,\phi]$ gives the desired optimal error value at the root node of the Haar tree (the corresponding error-optimal wavelet synopsis can then be built by simply re-tracing the choices of the DP computation using standard techniques).

The *base case* for the DP recurrence occurs for data (i.e., leaf) nodes in the Haar tree; that is, for $c_j = A[j-M]$ with $j \geq M$ (see Figure 4.1). In this case, $E[j,b,S]$ is not defined for $b > 0$, whereas for $b = 0$ and for each subset $S \subseteq \texttt{path}(A[j-M])$ (of size $\leq \min\{B, \log M+1\}$)

$$E[j,0,S] = \frac{|A[j-M] - \sum_{c_k \in S} \mathsf{sign}_{j-M,k} \cdot c_k|}{r},$$

where $r = \max\{|A[j-M]|, \mathtt{s}\}$ for $\mathsf{err} = \mathsf{relErr}$, and $r = 1$ for $\mathsf{err} = \mathsf{absErr}$.

In the case of an *internal Haar-tree node* c_j with $j < M$, the DP algorithm has two distinct choices when computing $E[j,b,S]$, namely either drop coefficient c_j or keep it in the final synopsis. If c_j *is dropped* from the synopsis, then it is easy to see that the maximum error from c_j's two child subtrees (i.e., c_{2j} and c_{2j+1}) will be propagated upward; thus, the minimum possible maximum error $E[j,b,S]$ for T_j in this case is simply

$$E_{\text{drop}}[j,b,S] \;=\; \min_{0 \leq b' \leq b} \max\left\{ E[2j,b',S] \,,\, E[2j+1,b-b',S] \right\}. \qquad (4.4)$$

If, on the other hand, c_j *is kept* in the synopsis (assuming, of course, $b \geq 1$), the least possible error $E[j,b,S]$ for T_j is computed as

$$E_{\text{keep}}[j,b,S] \;=\; \min_{0 \leq b' \leq b-1} \max \big\{\, E[2j,b',S \cup \{c_j\}]\,, \\ E[2j+1,b-b'-1,S \cup \{c_j\}] \,\big\}. \quad (4.5)$$

(Note that the right-hand side of the recurrence is well-defined in both cases.) The final value for $E[j,b,S]$ is defined as the *minimum* of the two possible choices for coefficient c_j (Equations (4.4) and (4.5) above); that is,

$$E[j,b,S] \;=\; \min\{\, E_{\text{drop}}[j,b,S]\,,\, E_{\text{keep}}[j,b,S] \,\}.$$

The above recurrence can be translated to a DP algorithm in a straightforward manner; furthermore, both the time and (total) space complexity of the algorithm can be shown to be $O(M^2)$ [112, 133]. (Assuming that unnecessary parts of the DP matrix can be paged out, the working-space requirement of the algorithm is only $O(M \min\{B, \log M\})$ [112].) Guha [134] proposes a space-efficient implementation of this dynamic program that requires only $O(B \log M)$ space while slightly increasing the time complexity to $O(M^2 \log B)$. The key technical idea is to avoid tabulating the solutions to all Haar-subtree problems and repeatedly solve the same subproblems in a top-down fashion in order to determine the optimal synopsis (essentially, trading time for space).

Interestingly, the above algorithmic solutions have general applicability for a natural, wide class of *distributive error metrics* [112]. Briefly, an error metric $f()$ is distributive if and only if for any collection of disjoint domain ranges $R_1, \ldots, R_k$,

$$f(\cup_{i=1}^{k} R_i) \;=\; g(f(R_1), \ldots, f(R_k)),$$

where $g()$ is some combining function for the errors over individual regions. In addition to maximum absolute/relative error metrics, the class of distributive error functions also naturally encompasses several other important error metrics, including (the p^{th} power of) L_p-norm and weighted L_p-norm error [112].

Heuristics and Related Approaches. The quadratic $O(M^2)$ time and space complexity of the optimal DP thresholding algorithm of [111, 112] can be

problematic for large data sets. To overcome such potential limitations, Karras and Mamoulis [202] propose a greedy heuristic solution for maximum-error thresholding that guarantees near-linear complexity. We now briefly review the main ideas of their thresholding scheme.

Let $\mathsf{serr}_i = \hat{A}[i] - A[i]$ denote the *signed* accumulated error in the synopsis for a data node $A[i]$ after some coefficient deletions. For a coefficient node c_j not yet discarded from the synopsis, the *maximum potential absolute error* MA_k that c_k can effect on the synopsis is

$$\mathsf{MA}_k = \max_{A[i] \in \mathtt{leaves}(c_k)} \{|\mathsf{serr}_i - \mathsf{sign}_{i,k} \cdot c_k|\}.$$

The greedy algorithm for maximum absolute error discards, at each step, the coefficient in the running synopsis with the maximum value of MA_k [202]. The key observation here is that the maintenance of MA_k for any coefficient c_k can be performed efficiently, without accessing all data nodes in $\mathtt{leaves}(c_k)$. More specifically, since the removal of a coefficient c_k impacts equally the signed errors in its left or right subtrees, the *maximum* and *minimum* signed errors in the left (right) subtree of c_k are decreased (increased) by exactly c_k. And, clearly, the maximum absolute error incurred by the removal of c_k must occur at one of these four positions of error extrema: Letting max_k^l, min_k^l (max_k^r, min_k^r) denote the maximum and minimum signed errors in the left (right) subtree of c_k, we can compute MA_k as follows:

$$\mathsf{MA}_k \;=\; \max\{\; |max_k^l - c_k|,\; |min_k^l - c_k|,\; |max_k^r + c_k|,\; |min_k^r + c_k|\;\}.$$

Thus, by maintaining these four quantities max_k^l, min_k^l, max_k^r, and min_k^r at each coefficient node, the greedy algorithm can compute MA_k in constant time using the above formula. To determine the next coefficient to discard, all coefficients are placed in a min-heap structure H based on their MA values, and the coefficient c_k at the root of the heap is removed. After the removal of c_k the min/max error information for all descendant and ancestor coefficients of c_k in the Haar tree must be updated. The total number of such updates can be shown to be $O(M \log M)$ and each such update can require an $O(\log M)$ re-positioning in the heap — this gives an overall time complexity of $O(M \log^2 M)$ while the space complexity is $O(M)$ (constant information for each coefficient node) [202].

For maximum *relative* error minimization, the greedy heuristic of [202] discards, at each step, the coefficient with the minimum *maximum potential*

relative error:

$$\mathsf{MR}_k = \max_{A[i] \in \texttt{leaves}(c_k)} \left\{ \frac{|\mathsf{serr}_i - \mathsf{sign}_{i,k} \cdot c_k|}{\max\{|A[i]|, \mathtt{s}\}} \right\}.$$

In this case, however, the effect of removing a coefficient c_k is different for each data value in $\texttt{leaves}(c_k)$; thus, the algorithm maintains the values affected for each c_k in an augmented max-heap H_k based on their potential relative error values, so that the value causing the maximum potential error can be accessed in constant time. These per-coefficient heaps raise the time/space costs by a factor $O(\log M)$, raising the overall time and space complexity of the greedy algorithm to $O(M \log^3 M)$ and $O(M \log M)$, respectively [202].

Also motivated by the quadratic time complexity of the optimal DP in [111, 112], Muthukrishnan [237] considers the special case of weighted L_2-error metrics where the weight vector $\mathbf{w} = (w_0, w_2, \ldots, w_{M-1})$ is *k-flat*, that is it comprises $k \ll M$ piecewise constant partitions (or, equivalently, $\mathbf{w}$ is represented as a k-bucket histogram over $[0, M-1]$). The key technical observation here is that the optimization problem for each piecewise constant (in terms of weight) portion of the Haar tree can be solved greedily through a local application of Parseval's theorem. This implies that the expensive optimal dynamic program needs to be executed only over the portion of the Haar tree up to internal nodes that correspond to dyadic ranges contained within piecewise constant portions of $\mathbf{w}$; essentially, such piecewise constant tree nodes are treated as leaves in the DP. Thus, for such k-flat weight vectors, the weighted L_2 optimization problem can be solved in time $O(MB^2 k \log M)$ [237].

4.3.2 Unrestricted Haar Wavelets for Non-L_2 Error

In its most general form, wavelet synopsis construction is a *sparse wavelet representation problem* where, given a wavelet basis $\{\phi_i\}_{i=0}^{M-1}$ for $\mathbb{R}^M$ and an input data vector (or, signal) $A \in \mathbb{R}^M$, the goal is to construct an approximate representation $\hat{A}$ as a linear combination of at most B basis vectors so as to minimize some normed distance between A and $\hat{A}$.[6] The sparse B-term representation $\hat{A}$ belongs to the non-linear space $\{\sum_{i=0}^{M-1} z_i \phi_i : z_i \in \mathbb{R}, ||Z||_0 \leq B\}$, where the L_0 norm $||Z||_0$ denotes the number of non-zero coefficients in the vector $Z = (z_0, z_1, \ldots, z_{M-1})$. (We use $\hat{A}_Z$ to denote the approximate

[6] This is an instance of the general *non-linear approximation problem* in approximation theory [85].

representation for a specific coefficient vector Z.) For the case of L_2 error, by Parseval's theorem, the L_2 norm of $A - \hat{A}$ is preserved in the wavelet space; thus, generalizing Equation (4.3), we have

$$||A - \hat{A}_Z||_2^2 \;=\; \sum_i \left(A[i] - \sum_j z_j \phi_i[j] \right)^2 \;=\; \sum_i (\langle A, \phi_i \rangle - z_i)^2.$$

It is clear that the optimal solution under the L_2 error measure is to retain the largest B inner products $\langle A, \phi_i \rangle$ which are exactly the largest (normalized) coefficients c_i^* in the HWT expansion of A. Thus, the greedy thresholding approach of Section 4.2.2 is optimal for L_2-error minimization even in this generalized setting. For other error norms, however, restricting the z_i's to the set of computed HWT coefficients of A can result in suboptimal solutions. As a simple example, consider the data vector $A = [1, 4, 5, 6]$ whose HWT gives $[4, -1.5, -1.5, -0.5]$. Assuming $B = 1$, the optimal solution for maximum absolute (or, L_∞) error is $Z^* = [3.5, 0, 0, 0]$, whereas the best solution restricted to the computed HWT coefficients is $[4, 0, 0, 0]$; furthermore, the example can be generalized to any B and larger error gaps by simply scaling and repeating the values in A with alternating signs [135].

A first step in solving the generalized (unrestricted) sparse Haar wavelet representation problem is demonstrating the existence of a bounded set R from which coefficient values in Z can be chosen while ensuring a solution that is close to the optimal unrestricted solution (where z_i's range over all reals). Guha and Harb [135] prove that, for L_p-error minimization, the maximum (un-normalized) coefficient value in the optimal solution Z^* satisfies $\max_i\{|z_i^*|\} \leq 2M^{1/p}\alpha_{max}$, where $\alpha_{max} = \max_i\{|A[i]|\}$ (i.e., the maximum absolute value in the input data).[7] Furthermore, they demonstrate that, by rounding the coefficient values in the optimal solution Z^* to the nearest multiple of some $\delta > 0$ (obtaining a "rounded" solution $\hat{Z}_\delta$) introduces bounded additive error in the target L_p norm; more specifically,

$$||A - \hat{A}_{\hat{Z}_\delta}||_p \leq ||A - \hat{A}_{Z^*}||_p \;+\; \delta M^{1/p} \min\{B, \log M\}. \tag{4.6}$$

Thus, the above additive error over the optimal solution can be guaranteed

[7] Following [135], we focus on the *un-normalized* Haar basis in order to simplify the statement of the results.

while restricting the search for coefficient values over a set of size [135]:

$$|R| \;=\; 2 \cdot \frac{\max_i\{|z_i^*|\}}{\delta} \;\leq\; \frac{4M^{1/p}\alpha_{max}}{\delta}. \tag{4.7}$$

The search for the optimal sparse wavelet representation is conducted using a dynamic program that is very similar to the DP formulation for the restricted case [111, 112], with two key differences: First, the search at a given Haar-tree node j also ranges over all possible value selections at node j (i.e., over all values in R); and, second, the optimal error at a Haar-tree node j is conditioned on the on the sum of values of ancestors of j (which can also be restricted to R) [135]. Letting $E[j,b,V]$ denote the minimum L_p error for subtree T_j using exactly b coefficients and assuming that the ancestors of node j add up (with the proper signs) to value V in the final solution. Then, similar to the restricted case, we can compute $E[j,b,V] = \min\{E_{\text{drop}}[j,b,V], E_{\text{keep}}[j,b,V]\}$, where

$$\begin{aligned} E_{\text{drop}}[j,b,V] &= \min_{b' \in [0,b]} \left\{ E[2j,b',V] \;,\; E[2j+1,b-b',V] \right\} \\ E_{\text{keep}}[j,b,V] &= \min_{b' \in [0,b-1], r \in R} \left\{ E[2j,b',V+r] \;,\; E[2j+1,b-b'-1,V-r] \right\} \end{aligned}$$

represent the choice of dropping or keeping the basis function at node j. Note that, in the case that j is kept, the search also ranges over all possible coefficient values in R. The running time of this DP is $O(|R|^2 MB)$, while its working-space requirement is $O(|R|B\log(M/B))$. (In the case of maximum (L_∞) error, binary search over R can be used, reducing the running time complexity to $O(|R|^2 M \log^2 B)$.) Combined with Equations (4.6)-(4.7), this implies that the unrestricted L_p-optimal Haar wavelet synopsis can be approximated to within an additive error of $\varepsilon\alpha_{max}$ (where $\alpha_{max} = \max_i\{|A[i]|\}$) in time $O(\frac{1}{\varepsilon^2}M^{1+4/p}B(\min\{B,\log M\})^2)$ and space $O(\frac{1}{\varepsilon}M^{2/p}B\min\{B,\log M\}\log(M/B))$ [135].

The approach also extends to weighted L_p error and relative error norms; however, the time and space complexities become dependent on the value of the minimum weight coefficient $\min_i\{w_i\}$ and the minimum normalizing factor $\max\{\min_i\{A[i]\}, \mathtt{s}\}$, respectively [135]. It is also interesting to note that removing the restriction of choosing Haar coefficient values enables the streaming computation of the (near-optimal) sparse wavelet representation in one pass over the entries of A — see Section 4.6.1.

In more recent theoretical work, Guha and Harb [136, 161] demonstrate how one can extend the dynamic-programming ideas described above to obtain a true $(1+\varepsilon)$ approximation scheme for unrestricted sparse Haar wavelet representations under L_p error that runs in time $O(\frac{1}{\varepsilon^2}M^{1+2/p}B\log^3 M)$ and uses $O(\frac{1}{\varepsilon}M^{1/p}B\log^3 M)$ space. (As earlier, the running time for the case of maximum error ($p=\infty$) can be reduced by a factor of $B/\log^2 B$ by using binary search.) Their work also examines the gap between the restricted and unrestricted versions of the wavelet approximation problem, proving that a restricted solution which greedily retains at most B wavelet coefficients is a $O(M^{1/p}\log M)$ approximation to the optimal unrestricted solution for all L_p norms; in fact, this result holds for *any* wavelet family of compact support [136].

Optimizing Maximum-Error through the Dual Problem. For the special case of (weighted) maximum (L_∞) error metrics, Karras et al. [204] propose an alternative approach to constructing near-optimal unrestricted Haar synopses based on the *dual* problem of finding *error-bounded* Haar summaries: Given an upper bound $\mathscr{E}$ on the error, find a space-optimal synopsis that guarantees maximum error $\leq \mathscr{E}$.[8] A key observation here is that there exists a natural DP formulation for this dual problem, that avoids the two-dimensional tabulation over all possible incoming values v and space allotments b for every Haar tree node — this essentially eliminates the dependence of running time and space on the synopsis size B, which can result in significant efficiency gains [204].

More specifically, let $S[j,v]$ denote the minimum number of non-zero coefficients that need to be retained in subtree T_j assuming an incoming value V (from the ancestor nodes of node j), in order to ensure a maximum error bound of $\mathscr{E}$; then, $S[j,V]$ can be recursively computed through the following DP recurrence:

$$S[j,V] \;=\; \min_{r\in R_{j,V}} \{S[2j,V+r]+S[2j+1,V-r]+(r\neq 0)\},$$

where $(r\neq 0)$ denotes the boolean indicator variable that is 1 iff $r\neq 0$, and $R_{j,V}$ is the set of possible values for node j given an incoming value V. By

[8] Some preliminary ideas on using the dual formulation for the restricted problem are also sketched in the short article by Muthukrishnan [237].

quantizing the potential values into multiples of a small resolution step δ (as earlier), the size of the set of possible values at each node can be shown to be $|R_{j,V}| = O(\frac{\mathscr{E}}{\delta})$; this, in turn, implies a DP algorithm for constructing the error-bounded Haar synopsis in $O((\frac{\mathscr{E}}{\delta})^2 M \log M)$ time using $O(\frac{\mathscr{E}}{\delta} \log M)$ space [204]. This error-bounded algorithm can be used as a subroutine in a solution to the original *space-bounded* maximum-error synopsis construction problem that uses *binary search* over a range defined by lower/upper bounds on the optimal maximum error for a B-coefficient synopsis until the "right" error value is discovered. Compared to the time requirements of the error-bounded solution, this binary search increases the running time of the space-bounded procedure by a factor of $O(\log \mathscr{E}^*)$, where $\mathscr{E}^*$ denotes the optimal (minimum) maximum-error value for a B-coefficient synopsis [204].

Pang et al. [248] also address the (dual) error-bounded version of the unrestricted maximum-error wavelet representation problem, and propose fast, linear-time (i.e., $O(M)$) approximation algorithms that have a $\log M$ approximation guarantee (that is, the space for the resulting synopsis is at most $\log M$ times the size of the optimal synopsis for the given maximum-error bound $\mathscr{E}$). Their key technical observation is that unrestricted coefficient values in the synopsis can be viewed as simple *shift transformations* on the ranges of values in Haar subtrees. The idea is then to employ such shift transformations in a bottom-up manner over the Haar-tree structure in a manner that ensures that the error ranges in the underlying subtrees satisfy the given error bound $\mathscr{E}$. Their most sophisticated strategy (termed *S-Shift*) computes only ranges of shift values in the bottom-up pass, and then uses a top-down pass to perform "late-binding" of shift coefficients in a smart way that tries to further reduce the size of the synopsis [248].

4.3.3 Weighted Haar Wavelets for Workload-based Error

Weighted error metrics, such as the weighted L_2 error $\sqrt{\sum_i w_i (A[i] - \hat{A}[i])^2}$, generalize standard error norms making it possible to model the distribution of the query workload. The idea here is that the weight vector $\mathbf{w} = (w_0, w_1, \ldots, w_{M-1})$ with $w_i \in [0,1]$ and $\sum_i w_i = 1$ captures the distribution of individual point queries, with w_i being the probability of occurrence for a query on $A[i]$. Thus, the w_i weights represent the "significance" of individual data points in the query workload, and minimizing the weighted error norm

is essentially equivalent to minimizing the expected query-answering error.

Matias and Urieli [223, 224] have proposed a methodology for designing wavelet synopses that specifically targets such weighted (workload-based) error metrics. Their solution relies on exploiting the weight vector $\mathbf{w}$ to define appropriate *weighted* inner-product and norm operators, and design a corresponding orthonormal *weighted* Haar wavelet basis for $\mathbb{R}^M$; then, by Parseval's theorem, the largest coefficients in this weighted basis are guaranteed to minimize the corresponding weighted error metric.

Consider the weighted L_2 error norm $||A-\hat{A}||_{2,\mathbf{w}} = \sqrt{\sum_i w_i (A[i]-\hat{A}[i])^2}$. Based on the workload vector $\mathbf{w}$, define a weighted inner product

$$\langle a,b\rangle_{\mathbf{w}} \;=\; M \sum_{i=0}^{M-1} w_i \cdot a[i]b[i].$$

Note that this is a generalization of the standard inner product (which assumes $w_i = 1/M$ for all i), that also satisfies all the mathematical conditions for an inner-product operator [224]. Furthermore, it is easy to verify that the corresponding weighted-inner-product norm is exactly

$$\sqrt{\langle a,a\rangle_{\mathbf{w}}} \;=\; \sqrt{M}\cdot||a||_{2,\mathbf{w}},$$

that is, directly proportional to the target weighted L_2 norm. Thus, minimizing the weighted-inner-product norm for the error vector $A-\hat{A}$ is equivalent to minimizing the weighted L_2 error. The next step is to define a *weighted* Haar wavelet basis that is orthonormal with respect to the weighted-inner-product operator $\langle\cdot\rangle_{\mathbf{w}}$. This is accomplished by retaining the support and general shape of the original Haar basis functions (Equation (4.2)) and, for each basis function ϕ_i, scaling its positive (negative) part by some factor x_i (resp., y_i). Thus, in contrast to the original Haar basis, where all coefficients at resolution level l are scaled by $\sqrt{2^l/M}$ (Equation 4.2)), the weighted Haar basis functions $\phi_{i,\mathbf{w}}$ are of the form illustrated in Figure 4.2, where the positive and negative scaling factors x_i and y_i are not necessarily equal.

The scales x_i and y_i for each basis function $\phi_{i,\mathbf{w}}$ are chosen to ensure orthonormality of the resulting weighted basis with respect to the corresponding weighted-inner-product. Letting l_i (r_i) denote the sum of workload weights w_i under the positive (resp., negative) half of $\phi_{i,\mathbf{w}}$ (Figure 4.2), it can be shown

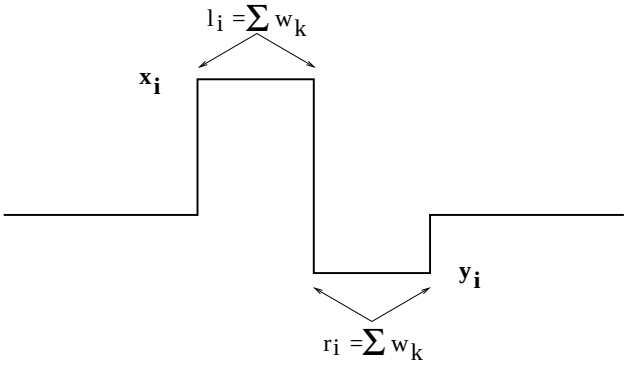


Fig. 4.2 Weighted Haar basis function $\phi_{i,\mathbf{w}}$.

that choosing

$$x_i = \sqrt{\frac{r_i}{M(l_i r_i + l_i^2)}} \quad \text{and} \quad y_i = \sqrt{\frac{l_i}{M(l_i r_i + r_i^2)}},$$

guarantees that the resulting weighted Haar wavelet basis is orthonormal; that is, $\langle \phi_{i,\mathbf{w}}, \phi_{j,\mathbf{w}} \rangle_{\mathbf{w}} = 1$ if $i = j$ and 0 otherwise [224]. (The root basis vector (overall average) remains $\phi_{0,\mathbf{w}} = \frac{1}{\sqrt{M}}\mathbf{1}^M$.) Note that the weighted Haar basis generalizes the standard Haar basis: for a uniform workload (i.e., $w_i = 1/M$ for all i), both scaling factors for a basis function at level l are exactly $\sqrt{2^l/M}$, resulting in the standard Haar basis functions (Equation (4.2)).

Given a workload vector $\mathbf{w}$, computing the weighted Haar basis can be done bottom-up (in a manner similar to the HWT process) in $O(M)$ time. Similarly, computing the weighted Haar wavelet transform for an input data vector $A \in \mathbb{R}^M$ can also be done in $O(M)$ time — the key difference from standard HWT is that the basic steps of the decomposition now involve *weighted* averaging and differencing [224]. By the orthonormality of the weighted Haar basis and Parseval's theorem, it immediately follows that retaining the B largest weighted HWT coefficients in the synopsis of A is optimal with respect to minimizing the weighted L_2-error norm $||A - \hat{A}||_{2,\mathbf{w}}$. Furthermore, the methodology can be easily extended to handle other workload-based sum-squared-error metrics. For instance, consider the problem of minimizing the weighted sum-squared *relative* error norm $\sum_{i=0}^{M-1} w_i \mathsf{relErr}_i^2 = \sum_{i=0}^{M-1} w_i \frac{(A[i]-\hat{A}[i])^2}{\max\{A[i]^2, \mathbf{s}^2\}}$. Then, by simply defining a modified

weight vector $\mathbf{w}' = (w'_0, w'_1, \ldots, w'_{M-1})$, where $w'_i = \frac{w_i}{\max\{A[i]^2, s^2\}}$, the problem is transformed to weighted L_2-error minimization (using workload $\mathbf{w}'$) and the foregoing approach is immediately applicable [224].

4.4 Multi-Dimensional Wavelets

The foregoing discussion has focused on the case of Haar wavelets for one-dimensional data arrays (i.e., for a single data attribute). As in the case of histograms, effective data and query approximations become much more challenging for *multi-dimensional* data arrays capturing a joint data distribution over a domain $\mathscr{U} = [0, M_1 - 1] \times \cdots \times [0, M_d - 1]$ of some dimensionality $d > 1$. The Haar wavelet decomposition can be extended to such multi-dimensional data arrays using two distinct methods, namely the *standard* and *nonstandard* Haar decomposition [276]. Each of these transforms results from a natural generalization of the one-dimensional decomposition process described in Section 4.2, and both have been used in a wide variety of applications (including approximate query answering over multi-dimensional data sets [41, 42, 284]).

- *The standard decomposition* first fixes an ordering for the data dimensions (say, $1, 2, \ldots, d$) and then proceeds to apply the complete one-dimensional wavelet transform for each one-dimensional "row" of array cells along dimension k, for all $k = 1, \ldots, d$.
- *The nonstandard decomposition* alternates between dimensions during successive steps of pairwise averaging and differencing. Given an ordering for the data dimensions $(1, 2, \ldots, d)$, we perform *one step of pairwise averaging and differencing* for each one-dimensional row of array cells along dimension k, for each $k = 1, \ldots, d$. We then recurse on the quadrant containing the averages across all d dimensions. See Section 4.4.1.2 for details and an example.

As in the one-dimensional case, the (standard or nonstandard) HWT of a d-dimensional data array A results in a d-dimensional wavelet-coefficient array W_A with the same dimension ranges and number of entries. Each data cell in A can be accurately reconstructed by adding up the contributions (with the appropriate signs) of those coefficients whose support regions include the

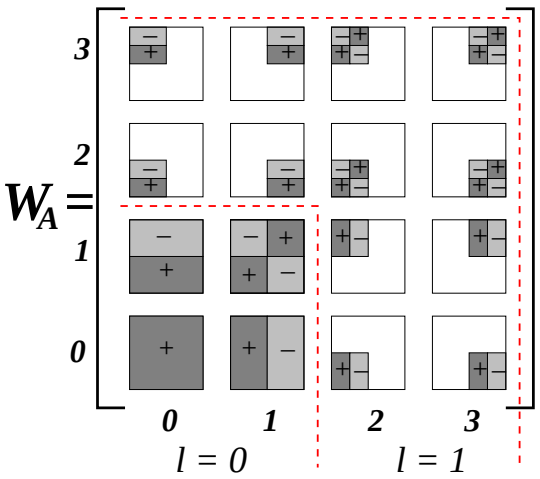


Fig. 4.3 Support regions and signs for the sixteen nonstandard two-dimensional Haar basis functions. The coefficient magnitudes are multiplied by $+1$ (-1) where a sign of $+$ (respectively, $-$) appears, and 0 in blank areas.

cell. (To simplify the exposition to the basic ideas of multi-dimensional wavelets, in this section, we assume all dimensions of the input array to be of equal size $M_i = m$; thus, the total number of data cells is $M = \prod_{i=1}^{d} M_i = m^d$.) Consider a d-dimensional wavelet coefficient c in the (standard or nonstandard) wavelet-coefficient array W_A. c contributes to the reconstruction of a d-dimensional rectangular region of cells in the original data array A (i.e., c's *support region*). Further, the sign of c's contribution ($+c$ or $-c$) can vary along the quadrants of c's support region in A.

As an example, Figure 4.3 depicts the support regions and signs of the sixteen nonstandard, two-dimensional Haar coefficients in the corresponding locations of a 4×4 wavelet-coefficient array W_A. The blank areas for each coefficient correspond to regions of A whose reconstruction is independent of the coefficient, i.e., the coefficient's contribution is 0. Thus, $W_A[0,0]$ is the overall average that contributes positively (i.e.,"$+W_A[0,0]$") to the reconstruction of all values in A, whereas $W_A[3,3]$ is a detail coefficient that contributes (with the signs shown in Figure 4.3) only to values in A's upper right quadrant. Each data cell in A can be accurately reconstructed by adding up the contributions (with the appropriate signs) of those coefficients whose support regions include the cell.

Haar-tree structures for d-dimensional Haar coefficients can also be naturally defined based on the specifics of the multi-dimensional decomposition process and the natural inclusion relationships that arise between coefficient support regions. In the case of nonstandard decomposition, the Haar-tree is essentially a *d-dimensional quadtree*, where each internal node t corresponds to a *set* of (at most) $2^d - 1$ Haar coefficients, and has 2^d children corresponding to the quadrants of the (common) support region of all coefficients in t. A key difference is that, in a d-dimensional Haar tree, each node (except for the root, i.e., the overall average) actually corresponds to a *set* of $2^d - 1$ wavelet coefficients that have the same support region but different quadrant signs and magnitudes for their contribution.[9] (Note that the sign of each coefficient's contribution to the leaf (data) values residing at each of its children in the tree is determined by the coefficient's quadrant sign information.) As an example, Figure 4.4 depicts the Haar-tree structure for the two-dimensional 4×4 Haar coefficient array in Figure 4.1(b). Thus, the (single) child t of the root node contains the coefficients $W_A[0,1], W_A[1,0]$, and $W_A[1,1]$, and has four children corresponding to the four 2×2 quadrants of the array; the child corresponding to the lower-left quadrant contains the coefficients $W_A[0,2], W_A[2,0]$, and $W_A[2,2]$, and all coefficients in t contribute with a "+" sign to all values in this quadrant. The formula for data-value reconstruction (Equation (4.1)) naturally extends to nonstandard multi-dimensional Haar wavelets using this quadtree structure. Once again, the reconstruction of $A[i]$ depends only on the *coefficient sets* for all Haar-tree nodes in $\texttt{path}(A[i_1, \ldots, i_d])$, where the sign of the contribution for each coefficient c in node $t \in \texttt{path}(A[i_1, \ldots, i_d])$ is determined by the quadrant sign information for c. Thus, each data value can be reconstructed as a simple linear combination of $(2^d - 1)\log m + 1$ nonstandard HWT coefficients.

In the case of standard Haar wavelets, each dimension is decomposed independently, giving rise to d distinct one-dimensional Haar trees for each individual dimension (Figure 4.5). Each standard coefficient in the transformed array has d indices (one for each dimension), where each index identifies a position in the Haar tree for the corresponding dimension; furthermore, the

[9] The number of children (coefficients) for an internal Haar-tree node can actually be less than 2^d (respectively, $2^d - 1$) when the sizes of the data dimensions are not all equal. In these situations, the exponent for 2 is determined by the number of dimensions that are *"active"* at the current level of the decomposition (i.e., those dimensions that are still being recursively split by averaging/differencing).

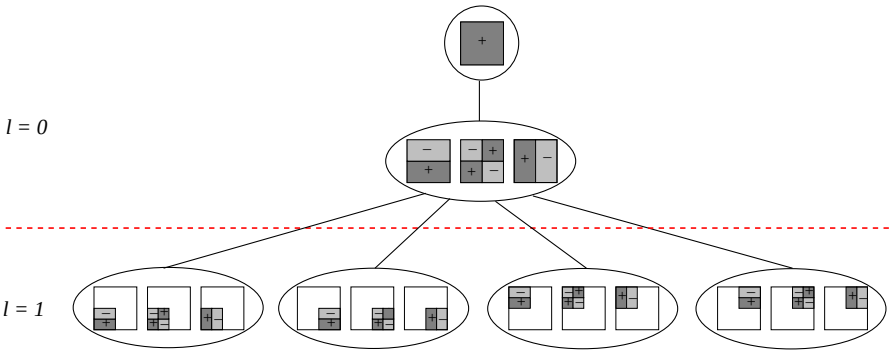


Fig. 4.4 Haar-tree structure for the sixteen nonstandard two-dimensional Haar coefficients for a 4×4 data array (data values omitted for clarity).

support of the coefficient is the d-dimensional rectangle whose sides are defined by the cross-product of the corresponding dyadic intervals along each of the d dimensions. (This is in contrast to the nonstandard Haar basis where support regions are always cubic [276].) The coefficients required for reconstructing data values in the original array are also determined through the d per-dimension Haar trees. For instance, a single data value $A[i_1, \ldots, i_d]$ corresponds to a root-to-leaf path in each of the d per-dimension Haar trees, or, equivalently, a set of one-dimensional coefficient indices — the cross product across these d index sets defines the standard multi-dimensional HWT coefficient indexes that are used to reconstruct $A[i_1, \ldots, i_d]$. Thus, reconstructing an original data value requires $(\log m + 1)^d$ standard HWT coefficients (the signs of each coefficient are determined based on the corresponding one-dimensional signs). As an example, Figure 4.5 depicts the standard multi-dimensional Haar-tree structures for a 4×4 data array, and an example data-cell reconstruction.

4.4.1 Multi-Dimensional Wavelet Decomposition Algorithms

The complexity in designing efficient Haar-wavelet decomposition algorithms for a multi-attribute relational table R comes from the fact that such algorithms cannot afford to build the multi-dimensional joint-frequency array A_R (i.e., the *datacube* representation) of R. The problem here is that the number of cells M in this datacube array A_R grows exponentially in the di-

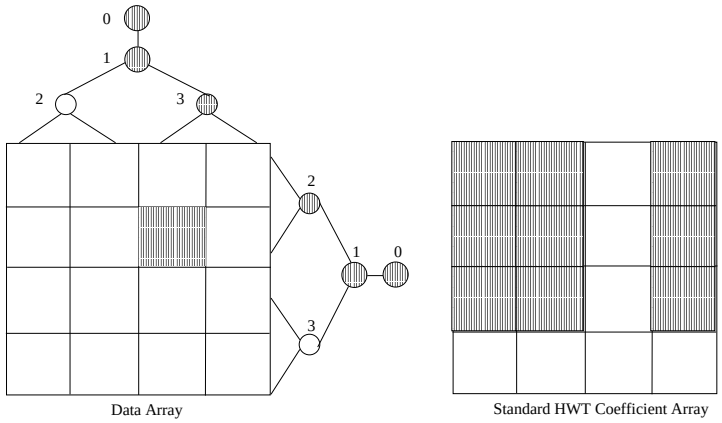


Fig. 4.5 Haar-tree structures for the standard HWT of a 4×4 data array A. The shading indicates the nine standard HWT coefficients used in reconstructing data cell $A[2,1]$.

mensionality d, and is typically orders of magnitude larger than the number of tuples in R. Thus, computation- and I/O-efficient decomposition algorithms for multi-dimensional data have to work off the *relational* "set-of-tuples" representation of R. Such efficient decomposition algorithms have been proposed for both forms of the multi-dimensional HWT.

4.4.1.1 Standard HWT Decomposition

Recall that, in the standard decomposition, the complete one-dimensional HWT is applied to each one-dimensional "row" of cells along dimension k, for all $k = 1, \ldots, d$. For instance, in the two-dimensional case, the one-dimensional HWT is first applied to each row of the data; then, we treat the transformed rows as data and apply another pass of the one-dimensional HWT to each column.

Vitter and Wang [284] propose I/O-efficient algorithms for the standard multi-dimensional HWT of a (ROLAP) relation R, which is assumed to be in *dimension order* $\langle d, d-1, \ldots, 1 \rangle$; that is, the indices of the tuple entries change most rapidly along the rightmost dimension 1, next most rapidly along dimension 2, and so on. In the multi-dimensional data space, the entries for which the values in the k-prefix of the dimensions $d, \ldots, d-k+1$ are fixed form a $(d-k)$-dimensional *hyperplane* denoted as $\langle d-k, \ldots, 1 \rangle$.

The key idea in the decomposition algorithms of [284] is to break the standard HWT computation into hyperplanes (in the chosen dimension order), such that the complete decomposition for each hyperplane can be carried out entirely in-memory. Assuming a memory of size S and a disk-block size b, their first algorithm partitions the dimensions into *groups* (i.e., hyperplanes $\langle i+j,\ldots,i+1,i\rangle$ such that the product of the group's dimension sizes satisfies $M_i \times \cdots \times M_{i+j} \leq S-2b$ (reserving two block-sized buffers for I/O). Assuming g such groups, the algorithm performs g passes over the data, processing the groups in reverse order (i.e., in increasing order of the start index i), one per pass. In the $(g-m+1)$-st pass, each hyperplane in the m^{th} dimension group is read in, one by one, and processed (performing an in-memory standard HWT); then, the results are written out to disk for the next pass. After each pass, the output produced on disk needs to be *transposed* in order to regroup the cells according to the dimension order required by the next pass; this transposition step can be performed in a logarithmic number of distribution passes over the data.

The second algorithm of [284], exploits intelligent buffering and knowledge of the dimension sizes to avoid explicit transposition steps between passes. The key idea is to employ a double output buffer for each cell of the hyperplane to appropriately re-order the output of each pass; in this manner, the transposition is essentially achieved "for free" as a result of the buffering mechanism. (Of course, the size of the hyperplanes for each pass must be smaller to account for all the output buffers needed — more specifically, the product of the dimension sizes in a hyperplane now has to be $\leq S/(2b+1)$.) Letting $M = \prod_i M_i$ denote the number of cells in the data array, the I/O complexity of both algorithms in [284] can be shown to be $O(\frac{|R|}{S}\log_{S/b}\frac{M}{b})$ — that is, they both require $O(\log_{S/b}\frac{M}{b})$ passes over the tuples in R [284].

4.4.1.2 Nonstandard HWT Decomposition

Abstractly, the nonstandard Haar decomposition alternates between dimensions during successive steps of pairwise averaging and differencing: given an ordering for the data dimensions $(1,2,\ldots,d)$, it performs *one step of pairwise averaging and differencing* for each one-dimensional row of array cells along dimension k, for each $k = 1,\ldots,d$. (The results of earlier averaging and differencing steps are treated as data values for larger values of k.) This

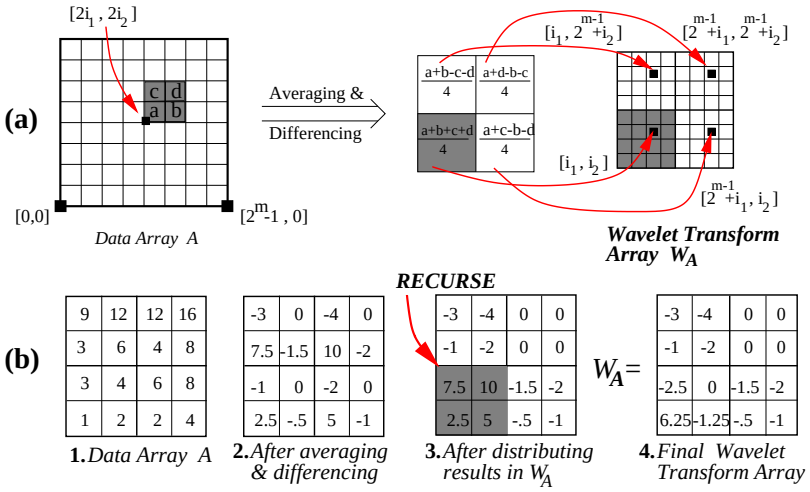


Fig. 4.6 Non-standard decomposition in two dimensions. (a) Computing pairwise averages and differences and distributing them in the wavelet transform array. (b) Example decomposition of a 4×4 array.

process is then repeated recursively only on the quadrant containing averages across all dimensions. One way of conceptualizing this procedure is to think of a $2 \times 2 \times \cdots \times 2 (= 2^d)$ hyper-box being shifted across the data array, performing pairwise averaging and differencing, distributing the results to the appropriate locations of the HWT array W_A (with the averages for each box going to the "lower-left" quadrant of W_A) and, finally, recursing the computation on the lower-left quadrant of W_A. This procedure is demonstrated pictorially for a (two-dimensional) $2^j \times 2^j$ data array A in Figure 4.6(a). More specifically, Figure 4.6(a) shows the pairwise averaging and differencing step for one positioning of the 2×2 box with its "root" (i.e., lower-left corner) located at the coordinates $[2i_1, 2i_2]$ of A followed by the distribution of the results in the wavelet transform array. This step is repeated for every possible combination of i_j's, $i_j \in \{0, \ldots, 2^{j-1} - 1\}$. Finally , the process is recursed only on the lower-left quadrant of W_A (containing the averages collected from all boxes).

Chakrabarti et al. [42] present an efficient nonstandard decomposition algorithm that employs a "chunk-based" organization of R, where the joint-frequency array A_R is (conceptually) split into d-dimensional *chunks*, and tuples of R belonging to the same chunk are stored contiguously on disk. (If R is

not chunked, then an extra sorting step is required.) Their algorithm assumes that each chunk can fit in memory, and uses the nonstandard methodology described in Figure 4.6 to compute and distribute coefficients in the HWT array in a recursive, "depth-first" fashion. The key observation here is that the decomposition of the d-dimensional array A_R can be computed by recursively computing the full decomposition for each of the 2^d quadrants of A_R and then performing pairwise averaging and differencing on the computed 2^d quadrant averages. Thus, the entire computation for decomposing a chunk can be performed when the chunk is loaded from disk for the first time (hence no chunk is read twice). Lower resolution coefficients are computed by first accumulating, in main memory, averages from the 2^d quadrants (generated from the previous level of resolution) followed by pairwise averaging and differencing, thus requiring no extra I/O. The end result is a fairly simple recursive algorithm for efficiently computing the nonstandard HWT [42]. Assuming a relation R already stored in chunks, the algorithm of [42] only needs one pass over the tuples of R; if R is not already chunked, then an extra sorting step is needed, thus resulting in an I/O complexity that matches that of the standard HWT algorithms in [284].

4.4.2 Multi-Dimensional HWT Coefficient Thresholding

L_2 Error Thresholding. Modulo a simple normalization step, both the standard and nonstandard multi-dimensional HWT bases retain the orthonormality property. Thus, as in the one-dimensional case, Parseval's theorem guarantees that retaining the *B largest multi-dimensional HWT coefficients in absolute normalized value* is the optimal strategy for minimizing the overall L_2 error in the data approximation [222, 276].

A potential issue with the multi-dimensional HWT is that, due to the repeated averaging and differencing of neighboring cells, the density of the data (i.e., number of non-zero cells) continuously increases (by a factor as high as $O(\log M)$) during each pass of the decomposition process. Thus, the decomposition algorithms have to process more and more entries during successive passes, even though a large portion of the entries can be very small in magnitude. To avoid this problem, Vitter and Wang [284] propose an *adaptive thresholding* scheme that discards intermediate coefficient values in each pass via an on-line learning process. (A similar technique is also used in [42].) The

key idea is to dynamically maintain a *cutoff value* based on on-line statistics on the distribution of intermediate coefficient values maintained during each pass. Coefficients smaller than the cutoff are discarded on the fly. The cutoff value is adjusted periodically as the statistics on the observed coefficient values shifts. Despite its heuristic nature, this adaptive technique works quite well in practice [42, 284].

Non-L_2 Error Thresholding. There are very few known results on the problem of effectively thresholding multi-dimensional Haar wavelets for non-L_2 error metrics. The basic DP approaches for restricted and unrestricted wavelet synopses (Sections 4.3.1.2 and 4.3.2) can be naturally extended to the case of multi-dimensional data (based on the appropriate multi-dimensional Haar-tree structures, e.g., Figure 4.4). However, the time and space complexity of the dynamic program increase explosively with data dimensionality rendering the approach inapplicable even for relatively small dimensionalities (e.g., 2–5). For the restricted case, Garofalakis and Kumar [112] introduce efficient approximation schemes based on *approximate dynamic programs* that explore a much smaller number of options than the optimal DP formulation, while offering tunable ε-approximation guarantees for the final target maximum-error metric.

4.4.3 Multi-Dimensional Range-Aggregate Query Estimation

For range-count aggregates, the observations and techniques discussed in the case of one-dimensional data (Section 4.2.4) can be naturally extended to the multi-dimensional case. In the case of the d-dimensional *standard* HWT (where the Haar "tree" structure can conceptually be thought of as the cross-product of the d one-dimensional Haar trees), a coefficient with value v at location $(i_1, i_2, \ldots, i_d)$ contributes to the range-`COUNT` query $A(l_1 : h_1, \ldots, l_{d'} : h_{d'})$ only if $i_{d'+1} = \cdots = i_d = 0$, and its contribution is given by the formula [284]:

$$v \cdot \prod_{j=1}^{d'} (|\texttt{leftleaves}_j(i_j, l_j : h_j)| - |\texttt{rightleaves}_j(i_j, l_j : h_j)|) \times \prod_{j=d'+1}^{d} M_j,$$

where $|\texttt{leftleaves}_j(i_j, l_j : h_j)|$ denotes the size of the intersection of the leaf set in the left child subtree of coefficient i_j (along dimension j) and the cor-

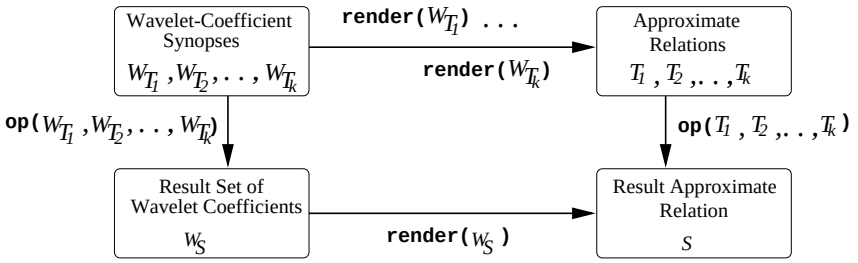


Fig. 4.7 Processing relational query operators efficiently in the wavelet-coefficient domain (i.e., the "down-then-right" execution path) [42].

responding one-dimensional query range $[l_j, h_j]$ (with $|\texttt{rightleaves}_j(i_j, l_j : h_j)|$ defined similarly). Using this fact, it can be shown that, given a synopsis comprising B standard HWT coefficients of A, the range-`COUNT` $A(l_1 : h_1, \ldots, l_{d'} : h_{d'})$ can be estimated in time $O(\min\{B, \prod_{i=1}^{d'} M_i\})$ [284]. Similar bounds and estimation algorithms can also be given for the case of the nonstandard decomposition, by exploiting the regular quadtree structure of the nonstandard Haar tree (Figure 4.4).

4.5 Approximate Processing of General SQL Queries

We now turn our attention to techniques for fast approximate processing of full-fledged SQL queries using wavelet synopses of data (thought of as relations). Going beyond simple range aggregations, Chakrabarti et al. [42] propose an approximate query processing algebra (which includes all conventional aggregate and non-aggregate SQL operators, such as `select`, `project`, `join`, `sum`, and `average`) that operates *directly over the wavelet synopses of relations*, while guaranteeing the correct relational operator semantics. Query processing algorithms for these operators work *entirely* in the wavelet-coefficient domain — that is, their input(s) and output are sets of wavelet coefficients (rather than relational tables). This allows for extremely fast response times, since the approximate query execution engine can do the bulk of its processing over compact wavelet synopses, essentially postponing the (expensive) expansion/rendering step into relational tuples until the end-result of the query (Figure 4.7).

In the approximate query processing framework of [42], a HWT coefficient is represented by a triple $W = \langle R, S, v \rangle$, where: (1) $W.R$ is the d-

dimensional support hyper-rectangle of W (represented by low and high boundary values along each dimension); (2) *W.S* stores the *coefficient sign information for different regions of W.R*, which can be captured by a 2-bit sign vector (storing the sign value $(+/-)$ for the low/high end of the coefficient range) plus the value where the sign changes, for each of the D dimensions; and, (3) *W.v* is the *(scalar) magnitude* of the coefficient. Based on this coefficient representation, the semantics of basic relational operators can be defined in a natural manner.

Selection, Projection, and Join. Consider a selection operator over a wavelet synopsis (i.e., set of HWT coefficients) W_T, $\texttt{select}_{pred}(W_T)$, where *pred* represents a generic conjunctive predicate on a subset of $k \leq d$ data attributes. This `select` operator effectively filters out the portions of the wavelet coefficients in the synopsis W_T that do not overlap with the k-dimensional selection range, and thus do not contribute to cells in the selected hyper-rectangle [42]. Similarly, the projection operator $\texttt{project}_{X_{i_1},\ldots,X_{i_k}}(W_T)$ (where $X_{i_1},\ldots,X_{i_k}$ denote the $k \leq d$ projection attributes) effectively projects out the remaining $d-k$ dimensions for each coefficient support hyper-rectangle and adjusts the coefficient's magnitude by an appropriate multiplicative constant (to aggregate the contributions of array cells that are "projected out") [42].

The most interesting operator in this approximate, wavelet-based relational algebra is probably the join, with the general form $\texttt{join}_{pred}(W_{T_1}, W_{T_2})$, where T_1 and T_2 are (approximate) relations of dimensionality d_1 and d_2, respectively, and *pred* is a conjunctive k-ary equi-join predicate of the form $(X_1^1 = X_1^2) \wedge \ldots \wedge (X_k^1 = X_k^2)$, where X_j^i $(j = 1,\ldots,d_i)$ denotes the j^{th} attribute of T_i $(i = 1,2)$. (Assuming, without loss of generality, that the join attributes are the first $k \leq \min\{d_1, d_2\}$ attributes of each joining relation.) Note that the result of the join operation is a set of $(d_1 + d_2 - k)$-dimensional wavelet coefficients; that is, the join operation returns coefficients of (possibly) different dimensionality than any of its inputs.

To understand the join processing algorithm of [42], consider the multi-dimensional joint-frequency arrays A_{T_1} and A_{T_2} corresponding to the join operator's input arguments. Let $(i_1^1,\ldots,i_{d_1}^1)$ and $(i_1^2,\ldots,i_{d_2}^2)$ denote the coordinates of two cells belonging to A_{T_1} and A_{T_2}, respectively. If the indexes of the two cells match on the join dimensions, i.e., $i_1^1 = i_1^2,\ldots,i_k^1 = i_k^2$, then the cell

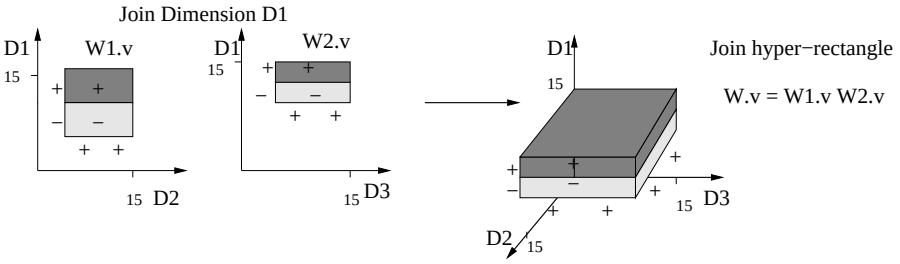


Fig. 4.8 Processing a `join` operation in the wavelet-coefficient domain.

in the join result array with coordinates $(i_1^1,\ldots,i_{d_1}^1,i_{k+1}^2,\ldots,i_{d_2}^2)$ is populated with the *product* of the count values contained in the two joined cells. Since the cell counts for A_{T_i} are derived by appropriately summing the contributions of the HWT coefficients in W_{T_i} and a numeric product can always be distributed over summation, the `join` operator can be processed entirely in the wavelet-coefficient domain by considering all pairs of coefficients from W_{T_1} and W_{T_2}. Briefly, for any two coefficients from W_{T_1} and W_{T_2} that overlap in the join dimensions, and, therefore, contribute to joining data cells, define an output coefficient with magnitude equal to the product of the two joining coefficients and a support hyper-rectangle with ranges that are (a) equal to the overlap of the two coefficients for the k (common) join dimensions, and (b) equal to the original coefficient ranges along any of the d_1+d_2-2k remaining dimensions [42]. The sign information for an output coefficient along any of the k join dimensions is derived by appropriately multiplying the sign-vectors of the joining coefficients along that dimension, taking care to ensure that only signs along the overlapping portion are taken into account. (The sign information along non-join dimensions remains unchanged.) An example of this process in two dimensions ($d_1=d_2=2$, $k=1$) is depicted in Figure 4.8.

Aggregation. Most conventional aggregation operators, like `count`, `sum`, and `avg`, can also be naturally processed over the wavelet-coefficient domain[10] [42]. As before, the input to each aggregate operator is a set of wavelet coefficients W_T. If the aggregation is not qualified with a `GROUP-BY`

[10]Like sampling and histograms, wavelet synopses are also inherently limited to trivial answers when it comes to "extrema" aggregates (e.g., `min` or `max`).

clause, then the output of the operator is a simple scalar value for the aggregate. In the more general case, where a `GROUP-BY` clause over dimensions $\mathscr{D}' = \{X_1, \ldots, X_k\}$ has been specified, the output of the aggregate operator consists of a k-dimensional array spanning the dimensions in $\mathscr{D}'$, whose entries contain the computed aggregate value for each cell [42].

Rendering into Approximate Relations. The final step of a wavelet-based approximate query processing plan is to *render* an output set W_S of d-dimensional wavelet coefficients into an approximate result relation (Figure 4.7). A naive, inefficient approach to rendering W_S would simply consider each cell in the multi-dimensional array A_S and sum the contributions of every coefficient in W_S to that cell in order to obtain the corresponding tuple count. A more efficient rendering algorithm (proposed in [42]) exploits the fact that, since the number of coefficients in W_S is typically much smaller than the number of cells in A_S, the result approximate array consists of large, contiguous multi-dimensional regions, where all the cells contain the same count. (In fact, because of the sparsity of the data, many of these regions will have counts of 0.) Furthermore, the total number of such "uniform-count" regions in A_S is typically considerably smaller than the number of A_S cells. Thus, an efficient rendering algorithm can partition the multi-dimensional array A_S, one dimension at a time, into such uniform-count data regions and output the (single) count value corresponding to each such region (the same for all enclosed cells). In order to partition A_S into uniform-count ranges along dimension i, the only points that should be considered are those where the cell counts along i could potentially change. These are precisely the points where a new coefficient's support hyper-rectangle starts or ends, or the sign of the coefficient's contribution changes (along dimension i). The process can be carried out efficiently using a priority queue for the coefficients in W_S [42].

4.6 Additional Topics

We conclude this chapter with a discussion of wavelet-based synopses over streaming data, techniques to optimize coefficient storage (e.g., for handling multi-measure data sets or compressing the synopsis), and "hybrid" methods combining features of both wavelets and histograms.

4.6.1 Wavelets over Streaming Data

When dealing with streaming data, the goal is to continuously track a compact synopsis of B wavelet coefficient values for a dynamic data vector A that is rendered as a continuous stream of updates. Algorithms for this problem should satisfy the key *small space/time* requirements for streaming algorithms; more formally, streaming wavelet algorithms should (ideally) guarantee (1) *sublinear space usage* (for storing a synopsis of the stream), (2) *sublinear per-item update time* (to maintain the synopsis), and (3) *sublinear query time* (to produce a, possibly approximate, wavelet summary), where "sublinear" typically means polylogarithmic in the (large) domain size M. The streaming wavelet summary construction problem has been examined under two distinct data streaming models.

Wavelets in the Ordered Aggregate Model

The bulk of the results on wavelets over streaming data assumes the limited *ordered aggregate* (or, *time-series*) model. Here, the entries of the input data vector A are rendered over time in the increasing (or, decreasing) order of the index domain values; this means, for instance, that $A[0]$ (or, the set of all updates to $A[0]$) is seen first, followed by $A[1]$, then $A[2]$, and so on. Thus, the synopsis construction algorithm works with the datacube access model but is essentially allowed only *one pass* over the $A[i]$ entries and memory that is significantly smaller than the size M of the data (typically in the order of B or $\mathsf{polylog}(M)$). Note that the algorithm for L_2-optimal wavelet synopses described in Section 4.2.3 is in fact such a streaming algorithm for time-series data that uses $O(B+\log M)$ space and $O(\log M)$ processing time per item [131].

For non-L_2 error, Karras and Mamoulis [202] describe heuristic extensions of their greedy maximum absolute/relative error algorithms (for the *restricted* version of the problem) that operate in a single pass over the $A[i]$ data entries employing only $O(B)$ memory. Their streaming methods work by discarding a pair of coefficients for each arriving pair of data entries: As in the static case, the pair is selected greedily (based on minimizing the maximum potential error), but with the scope now limited to the already-built part of the Haar tree. Through the use of appropriate data structures and auxiliary

information, the time complexity of the algorithm is also shown to be $O(B)$ per stream element [202].

For the *unrestricted* case, Guha and Harb [135] observe that the computation of their approximate DP algorithm (Section 4.3.2) can naturally be performed in a streaming fashion over the entries of A, using the general paradigm of "stream-and-merge" in a manner similar to the L_2 streaming algorithm in Section 4.2.3. The key observation is that, in order to compute the error array $E[i,\cdot,\cdot]$ at an internal node i, the algorithm only requires knowledge of the error arrays in i's two child nodes, $E[2i,\cdot,\cdot]$ and $E[2i+1,\cdot,\cdot]$. For instance, in Figure 4.1, when $A[0]$ arrives, the algorithm computes the error array associated with that leaf node, say $E_{A[0]}$. When $A[1]$ arrives, the error array $E_{A[1]}$ is computed, and immediately combined with $E_{A[0]}$ to compute the error array $E[4,\cdot,\cdot]$ for node c_4, after which both $E_{A[0]}$ and $E_{A[1]}$ are discarded. Proceeding in this manner, it is easy to see that, at any point in time, there is *at most one* error array stored at each level of the underlying Haar tree. This implies a streaming DP algorithm that employs the same space as the offline algorithm described in Section 4.3.2; for instance, the streaming algorithm guarantees an approximation of the unrestricted L_p-optimal Haar wavelet synopsis to within an additive error of $\varepsilon\alpha_{max}$ (where $\alpha_{max} = \max_i\{|A[i]|\}$) in time $O(\frac{1}{\varepsilon^2}M^{1+4/p}B(\min\{B,\log M\})^2)$ and space $O(\frac{1}{\varepsilon}M^{2/p}B\min\{B,\log M\}\log(M/B))$ (which is sublinear for $p > 2$) [135]. Analogous results hold for weighted and relative L_p error, as well as maximum (L_∞) error [135].

It is interesting to note that the unrestricted optimization of coefficient values actually *enables* the above streaming DP computation: Error arrays can be computed at each internal node i based on the estimated range R of incoming values and value selections at the node (Section 4.3.2), and this can be done independently of coefficient values at ancestors of i (which are required in the optimal *restricted* dynamic program of [112]). Using this observation, Guha and Harb [135] also suggest a simpler, faster variant of their unrestricted DP algorithm that offers a guaranteed additive-error approximation of the optimal *restricted* solution in the aggregate streaming model: The key idea is to only consider at each internal node i the better of rounding up or rounding down the coefficient value c_i to the nearest multiples of the rounding step δ (rather than considering the entire range R). This simplification

improves the running time by a factor of $|R|$, while guaranteeing an additive error of $\varepsilon\alpha_{max}$ over the optimal *restricted* wavelet synopsis [135].

Wavelets in the General Turnstile Model

As discussed in Chapter 3, the general turnstile streaming model allows updates to the data vector A to appear in arbitrary order in the stream; at any time point, the value of a vector entry is simply the aggregate of all updates for that entry seen so far. I.e., each streaming update is a pair of the form $(i, \pm v)$, denoting a net change of $\pm v$ in the $A[i]$ entry; that is, the effect of the update is to set $A[i] \leftarrow A[i] \pm v$. (The model naturally generalizes to multi-dimensional data: for d data dimensions, each update $((i_1, \ldots, i_d), \pm v)$ effects a net change of $\pm v$ on entry $A[i_1, \ldots, i_d]$.) The problem of maintaining an accurate wavelet summary becomes significantly more complex when moving to this much more general streaming model. For instance, Gilbert et al. [131] prove a strong *lower bound* on the space requirements of the problem — their result essentially shows that, for arbitrary data vectors rendered in the turnstile streaming model, *nearly all* of the data must be stored to recover the exact top-B HWT coefficients.

In early work on incremental wavelet maintenance, Matias et al. [226] consider the problem of maintaining the top-B HWT coefficients (i.e., an L_2-optimized Haar wavelet synopsis) in the presence of such turnstile updates to the data. They observe that, given an update $(i, \Delta(i))$ to entry $A[i]$, the value of a coefficient $c_j \in \texttt{path}(A[i])$ in the synopsis can be updated in constant time using the simple formula

$$c_{j,new} = \begin{cases} c_{j,old} + \frac{\Delta(i)}{2^{\log M - l(j)}} & \text{if } A[i] \in \texttt{leftleaves}(c_j) \\ c_{j,old} - \frac{\Delta(i)}{2^{\log M - l(j)}} & \text{if } A[i] \in \texttt{rightleaves}(c_j) \end{cases},$$

where $l(j)$ denotes the level of resolution of coefficient c_j. Still, a key problem is how to detect that a *new coefficient* becomes significant and needs to enter the top-B synopsis (due to a shift in the underlying data distribution), and *what value* should be assigned to this new coefficient given that it was not monitored as part of the previous top-B collection. Matias et al. [226] propose a solution based on a *probabilistic counting* technique [233]: Given an update to $A[i]$ that affects coefficient c_j (i.e., $c_j \in \texttt{path}(A[i])$), a coin is flipped with a probability $p(j)$ of heads. If the coin flip gives heads, then coefficient c_j

is set to a value $v(j)$, and it replaces the smallest coefficient in the current top-B synopsis. Intuitively, the idea is to set the parameters $p(j)$ and $v(j)$ so that $1/p(j)$ corresponds to the (expected) number of updates needed at leaf $A[i]$ to bring the magnitude of coefficient c_j from its initial value of zero to $|v(j)|$. Based on the earlier formula, this gives $p(j) = 1/(|v(j)| \cdot 2^{\log M - l(j)})$. The value of parameter $v(j)$ depends on both the current synopsis and the position of the update; more specifically, the magnitude $|v(j)|$ is heuristically determined as a constant multiple of the minimum coefficient magnitude in the synopsis, while the sign of the coefficient is a $+$ ($-$) if the update occurs in its left (resp., right) subtree [226]. This probabilistic counting heuristic extends naturally to multi-dimensional wavelets and seems to perform adequately in practice [226]; however, it cannot provide any guarantees on the quality of the maintained synopsis.

More recent solutions for maintaining an L_2-optimized wavelet synopsis over turnstile data streams rely on randomized schemes that return only an approximate synopsis comprising (at most) B Haar coefficients that is *provably near-optimal* (in terms of the underlying L_2 error) assuming that the data vector satisfies the *"small-B property"* (i.e., most of its energy is concentrated in a small number of HWT coefficients) — this assumption is typically satisfied for most real-life data distributions [131]. One of the key ideas is to maintain a randomized *AMS sketch* [7], a broadly applicable stream synopsis structure comprising randomized linear projections of the streaming data vector A. The AMS sketch and its applications are examined in detail in Chapter 5; here, we just briefly outline some of its main properties in the context of dynamic wavelet maintenance. An *atomic AMS sketch* of A is simply the *inner product* $\langle A, \xi \rangle = \sum_i A[i]\xi(i)$, where ξ denotes a random vector of four-wise independent ± 1-valued random variates. Such variates can be easily generated on-line through standard pseudo-random hash functions $\xi()$ using only $O(\log M)$ space (for seeding) [7, 131]. To maintain this inner product over the stream of updates to A, initialize a running counter X to 0 and set $X \leftarrow X \pm v\xi(i)$ whenever the update $(i, \pm v)$ is seen in the input stream. An *AMS sketch* of A comprises several independent atomic AMS sketches (i.e., randomized counters), each with a different random hash function $\xi()$.

It can be shown that, for any two M-vectors (say, A and B), if we let Z denote the $O(\log(1/\delta))$-wise median of $O(1/\varepsilon^2)$-wise means of independent

copies of the atomic AMS sketch product $(\sum_i A[i]\xi_j(i))(\sum_i B[i]\xi_j(i))$, then, $|Z - \langle A,B\rangle| \leq \varepsilon||A||_2||B||_2$ with probability $\geq 1-\delta$ [7, 6]. In other words, using (lower-dimensional) AMS sketches comprising only $O(\frac{\log(1/\delta)}{\varepsilon^2})$ (typically, $\ll M$) atomic counters we can approximate the vector inner product $\langle A,B\rangle$ to within $\pm\varepsilon||A||_2||B||_2$ (hence, implying an ε-relative error estimate for the squared L_2 norm $||A||_2^2$).

The above property, in conjunction with the fact that Haar coefficients of A are inner products of A with a fixed set of wavelet-basis vectors, forms the key to developing efficient, approximate wavelet maintenance algorithms in the turnstile model. Gilbert et al. [131] propose a solution (termed "GKMS" in the remainder of our discussion) that focuses primarily on the one-dimensional case. GKMS maintains an AMS sketch for the streaming data vector A. To produce the approximate L_2-optimized B-term representation, GKMS employs the constructed sketch of A to estimate the inner product of A with *all wavelet basis vectors*, essentially performing an exhaustive search over the space of all wavelet coefficients to identify important ones. More formally, assuming that there is a B-coefficient approximate representation of the signal with energy at least $\eta||A||_2^2$ ("small B property"), the GKMS algorithm uses a maintained AMS sketch to exhaustively estimate each Haar coefficient and selects up to B of the largest coefficients (excluding those whose square is less than $\eta\varepsilon||A||_2^2/B$, where $\varepsilon < 1$ is the desired accuracy guarantee). GKMS also uses techniques based on range-summable random variables constructed using Reed-Muller codes to reduce or amortize the cost of this exhaustive search by allowing the sketches of basis vectors (with potentially large supports) to be computed more quickly. Summarizing, the key result of [131] states that, assuming there exists a B-term representation with energy at least $\eta||A||_2^2$, then, with probability at least $1-\delta$, the GKMS algorithm finds a representation of at most B coefficients that captures at least $(1-\varepsilon)\eta$ of the signal energy $||A||_2^2$, using $O(\log^2 M \log(M/\delta)B^2/(\eta\varepsilon)^2)$ space and per-item processing time.

A potential problem lies in the query time requirements of the GKMS algorithm: even with the Reed-Muller code optimizations, the overall query time for discovering the top coefficients remains superlinear in M (i.e., at least $\Omega(\frac{1}{\varepsilon^2}M\log M)$), violating our third requirement on streaming schemes. This also renders direct extensions of GKMS to multiple dimensions infeasi-

ble since it implies an exponential explosion in query cost (requiring at least $O(M) = O(m^d)$ time to cycle through all coefficients in d dimensions). In addition, the update cost of the GKMS algorithm is *linear in the size of the sketch* since the whole data structure must be "touched" for each update. This is problematic for high-speed data streams and/or even moderate sized sketch synopses.

To address these issues, Cormode et al. [64] propose a novel solution that relies on two key technical ideas. First, they work *entirely in the wavelet domain*: instead of sketching the original data entries, their algorithms sketch the wavelet-coefficient vector W_A as updates arrive. This avoids any need for complex range-summable hash functions (i.e., Reed-Muller codes). Second, they employ *hash-based grouping* in conjunction with *efficient binary-search-like techniques* to enable very fast updates as well as identification of important coefficients in polylogarithmic time.

- *Sketching in the Wavelet Domain.* The idea here is that it is possible to efficiently produce sketch synopses of the stream *directly in the wavelet domain*; that is, the impact of each streaming update can be translated on the relevant wavelet coefficients [64]. By the linearity properties of the HWT and the earlier description, an update to the data entries corresponds to only polylogarithmically many coefficients in the wavelet domain. Thus, on receiving an update to A, it can be directly converted to $O(\mathsf{polylog}(M))$ updates to the wavelet coefficients, and an approximate (sketch) representation of the wavelet coefficient vector W_A can be maintained.
- *Time-Efficient Updates and Large-Coefficient Searches.* Sketching in the wavelet domain means that, at query time, an approximate representation of the wavelet-coefficient vector W_A is available, and can be employed to identify all those coefficients that are "large", relative to the total energy of the data $\|W_A\|_2^2 = \|A\|_2^2$. While AMS sketches can provide such estimates (a point query is just a special case of an inner product), querying remains much too slow taking at least $\Omega(\frac{1}{\varepsilon^2}M)$ time to find which of the M coefficients are the B largest. Instead, the schemes in [64] rely on a *divide-and-conquer* or *binary-search-like* approach for finding the large coefficients. This requires the ability to efficiently esti-

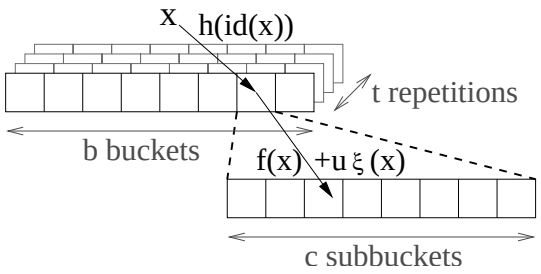


Fig. 4.9 The GCS data structure: Element x is hashed (t times) to a bucket of groups (using $h(\mathtt{id}(x))$) and then a sub-bucket within the bucket (using $f(x)$), where an AMS counter is updated.

> mate sums-of-squares for *groups* of coefficients, corresponding to dyadic subranges of the domain $[0,M-1]$. Low-energy regions can then be disregarded, recursing only on high-energy groups — this guarantees no false negatives, as a group that contains a high-energy coefficient will also have high energy as a whole.

The key to the solution of [64] is a hash-based probabilistic synopsis data structure, termed *Group-Count Sketch (GCS)*, that can estimate the energy of fixed groups of elements from a vector W of size M under the turnstile streaming model. This translates to several streaming L_2-norm estimation problems (one per group). A simple solution would be to keep an AMS sketch of each group separately; however, there can be *many* (e.g., linear in M) groups, implying space requirements that are $O(M)$. Streaming updates should also be processed as quickly as possible. The GCS synopsis requires small, sublinear space and takes sublinear time to process each stream update item; more importantly, a GCS can provide a high-probability estimate of the energy of a group within additive error $\varepsilon\|W\|_2^2$ in *sublinear time*. The GCS synopsis first partitions items of w into their group using an `id()` function (which, in the case of Haar coefficients, is trivial since it corresponds to fixed dyadic ranges over $[0,M-1]$), and then randomly maps groups to buckets using a hash function $h()$. Within each bucket, a second stage of hashing of items to sub-buckets is applied (using another hash function $f()$), where each contains an atomic AMS sketch counter in order to estimate the L_2 norm of the elements

in the bucket.[11] As with most randomized estimation schemes, a GCS synopsis comprises t independent instantiations of this basic randomized structure, each with independently chosen hash function pairs $(h(), f())$ and ξ families for the AMS estimator; during maintenance, a streaming update (x,u) is used to update each of the t AMS counters corresponding to element x. (A pictorial representation of the GCS synopsis is shown in Figure 4.9.) To estimate the energy of a group g, for each independent instantiation $m = 1,\ldots,t$ of the bucketing structure, the squared values of all the AMS counters in the sub-buckets corresponding to bucket $h_m(g)$ are summed, and then the *median* of these t values is returned as the estimate. Summarizing, the analysis of [64] demonstrates that the GCS can estimate the energy of item groups of the vector w within additive error $\varepsilon\|w\|_2^2$ with probability $\geq 1-\delta$ using space of $O\left(\frac{1}{\varepsilon^3}\log\frac{1}{\delta}\right)$ counters, per-item update time of $O\left(\log\frac{1}{\delta}\right)$, and query time of $O\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\right)$.

To recover coefficients with large energy in the w vector, the algorithm [64] employs a *hierarchical search-tree structure* on top of $[0,M-1]$: Each level in this tree structure induces a certain partitioning of elements into groups (corresponding to the nodes at that level), and per-level GCS synopses can be used to efficiently recover the high-energy groups at each level (and, thus, quickly zero in on high-energy Haar coefficients). Using these ideas, Cormode et al. [64] demonstrate that the accuracy guarantees of [131] can be obtained using $O(\frac{B^3\log M}{\varepsilon^3\eta^3}\cdot\log\frac{B\log M}{\varepsilon\eta\delta})$ space, $O(\log^2 M\cdot\log\frac{B\log M}{\varepsilon\eta\delta})$ per item processing time, *and* $O(\frac{B^3}{\varepsilon^3\eta^3}\cdot\log M\cdot\log\frac{B\log M}{\varepsilon\eta\delta})$ query time. In other words, the GCS-based solution guarantees sublinear space *and* query time, as well as per-item processing times that are sublinear *in the size of the stream synopsis*. Their results also naturally extend to the multi-dimensional case [64].

Optimizing Non-L_2 Error. To the best of our knowledge, there are no known results on approximating (restricted or unrestricted) wavelet synopses optimized for *non-L_2 error metrics* in the turnstile streaming model. This is a challenging problem that appears to mandate novel solution techniques, as traditional sketch-based approaches are only useful in the discovery of large-magnitude coefficients (concentrating a large portion of the data-vector

[11] Note that this second-level bucket structure essentially results in a "fast AMS" sketch (Section 5.3.3.1) that is used for L_2 estimation.

energy).

4.6.2 Optimizing Coefficient Storage: Extended and Hierarchically-Compressed Wavelet Synopses

Extended Wavelets. Massive complex tabular data sets *with multiple measures* (multiple numeric entries for each table cell) arise naturally in several application domains, including OLAP environments and time-series analysis/correlation systems. As an example, a corporate sales database may tabulate, for each available product, (1) the number of items sold, (2) revenue and profit numbers for the product, and (3) costs associated with the product, such as shipping and storage costs. Similarly, real-life applications that monitor continuous time-series typically have to deal with several readings (measures) that evolve over time; for example, a network-traffic monitoring system takes readings on each time-tick from a number of distinct elements, such as routers and switches, in the underlying network and typically several measures of interest need to be monitored (e.g., input/output traffic numbers for each router or switch interface) even for a fixed network element.

Traditionally, two obvious strategies, termed *Individual* and *Combined*, have been employed when adapting wavelet-based methods over such multi-measure data sets [276]. The *Individual* algorithm performs the wavelet decomposition on each of the individual measures, and stores the important coefficients for each measure separately. On the other hand, the *Combined* algorithm performs a joint wavelet decomposition on the multi-measure data set by treating all the measures as a *vector* of values and, at the end, determines a subset of vectors of coefficient values to retain in the synopsis (e.g., based on the L_2 norm of the coefficient vectors). As an example, Table 4.2 depicts the set of combined HWT coefficients for a data set with $\mu = 2$ measures, where the values for the first measure (i.e., first vector row) are identical to those in our first example array in Section 4.2.1, while the values for the second measure (i.e., second row) are $[4, 6, 3, 5, 2, 8, 3, 3]$.

Such "obvious" *Individual* and *Combined* approaches can lead to poor synopsis-storage utilization and suboptimal solutions even in very simple cases [81]. Due to the nature of the wavelet decomposition and the possible correlations across different measures, there are many scenarios in which multiple – but not necessarily all – wavelet coefficients at the same coor-

Resolution	Averages	Detail Coefficients
3	$[\begin{bmatrix}2\\4\end{bmatrix}\begin{bmatrix}2\\6\end{bmatrix}\begin{bmatrix}0\\3\end{bmatrix}\begin{bmatrix}2\\5\end{bmatrix}\begin{bmatrix}3\\2\end{bmatrix}\begin{bmatrix}5\\8\end{bmatrix}\begin{bmatrix}4\\3\end{bmatrix}\begin{bmatrix}4\\3\end{bmatrix}]$	—
2	$[\begin{bmatrix}2\\5\end{bmatrix}\begin{bmatrix}1\\4\end{bmatrix}\begin{bmatrix}4\\5\end{bmatrix}\begin{bmatrix}4\\3\end{bmatrix}]$	$[\begin{bmatrix}0\\-1\end{bmatrix}\begin{bmatrix}-1\\-1\end{bmatrix}\begin{bmatrix}-1\\-3\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}]$
1	$[\begin{bmatrix}3/2\\9/2\end{bmatrix}\begin{bmatrix}4\\4\end{bmatrix}]$	$[\begin{bmatrix}1/2\\1/2\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}]$
0	$[\begin{bmatrix}11/4\\17/4\end{bmatrix}]$	$[\begin{bmatrix}-5/4\\1/4\end{bmatrix}]$

Table 4.2 Example Combined Wavelet Decomposition.

dinates have large values, and are thus beneficial to retain, for instance, in an L_2-optimized synopsis. In such cases, the *Individual* algorithm essentially replicates the storage of the shared coordinates multiple times, wasting valuable synopsis storage. The *Combined* algorithm, on the other hand, stores *all* coefficient values sharing the same coordinates, thus wasting space by storing small, unimportant values for certain measures. To address these shortcomings, Deligiannakis et al. [81, 82] introduce the notion of an *extended wavelet coefficient* as an efficient, flexible storage format for retaining *any subset* of coefficient values. Briefly, an extended wavelet coefficient EC for a data set with μ measures is defined as a triple $EC = \langle C, \beta, V \rangle$ consisting of: (1) The coordinates C of the coefficient; (2) A bitmap β of size μ where the i^{th} bit denotes the existence or absence of a coefficient value for the i^{th} measure; and, (3) The set of stored coefficient values V. A key problem is then to construct an extended-coefficient synopsis that optimizes the weighted sum of L_2-error norms across all μ measures; more formally, letting $A_j[i]$ denote the i^{th} data value for the j^{th} measure, to goal is to select a synopsis $\mathscr{S}$ of extended coefficients such that minimizes

$$\sum_{j=1}^{\mu} \left(w_j \times \sum_i (A_j[i] - \hat{A}_j[i])^2 \right) \text{ subject to the space constraint } \sum_{EC \in \mathscr{S}} |EC| \leq B,$$

where $|EC|$ denotes the space requirement of an extended coefficient EC. By Parseval's theorem, the above loss minimization problem is equivalent to *maximizing the total benefit* of the coefficients selected in the synopsis $\mathscr{S}$, where the benefit of the (normalized) i^{th} coefficient for the j^{th} measure c^*_{ji} is

exactly its weighted squared magnitude $w_j(c_{ji}^*)^2$ [81, 82]; that is, maximize

$$\sum_{EC=\langle C,\beta,V\rangle\in\mathscr{S}} \left(\sum_{j,\beta(j)=1} w_j \times (c_{ji}^*)^2 \right) \text{ under the constraint } \sum_{EC\in\mathscr{S}} |EC| \leq B.$$

Deligiannakis et al. [81, 82] design an optimal DP algorithm for solving the above extended wavelet synopsis construction problem given the full sequence of combined wavelet coefficients for a multi-measure data set. A key complication here arises from the storage dependencies across measures due to the fact that the storage penalty for an extended coefficient's "header" (i.e., the (coordinates, bitmap) information) is shared across all chosen coefficient values. Thus, their solution relies on two mutually-recursive DP recurrences that are tabulated in parallel to compute an optimal solution using $O(M\mu B)$ time and space. They also propose a simpler, faster greedy algorithm that selects coefficient values for the synopsis based on their benefit-to-space ratio, and requires only $O(M\mu^2\log(M\mu))$ time and $O(M\mu)$ space; furthermore, they prove that their greedy solution is a $\min\{2, 1+\frac{1}{\frac{B}{MAX}-1}\}$ approximation for the extended-coefficient benefit-maximization problem, where *MAX* is the maximum space requirement for an extended coefficient (i.e., storing all μ values) [81]. Finally, they discuss the extension of earlier results on non-L_2 error minimization for single-measure data [110, 112] to the multi-measure setting using the idea of *Partial-Order DP* over the μ measure dimensions, as well as more efficient greedy heuristics for the problem [81].

In follow-up work, Guha et al. [138] make the crucial observation that the DP solution only needs to decide *the number* $p \in [0,\mu]$ of values to store for each combined coefficient, since the p values that rank highest in terms of weighted benefit $w_j(c_{ji}^*)^2$ always give the highest benefit among all p-subsets. This allows for a simpler and more efficient optimal DP solution that runs in $O(M\mu(\log\mu+\log B)+\mu^2B^2)$ time and $O(\mu B+B^2)$ space. Furthermore, they propose an efficient $O(M\mu(\log\mu+\log B))$ greedy strategy (also based on the coefficient values' benefit-to-space ratio) that guarantees a benefit that is no less than that of the optimal solution while using only slightly more space ($B+MAX$ instead of B) [138].

Hierarchically-Compressed Wavelet Synopses. One of the key ideas of extended wavelets is that the storage of coordinates for related coefficient val-

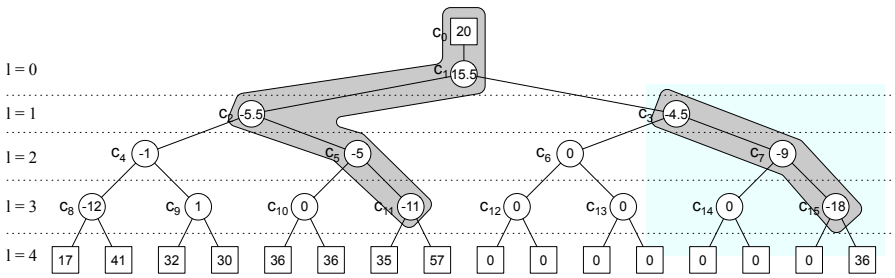


Fig. 4.10 Example Haar-tree structure depicting two hierarchically-compressed coefficients.

ues can be effectively shared in order to improve the quality of wavelet synopses for complex, multi-measure data. Sacharidis et al. [267] observe that similar ideas can also be exploited across hierarchically-related coefficients to effectively compress conventional, single-measure wavelet synopses. In a nutshell, the idea is to store entire *paths of wavelet coefficients* using only the coordinates of the bottom-most coefficient c_{bot} on the path, along with the set of coefficient values (ancestors of c_{bot}) and a small bitmap indicating the number of nodes on the stored path. A *Hierarchically-Compressed Wavelet Synopsis (HCWS)* then comprises a set of such hierarchically-compressed (HC) coefficients whose total storage requirements do not exceed a given space budget B. As an example, Figure 4.10 depicts two hierarchically-compressed coefficients over the Haar tree of a 16-entry data array — the corresponding triples of (bottom-most coordinates, bitmap, $\{$values$\}$) are $(11, 11110, \{-11, -5, -5.5, 15.5, 20\})$ and $(15, 110, \{-18, -9, -4.5\})$ (where the last 0 entry in the bitmap acts as a stop bit).

By avoiding the explicit storage of coefficient coordinates, a HCWS can allow more coefficient values to be stored for the given space budget, thus resulting in potentially significant gains in accuracy. This is especially the case for data sets with multiple spikes and discontinuities where the Haar-wavelet differencing process across averages of neighboring regions can give multiple hierarchically-nested coefficients with large values [267]. For the example in Figure 4.10, assuming a coordinate and a coefficient value each require 32 bits, the two HC coefficients store 8 coefficient values requiring a total of 328 bits, which would only allow storing only 5 coefficient values in the traditional scheme — the difference in the respective L_2 errors is also significant

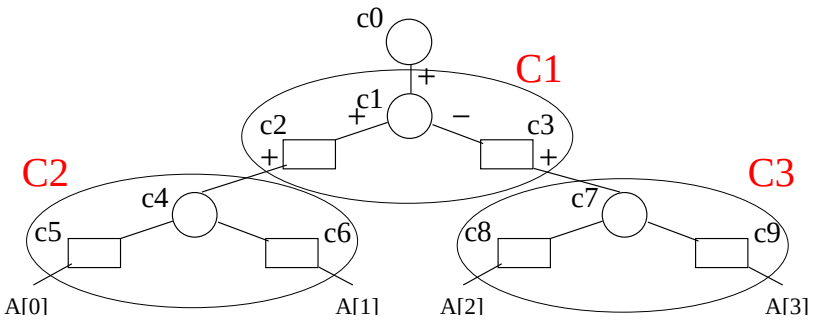


Fig. 4.11 Example Haar$^+$-tree structure over a 4-element data array.

(over 60%). Sacharidis et al. [267] formulate and solve the problem of constructing an L_2-optimal HCWS for a given space budget B. They propose an optimal DP algorithm that employs two mutually-recursive DP arrays (similar to the dynamic program in [81, 82]) computed over the Haar-tree structure in $O(MB)$ time using $O(M \log B)$ space. Their work also proposes faster approximation algorithms and greedy heuristics for the HCWS construction problem, and discusses extensions to multi-dimensional and streaming data.

4.6.3 Histogram-Wavelet Hybrids: The Haar$^+$ Tree

Hierarchical histogram synopses, such as *lattice histograms* (Section 3.4.3), attempt to capture the underlying data distribution using a collection of histogram buckets defined on a hierarchical structure over the data domain. Such hierarchical bucketing structures are clearly reminiscent of Haar wavelet coefficients; there is, however, one key difference: a bucket can only contribute positively to its descendants in the hierarchy. A more direct histogram-wavelet hybrid that tries to reap the benefits of both summarization techniques is the *Haar$^+$-tree* synopsis proposed by Karras and Mamoulis [203].

Figure 4.11 depicts a simple one-dimensional Haar$^+$ tree over a four-element data vector. As in a conventional Haar tree, the root coefficient node c_0 contributes its value (with a positive sign) to all data values; however, conventional Haar detail coefficient nodes have been replaced by *coefficient triads* (denoted C_1–C_3). In each triad, the *head* (or, root) coefficient (c_1, c_4, and c_7) behaves as a conventional HWT detail coefficient: it contributes its value positively to its left subtree and negatively to its right subtree; the other

two, left and right *supplementary* coefficients in each triad (e.g., c_2 and c_3 in C_1) contribute their value only positively in their corresponding support interval. For instance, if a value for c_3 is retained in the Haar$^+$ synopsis, it contributes positively in the reconstruction of $A[2]$ and $A[3]$. (As noted in [200], the Haar$^+$ tree can be seen as a direct merger of Haar trees and the Compact Hierarchical Histograms of Reiss et al. [264].)

The Haar$^+$-tree structure has some interesting properties [203]: First, it generalizes the traditional Haar tree, which is essentially a Haar$^+$ tree with all supplementary coefficients set to zero. Second, in a manner similar to Haar trees, each data (i.e., leaf) node value $A[i]$ is estimated through the sum $\sum_{c_j \in \texttt{path}(A[i])} \delta_{i,j} c_j$, where $\delta_{i,j} = -1$ if $(i-1) \mod 3 = 0$ and $A[i] \in \texttt{rightleaves}(c_j)$, and $\delta_{i,j} = +1$ otherwise. Finally, define the *state* of a coefficient triad (c_i, c_{i+1}, c_{i+2}) in a given Haar$^+$ tree as a 4-element vector $[v, a, b, c]$, where $v = \sum_{c_j \in \texttt{path}(c_i)} \delta_{i,j} c_j$ is the reconstructed value from the root up to (head) coefficient c_i (i.e., the incoming value at c_i), and a, b, c are the values at c_i, c_{i+1}, and c_{i+2}, respectively. It is easy to see that this triad state produces the pair of incoming values $v+a+b$ and $v-a+c$ to its left and right child triads, respectively. A first observation here is that a triad in a Haar$^+$ tree never needs to include both a non-zero head and a non-zero supplementary coefficient: A non-zero head can always be "pushed down" to its children, by transforming the state $[v, a, b, c]$ to the equivalent state $[v, 0, b+a, c-a]$ (producing exactly the same output). Similarly, a triad in the state $[v, 0, b, c]$ can be reduced to the equivalent state $[v + \frac{b+c}{2}, \frac{b-c}{2}, 0, 0]$ with the triad containing *only one* non-zero coefficient, by "pushing up" the average $\frac{b+c}{2}$ to the parent coefficient of the triad. Applying this transformation in a bottom-up fashion over the Haar$^+$-tree structure, allows the transformation of any Haar$^+$ tree H into an *equivalent* Haar$^+$ tree H' (producing exactly the same data values), such that H' is at least as sparse as H (i.e., contains the same or smaller number of non-zero coefficients) and every triad in H' contains *at most one* non-zero coefficient [203].

The above analysis simplifies the search for error-optimal B-term Haar$^+$-tree representations, since it implies that the search only needs to consider at most one non-zero coefficient in each triad of the optimal solution. Letting $E[i, b, v]$ denote the minimum error value at coefficient triad C_i with an incoming value of v and space b allocated to the subtree rooted at C_i, Karras and Mamoulis [203] present a generalization of the unrestricted DP thresh-

olding algorithm in Section 4.3.2 that can construct error-optimal Haar$^+$-tree synopses for a wide class of distributive error metrics (that include the various L_p-norm errors). A key result of [203] is an analysis of the Haar$^+$-tree structure that allows the effective delimitation of the search space for possible incoming and coefficient values to a range of values bounded by a small constant multiple of $\Delta = \max_i\{A[i]\} - \min_i\{A[i]\}$ (i.e., the range of underlying data values). Using an analysis similar to that of Section 4.3.2, the Haar$^+$-tree DP algorithm is shown to run in time $O((\frac{\Delta}{\delta})^2 MB)$, where δ is small quantization parameter that controls additive approximation error [203]. Extensions of Haar$^+$-tree synopses to multiple dimensions, as well as maximum-error optimization techniques based on the dual (space-bounded) problem are discussed in [200].

5

Sketches

5.1 Introduction

Of all the methods for generating data synopses presented in this volume, *sketches* have the shortest history, and consequently have had the least direct impact on real systems thus far. Nevertheless, their flexibility and power suggests that they will surely become a fixture in the next generation of tools and systems for working with large data. Certainly, they have already had significant impact within various specialized domains that process large quantities of structured data, in particular those that involve the *streaming* processing of data.

The notion of *streaming* has been popularized within recent years to capture situations where there is one chance to view the input, as it "streams" past the observer. For example, in processing financial data streams (streams of stock quotes and orders), many such transactions are witnessed every second, and a system must process these as they are seen, in real time, in order to facilitate real time data analysis and decision making. Another example that is closer to the motivating applications discussed so far is the sequence of updates to a traditional database—insertions and deletions to a given relation—which also constitute a stream to be processed. Streaming algorithms typi-

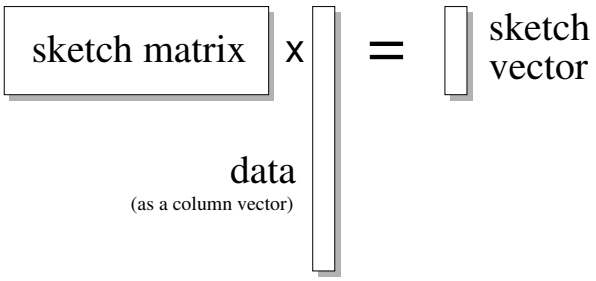


Fig. 5.1 Schematic view of linear sketching

cally create a compact synopsis of the data which has been observed, which is usually vastly smaller than the full data. Each update observed in the stream potentially causes this synopsis to be modified, so that at any moment the synopsis can be used to (approximately) answer certain queries over the original data. This fits exactly into our model of approximate query processing: provided we have chosen the right synopsis, the summary becomes a tool for AQP, in the same sense as a sample, or histogram or wavelet representation.

The earliest non-trivial streaming algorithms can be traced back to the late 1970s and early 1980s, when "pass efficient" algorithms for finding the median of a sequence and for finding the most frequently occurring items in a sequence were proposed [234, 229]. However, the growth in interest in streaming as a mechanism for coping with large quantities of data was stimulated by some influential papers in the late 1990s [7, 167], resulting in an explosion of work on stream processing in the first decade of the 21st Century.

We restrict our focus in this chapter to a certain class of streaming summaries known as *sketches*. This term has a variety of connotations, but in this presentation we use it to refer to a summary where each update is handled in the same way, irrespective of the history of updates. This notion is still rather imprecise, so we distinguish an important subset of *linear sketches*. These are data structures which can be represented as a *linear transform* of the input. That is, if we model a relation as defining a vector or matrix (think of the vector of discrete frequencies summarized by a histogram), then the sketch of this is found by multiplying the data by a (fixed) matrix. This is illustrated in Figure 5.1: a fixed sketch matrix multiplies the data (represented as a column

vector) to generate the sketch (vector). Such a summary is therefore very flexible: a single update to the underlying data (an insertion or deletion of a row in the relational data example) has the effect of modifying a single entry in the data vector. In turn, the sketch is modified by adding to the sketch the result of applying the matrix to this change alone. This meets our requirement that an update has the same impact irrespective of any previous updates. Another property of linear sketches is that the sketch of the union of two relations can be found as the (vector) sum of their corresponding sketches.

Any given sketch is defined for a particular set of queries. Queries are answered by applying some (technique specific) procedure to a given sketch. In what follows we will see a variety of different sketches. For some sketches, there are several different query procedures that can be used to address different query types, or give different guarantees for the same query type.

We comment that the idea of sketches, and in particular the linear transform view is not so very different from the summaries we have seen so far. Many histogram representations with fixed bucket boundaries can be thought of as linear transforms of the input. The Haar Wavelet Transform is also a linear transform[1]. However, for compactness and efficiency of computation, it is not common to explicitly materialize the (potentially very large) matrix which represents the sketch transform. Instead, all useful sketch algorithms perform a transform which is defined *implicitly* by a much smaller amount of information, often via appropriate randomly chosen hash functions. This is analogous to the way that a histogram transform is defined implicitly by its bucket boundaries, and the HWT is defined implicitly by the process of averaging and differencing.

5.2 Notation and Terminology

As in the preceding sections, we primarily focus on discrete data. We think of the data as defining a multiset D over a domain $\mathscr{U} = \{1,2,\ldots,M\}$ so that $f(i)$ denotes the number of points in D having a value $i \in \mathscr{U}$. These $f(i)$ values therefore represent a set of frequencies, and can also be thought of as defining a vector f of dimension $M = |\mathscr{U}|$. In fact, many of the sketches

[1] A key conceptual difference between the use of HWT and sketches is that the HWT is lossless, and so requires additional processing to produce a more compact summary via thresholding, whereas the sketching process typically provides data reduction directly.

we will describe here also apply to the more general case where each $f(i)$ can take on arbitrary real values, and even negative values. We say that f is "strict" when it can only take on non-negative values, and talk of the "general case" when this restriction is dropped.

The sketches we consider, which were primarily proposed in the context of streams of data, can be created from a stream of updates: think of the contents of D being presented to the algorithm in some order. Following Muthukrishnan [236], a stream updates is referred to as a "time-series" if the updates arrive in sorted order of i; "cash-register" if they arrive in some arbitrary order; and "turnstile" if items which have previously been observed can subsequently been removed. "Cash-register" is intended to conjure the image of a collection of unsorted items being rung up by a cashier in a supermarket, whereas "turnstile" hints at a venue where people may enter or leave. Streams of updates in the time-series or cash-register models necessarily generate data in the strict case, whereas the turnstile model can provide strict or general frequency vectors, depending on the exact situation being modeled.

These models are related to the datacube and relational models discussed already: the cash-register and turnstile models, and whether they generate strict or general distributions, can all be thought of as special cases of the relational model. Meanwhile, the time-series model is similar to the datacube model. Most of the emphasis in the design of streaming algorithms is on the cash-register and turnstile models. For more details on models of streaming computations, and on algorithms for streaming data generally, see some of the surveys on the topic [236, 12, 106].

Modeling a relation being updated with insert or delete operations, the number of rows with particular attribute values gives a strict turnstile model. But if the goal is to summarize the distribution of the sum of a particular attribute, grouped by a second attribute, then the general turnstile model may be generated. In both cases, sketch algorithms are designed to correctly reflect the impact of each update on the summary.

5.2.1 Simple Examples: Count, Sum, Average, Variance, Min and Max

Within this framework, perhaps the simplest example of a linear sketch computes the cardinality of a multiset D: this value N is simply tracked exactly, and incremented or decremented with each insertion into D or deletion from

D respectively. The sum of all values within a numeric attribute can also be sketched trivially by maintaining the exact sum and updating it accordingly. These fit our definition of being a (trivial) linear transformation of the input data. The average is found by dividing the sum by the count. Here, in addition to the maintenance of the sketch, it was also necessary to define an operation to extract the desired query answer from the sketch (the division operation). The sample variance of a frequency vector can also be computed in a sketching fashion, by tracking the appropriate sums and sums of squared values.

Considering the case of tracking the maximum value over a stream of values highlights the restriction that linear sketches must obey. There is a trivial streaming algorithm to find the maximum value of a sequence—just remember the largest one seen so far. This is a sketch, in the sense that every value is treated the same way, and the sketch maintenance process keeps the greatest of these. However, it is clearly *not* a linear sketch. Note that any linear sketch algorithm implicitly works in the turnstile model. But there can be no efficient streaming algorithm to find the maximum in a turnstile stream (where there are insertions and deletions to the dataset D): the best thing to do is to retain f in its entirety, and report the greatest value i for which $f(i) > 0$.

5.2.2 Fingerprinting as sketching

As a more involved example, we describe an method to *fingerprint* a data set D using the language of sketching. A fingerprint is a compact summary of a multiset so that if two multisets are equal, then their fingerprints are also equal; and if two fingerprints are equal then the corresponding multisets are also equal with high probability (where the probability is over the random choices made in defining the fingerprint function). Given a frequency vector f, one fingerprint scheme computes a fingerprint as

$$h(f) = \sum_{i=1}^{M} f(i)\alpha^i \mod p$$

where p is a prime number sufficiently bigger than M, and α is a value chosen randomly at the start. We observe that $h(f)$ is a linear sketch, since it is a linear function of f. It can easily be computed in the cash register model, since each update to $f(i)$ requires adding an appropriate value to $h(f)$ based on computing α^i and multiplying this by the change in $f(i)$ modulo p.

The analysis of this procedure relies on the fact that a polynomial of degree d can have at most d roots (where it evaluates to zero). Testing whether two multisets D and D' are equal, based on the fingerprints of their corresponding frequency vectors, $h(f)$ and $h(f')$, is equivalent to testing the identity $h(f) - h(f') = 0$. Based on the definition of h, if the two multisets are identical then the fingerprints will be identical. But if they are different and the test still passes, the fingerprint will give the wrong answer. Treating $h()$ as a polynomial in α, $h(f) - h(f')$ has degree no more than M: so there can only be M values of α for which $h(f) - h(f') = 0$. Therefore, if p is chosen to be at least M/δ, the probability (based on choosing a random α) of making a mistake is at most δ, for a parameter δ. This requires the arithmetic operations to be done using $O(\log M + \log 1/\delta)$ bits of precision, which is feasible for most reasonable values of M and δ.

Such fingerprints have been used in streaming for a variety of purposes. For example, Yi *et al.* [294] employ fingerprints within a system to verify outsourced computations over streams.

5.2.3 Comparing Sketching with Sampling

These simple examples seem straightforward, but they serve to highlight the difference between the models of data access assumed by the sketching process. We have already seen that a small sample of the data can only estimate the average value in a data set, whereas this "sketch" can find it exactly. But this is due in part to a fundamental difference in assumptions about how the data is observed: the sample "sees" only those items which were selected to be in the sample whereas the sketch "sees" the entire input, but is restricted to retain only a small summary of it. Therefore, to build a sketch, we must either be able to perform a single linear scan of the input data (in no particular order), or to "snoop" on the entire stream of transactions which collectively build up the input. Note that many sketches were originally designed for computations in situations where the input is never collected together in one place (as in the financial data example), but exists only implicitly as defined by the stream of transactions.

Another way to understand the difference in power between the models of sampling and streaming is to observe that it is possible to design algorithms to draw a sample from a stream (see Section 2.7.4), but that there are queries

that can be approximated well by sketches that are provably impossible to compute from a sample. In particular, we will see a number of sketches to approximate the number of distinct items in a relation (Section 5.4), whereas no sampling scheme can give such a guarantee (Section 2.6.2). Similarly, we saw that fingerprinting can accurately determine whether two relations are identical, whereas unless every entry of two relations is sampled, it is possible that the two differ in the unsampled locations. Since the streaming model can simulate sampling, but sampling cannot simulate streaming, the streaming model is strictly more powerful in the context of taking a single pass through the data.

5.2.4 Properties of Sketches

Having seen these simple examples, we now formalize the main properties of a sketching algorithm.

- **Queries Supported.** Each sketch is defined to support a certain set of queries. Unlike samples, we cannot simply execute the query on the sketch. Instead, we need to perform a (possibly query specific) procedure on the sketch to obtain the (approximate) answer to a particular query.
- **Sketch Size.** In the above examples, the sketch is constant size. However, in the examples below, the sketch has one or more parameters which determine the size of the sketch. A common case is where parameters ε and δ are chosen by the user to determine the accuracy (approximation error) and probability of exceeding the accuracy bounds, respectively.
- **Update Speed.** When the sketch transform is very dense (i.e. the implicit matrix which multiplies the input has very few zero entries), each update affects all entries in the sketch, and so takes time linear in the sketch size. But typically the sketch transform can be made very sparse, and consequently the time per update may be much less than updating every entry in the sketch.
- **Query Time.** As noted, each sketch algorithm has its own procedure for using the sketch to approximately answer queries. The time to do this also varies from sketch to sketch: in some cases it

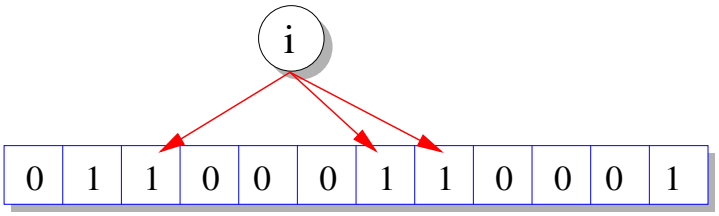


Fig. 5.2 Bloom Filter with $k = 3$, $m = 12$

is linear (or even superlinear) in the size of the sketch, whereas in other cases it can be much less.

- **Sketch Initialization.** By requiring the sketch to be a linear transformation of the input, sketch initialization is typically trivial: the sketch is initialized to the all-zeros vector, since the empty input is (implicitly) also a zero vector. However, if the sketch transform is defined in terms of hash functions, it may be necessary to initialize these hash functions by drawing them from an appropriate family.

5.2.5 Sketching Sets with Bloom Filters

As a more complex example, we briefly discuss the popular Bloom Filter as an example of a sketch. A Bloom filter, named for its inventor [24], is a compact way to represent a subset S of a domain $\mathscr{U}$. It consists of a binary string B of length $m < M$ initialized to all zeros, and k hash functions $h_1 \ldots h_k$, which each independently map elements of $\mathscr{U}$ to $\{1, 2, \ldots m\}$. For each element i in the set S, the sketch sets $B[h_j(i)] = 1$ for all $1 \leq j \leq k$. This is shown in Figure 5.2: an item i is mapped by $k = 3$ hash functions to a filter of size $m = 12$, and these entries are set to 1. Hence each update takes $O(k)$ time to process.

After processing the input, it is possible to test whether any given i is present in the set: if there is some j for which $B[h_j(i)] = 0$, then the item is not present, otherwise it is concluded that i is in S. From this description, it can be seen that the data structure guarantees no false negatives, but may report false positives.

Analysis of the Bloom Filter. The false positive rate can be analyzed as a function of $|S| = n$, m and k: given bounds on n and m, optimal values of k

can be set. We follow the outline of Broder and Mitzenmacher [26] to derive the relationship between these values. For the analysis, the hash functions are assumed to be fully random. That is, the location that an item is mapped to by any hash function is viewed as being uniformly random over the range of possibilities, and fully independent of the other hash functions. Consequently, the probability that any entry of B is zero after n distinct items have been seen is given by

$$p' = \left(1 - \frac{1}{m}\right)^{kn}$$

since each of the kn applications of a hash function has a $(1 - \frac{1}{m})$ probability of leaving the entry zero.

A false positive occurs when some item not in S hashes to locations in B which are all set to 1 by other items. This happens with probability $(1-\rho)^k$, where ρ denotes the fraction of bits in B that are set to 0. In expectation, ρ is equal to p', and it can be shown that ρ is very close to p' with high probability. Given fixed values of m and n, it is possible to optimize k, the number of hash functions. Small values of k keep the number of 1s lower, but make it easier to have a collision; larger values of k increase the density of 1s. The false positive rate is approximated well by

$$f = (1 - e^{-kn/m})^k = \exp(k \ln(1 - e^{kn/m}))$$

for all practical purposes. The smallest value of f as a function of k is given by minimizing the exponent. This in turn can be written as $-\frac{m}{n}\ln(p)\ln(1-p)$, for $p = e^{-kn/m}$, and so by symmetry, the smallest value occurs for $p = \frac{1}{2}$. Rearranging gives $k = (m/n)\ln 2$.

This has the effect of setting the occupancy of the filter to be 0.5, that is, half the bits are expected to be 0, and half 1. This causes the false positive rate to be $f = (1/2)^k = (0.6185)^{m/n}$. To make this probability at most a small constant, it is necessary to make $m > n$. Indeed, setting $m = cn$ gives the false positive probability at 0.6185^c: choosing $c = 9.6$, for example, makes this probability less than 1%.

Bloom Filter viewed as a sketch. In this form, we consider the Bloom filter to be a sketch, but it does not meet our stricter conditions for being considered as a linear sketch. In particular, the data structure is not a linear transform of the input: setting a bit to 1 is not a linear operation. We can

modify the data structure to make it linear, at the expense of increasing the space required. Instead of a bitmap, the Bloom filter is now represented by an array of counters. When adding an item, we increase the corresponding counters by 1, i.e. $B[h_j(i)] \leftarrow B[h_j(i)] + 1$. Now the transform is linear, and so it can process arbitrary streams of update transactions (including removals of items). The number of entries needed in the array remains the same, but now the entries are counters rather than bits.

One limitation when trying to use the Bloom Filter to describe truly large data sets is that the space needed is proportional to n, the number of items in the set S being represented. Within many approximate query processing scenarios, this much space may not be practical, so instead we look for more compact sketches. These smaller space sketches will naturally be less powerful than the Bloom filter: when using a datastructure that is sublinear in size (i.e. $o(n)$) we should not expect to be accurately answer all set-membership queries, even allowing for false positives and false negatives.

5.3 Frequency Based Sketches

In this Section, we present a selection of sketches which solve a variety of problems related to estimating functions of the frequencies, $f(i)$. Our presentation deliberately does not follow the chronological development of these sketches. Instead, we provide the historical context later in the section. We first define each sketch and the basic properties. In later sections, we study them in greater detail for approximate query answering.

5.3.1 Count-Min Sketch

The *Count-Min* sketch is so-called because of the two main operations used: counting of groups of items, and taking the minimum of various counts to produce an estimate [70]. It is most easily understood as keeping a compact array C of $d \times w$ counters, arranged as d rows of length w. For each row a *hash function* h_j maps the input domain $\mathscr{U} = \{1, 2, \ldots, M\}$ uniformly onto the range $\{1, 2, \ldots, w\}$. The sketch C is then formed as

$$C[j,k] = \sum_{1 \leq i \leq M : h_j(i) = k} f(i)$$

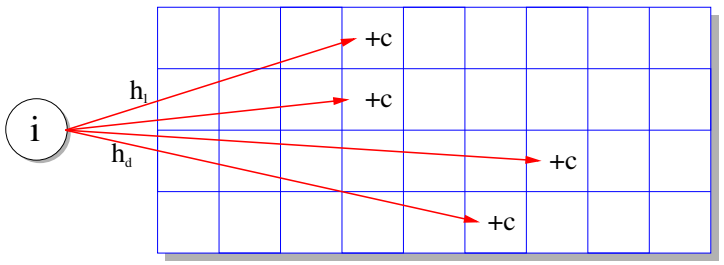


Fig. 5.3 Count-Min sketch data structure with $w = 9$ and $d = 4$

That is, the kth entry in the jth row is the sum of frequencies of all items i which are mapped by the jth hash function to value k. This leads to an efficient update algorithm: for each update to item i, for each $1 \leq j \leq d$, $h_j(i)$ is computed, and the update is added to entry $C[j,h_j(i)]$ in the sketch array. Processing each update therefore takes time $O(d)$, since each hash function evaluation takes constant time. Figure 5.3 shows this process: an item i is mapped to one entry in each row j by the hash function h_j, and the update of c is added to each entry.

The sketch can be used to estimate a variety of functions of the frequency vector. The primary function is to recover an estimate of $f(i)$, for any i. Observe that for it to be worth keeping a sketch in place of simply storing f exactly, it must be that wd is much smaller than M, and so the sketch will necessarily only approximate any $f(i)$. The estimation can be understood as follows: in the first row, it is the case that $C[1,h_1(i)]$ includes the current value of $f(i)$. However, since $w \ll M$, there will be many collisions under the hash function h_1, so that $C[1,h_1(i)]$ also contains the sum of all $f(i')$ for i' that collides with i under h_1. Still, if the sum of such $f(i')$s is not too large, then this will not be so far from $f(i)$.

In the strict case, all these $f(i')$s are non-negative, and so $C[1,h_1(i)]$ will be an overestimate for $f(i)$. The same is true for all the other rows: for each j, $C[j,h_j(i)]$ gives an overestimate of $f(i)$, based on a different set of colliding items. Now, if the hash functions are chosen at random, the items will be distributed uniformly over the row. So the expected amount of "noise" colliding with i in any given row is just $\sum_{1 \leq i' \leq M, i' \neq i} f(i')/w$, a $1/w$ fraction of the total count. Moreover, by the Markov inequality [233, 230], there is at least a 50% chance that the noise is less than twice this much. Here, the probabilities arise

due to the random choice of the hash functions. If each row's estimate of $f(i)$ is an overestimate, then the smallest of these will be the closest to $f(i)$. By the independence of the hash functions, it is now very unlikely that this estimate has error more than $2\sum_{1\leq i'\leq M} f(i')/w$: this only happens if *every* row estimate is "bad", which happens with probability at most 2^{-d}.

Rewriting this, if we pick $w = 2/\varepsilon$ and $d = \log 1/\delta$, then our estimate of $f(i)$ has error at most εN with probability at least $1-\delta$. Here, we write $N = \sum_{1\leq i'\leq M} f(i')$ as the sum of all frequencies—equivalently, the number of rows in the defining relation if we are tracking the cardinality of attribute values. The estimate is simply $\hat{f}(i) = \min_{j=1}^{d} C[j, h_j(i)]$. Producing the estimate is quite similar to the update procedure: the sketch is probed in one entry in each row (as in Figure 5.3). So the query time is the same as the update time, $O(d)$.

5.3.1.1 Perspectives on the Count-Min sketch

At its core, the Count-Min sketch is quite simple: just arrange the input items into groups, and compute the net frequency of the group. As such, we can think of it as a histogram with a twist: first, randomly permute the domain, then create an equi-width histogram with w buckets on this new domain. This is repeated for d random permutations. Query answering to estimate a single $f(i)$ is to find all the histogram buckets the item i is present in, and take the smallest of these. Viewed from another angle, the sketch can also be viewed as a small space, counting version of a Bloom filter [26, 57].

In this presentation, we omit detailed discussion of some of the technical issues surrounding the summary. For example, for the analysis, the hash functions are required to be drawn from a family of pairwise independent functions. However this turns out to be quite a weak condition: such functions are very simple to construct, and can be evaluated very quickly indeed [39, 280]. The estimator described is technically *biased*, in the statistical sense: it never underestimates but may overestimate, and so is not correct in expectation. However, it is straightforward to modify the estimator to be unbiased, by subtracting an appropriate quantity from the estimate. Heuristically, we can estimate the count of some "dummy" items such as $f(M+1)$ whose "true count" should be zero to estimate the error in the estimation [193]. We discuss the variant estimators in more detail in 5.3.5.3.

Lastly, the same sketch can also be used when the stream is general, and so can contain some items with negative frequencies. In this case, the sketch can be built in the same way, but now it is not correct to take the smallest row estimate as the overall estimate: this could be far from the true value if, for example, all the $f(i)$ values are negative. Instead, one can take the *median* of the row estimates, and apply the following general "Chernoff bounds argument".

Chernoff Bounds Argument. Suppose there are multiple independent copies of an estimator, each of which is a "good" estimate of a desired quantity with at least a constant probability (although it's not possible to tell whether or not an estimate is good just by looking at it). The goal is to combine these to make an estimate which is "good" with high probability. A standard technique is to take the *median* of enough estimates to reduce the error. Although it is not possible to determine which estimates are good or bad, sorting the estimates by value will place all the "good" estimates together in the middle, with "bad" estimates above and below (too low or too high). Then the only way that the median estimate can be bad is if more than half of the estimates are bad, which is unlikely. In fact, the probability of returning a bad estimate is now exponentially small in the number of estimates.

The proof makes use of a Chernoff bound. Assume that each estimate is good with probability at least $7/8$. The outcome of each estimate is an independent random event, so in expectation only $1/8$ of the estimates are bad. So the final result is only bad if the number of bad events exceeds it expectation by a factor of 4. Set the number of estimates to be $4\ln 1/\delta$ for some desired small probability δ. The Chernoff bound (slightly simplified) in this situation states that if X is the sum of independent Poisson trials, then for $0 < \rho \leq 4$,

$$\Pr[X > (1+\rho)\mathsf{E}[X]] < \exp(-\mathsf{E}[X]\rho^2/4).$$

See [233] for a derivation of this bound. Since each estimate is indeed an independent Poisson trial, then this setting is modeled with $\rho = 3$ and $\mathsf{E}[X] = \frac{1}{2}\ln 1/\delta$. Hence,

$$\Pr[X > 2\log 1/\delta] < \exp(-9/8\ln 1/\delta) < \delta$$

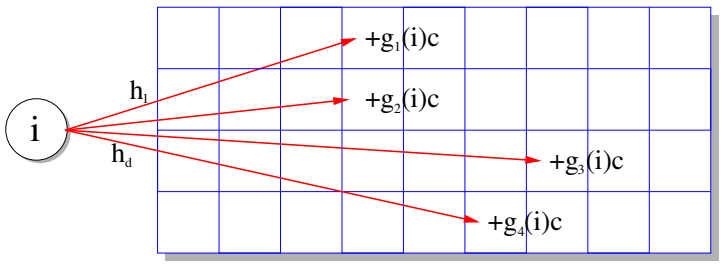


Fig. 5.4 Count Sketch data structure with $w = 9$ and $d = 4$

This implies that the taken the median of $O(\log 1/\delta)$ estimates reduces the probability of finding a bad final estimate to δ.

5.3.2 Count Sketch

The Count-Sketch [45] is similar to the Count-Min sketch, in that it can provide an estimate for the value of any individual frequency. The main difference is the nature of the accuracy guarantee provided for the estimate. Indeed, we can present the Count Sketch by using exactly the same sketch building procedure as the Count-Min sketch, so that the *only* difference is in the estimation procedure.

Now, given the Count-Min sketch data structure built on the stream, the row estimate of $f(i)$ for row j is computed based on two buckets: $h_j(i)$, the bucket which contains $f(i)$, and also

$$h'_j(i) = \begin{cases} h_j(i) - 1 & \text{if } h_j(i) \mod 2 = 0 \\ h_j(i) + 1 & \text{if } h_j(i) \mod 2 = 1 \end{cases}$$

which is an adjacent bucket [2]. So if $h_j(i) = 3$ then $h'_j(i) = 4$, while if $h_j(i) = 6$ then $h'_j(i) = 5$. The row estimate is then $C[j,h_j(i)] - C[j,h'_j(i)]$. Here, we assume that w, the length of each row in the sketch, is even.

The intuition for this estimate comes from considering all the "noise" items which collide with i: the distribution of such items in the $h_j(i)$th entry of row j should look about the same as the distribution in the $h'_j(i)$th entry, and so in expectation, these will cancel out, leaving only $f(i)$. More strongly,

[2] Equivalently, this can also be written as $[h'_j(i) = h_j(i) + (h_j(i) \mod 2) - (h_j(i) + 1 \mod 2)$

one can formally prove that this estimator is unbiased. Of course, the estimator still has variance, and so does not guarantee the correct answer. But this variance can be analyzed, and written in terms of the sum of squares of the items, $F_2 = \sum_{i=1}^{M} f(i)^2$. It can be shown that the variance of the estimator is bounded by $O(F_2/w)$. As a result, there is constant probability that each row estimate is within $\sqrt{F_2/w}$ of $f(i)$. Now by taking the median of the d row estimates, the probability of the final estimate being outside these bounds shrinks to $2^{-O(d)}$. Rewriting, if the parameters are picked as $d = O(\log 1/\delta)$ and $w = O(1/\varepsilon^2)$, the sketch guarantees to find an estimate of $f(i)$ so that the error is at most $\varepsilon\sqrt{F_2}$ with probability at least $1-\delta$.

In fact, this sketch can be compacted further. Observe that whenever a row estimate is produced, it is found as either $C[j,2k-1] - C[j,2k]$ or $C[j,2k] - C[j,2k-1]$ for some (integer) k. So rather than maintaining $C[j,2k-1]$ and $C[j,2k]$ separately, it suffices to keep this difference in a single counter. This can be seen by maintaining separate hash functions g_j which maps all items in $\mathscr{U} = \{1,2,\ldots,M\}$ uniformly onto $\{-1,+1\}$. Now the sketch is defined via

$$C[j,k] = \sum_{1 \le i \le M : h_j(i) = k} g_j(i) f(i).$$

This meets the requirements for a linear sketch, since it is a linear function of the f vector. It can also be computed in time $O(d)$: for each update to item i, for each $1 \le j \le d$, $h_j(i)$ is computed, and the update multiplied by $g_j(i)$ is added to entry $C[j,h_j(i)]$ in the sketch array. The row estimate for $f(i)$ is now $g_j(i) * C[j,h_j(i)]$. It can be seen that this version of the sketch is essentially equivalent to the version described above, and indeed all the properties of the sketch are the same, except that w can be half as small as before to obtain the same accuracy bounds. This is the version that was originally proposed in the 2002 paper [45]. An example is shown in Figure 5.4: an update is mapped into one entry in each row by the relevant hash function, and multiplied by a second hash function g. The figure serves to emphasize the similarities between the Count Sketch and Count-Min sketch: the main difference arises in the use of the g_j functions.

5.3.2.1 Refining Count-Sketch and Count-Min Sketch guarantees

We have seen that the Count-Sketch and Count-Min sketch both allow $f(i)$ to be approximated via somewhat similar data structures. They differ in providing distinct space/accuracy trade-offs: the Count sketch gives $\varepsilon\sqrt{F_2}$ error with $O(1/\varepsilon^2)$ space, whereas the Count-Min sketch gives εN error with $O(1/\varepsilon)$ space. In general, these bounds cannot be compared: there exist some frequency vectors where (given the same overall space budget) one guarantee is preferable, and others where the other dominates. Indeed, various experimental studies have shown that over real data it is not always clear which is preferable [65].

However, a common observation from empirical studies is that these sketches give better performance than their worst case guarantees would suggest. This can be explained in part by a more rigorous analysis. Most frequency distributions seen in practice are *skewed*: there are a few items with high frequencies, while most have low frequencies. This phenomenon is known by several names, such as Pareto, Zipfian, and long tailed distributions. For both Count Sketch and Count-Min sketch, a skewed distribution *helps* estimation: when estimating a frequency $f(i)$, it is somewhat unlikely that any of the few high frequency items will collide with i under h_j —certainly it is very unlikely that any will collide with i in a majority of rows. So it is possible to separate out some number k of the most frequent items, and separately analyze the probability that i collides with them. The probability that these items affect the estimation of $f(i)$ can then be bounded by δ. This leaves only a "tail" of lower frequency items, which still collide with i with uniform probability, but the net effect of this is lower since there is less "mass" in the tail. Formally, let f_i denote the ith largest frequency in f, and let

$$F_1^{\mathrm{res}(k)} = \sum_{i=k+1}^{M} f_i \qquad \text{and} \qquad F_2^{\mathrm{res}(k)} = \sum_{i=k+1}^{M} f_i^2$$

denote the sum and the sum of squares of all but the k largest frequencies. As a result, we can bound the error in the sketch estimates in terms of $F_1^{\mathrm{res}(k)}/w$ and $\sqrt{F_2^{\mathrm{res}(k)}/w}$ for the Count-Min and Count sketch respectively, where $k = O(w)$ [45, 71].

5.3.3 The AMS Sketch

The AMS sketch was first presented in the work of Alon, Matias and Szegedy [7]. It was proposed to solve a different problem, to estimate the value of F_2 of the frequency vector, the sum of the squares of the frequencies. Although this is straightforward if each frequency is presented in turn, it becomes more complex when the frequencies are presented implicitly, such as when the frequency of an item is the number of times it occurs within a long, unordered, stream of items. Estimating F_2 may seem like a somewhat obscure goal in the context of approximate query processing. However, it has a surprising number of applications. Most directly, F_2 equates to the self-join size of the relation whose frequency distribution on the join attribute is f (for an equi-join). The AMS sketch turns out to be highly flexible, and is at the heart of estimation techniques for a variety of other problems which are all of direct relevance to AQP.

5.3.3.1 AMS Sketch for Estimating F_2

We again revert to the sketch data structure of the Count-Min sketch as the basis of the AMS sketch, to emphasize the relatedness of all these sketch techniques. Now each row is used in its entirety to make a row estimate of F_2 as

$$\sum_{k=1}^{w/2} (C[j,2k-1] - C[j,2k])^2.$$

Expanding out this expression in terms of $f(i)$s, it is clear that the resulting expression includes $\sum_{i=1}^{M} f(i)^2 = F_2$. However, there are also a lot of cross terms of the form $\pm 2f(i)f(i')$ for $i \neq i'$ such that either $h_j(i) = h_j(i')$ or $|h_j(i) - h_j(i')| = 1$. That is, we have errors due to cross terms of frequencies of items placed in the same location or adjacent locations by h_j. Perhaps surprisingly, the expected contribution of these cross terms is zero. There are three cases to consider for each i and i' pair: (a) they are placed in the same entry of the sketch, in which case they contribute $2f(i)f(i')$ to the estimate; (b) one is placed in the $2k$th entry and the other in the $2k-1$th, in which case they contribute $-2f(i)f(i')$ to the estimate; or (c) they are not placed in adjacent entries and so contribute 0 to the estimate. Due to the uniformity of h_j, cases (a) and (b) are equally likely, so in expectation (over the choice of

h_j) the *expected* contribution to the error is zero.

Of course, in any given instance there are some i, i' pairs which contribute to the error in the estimate. However, this can be bounded by studying the variance of the estimator. This is largely an exercise in algebra and applying some inequalities (see [281, 7] for some details). The result is that the variance of the row estimator can be bounded in terms of $O(F_2^2/w)$. Consequently, using the Chebyshev inequality [233], the error is at most $F_2/\sqrt{w}$ with constant probability. Taking the median of the d rows reduces the probability of giving a bad estimate to $2^{-O(d)}$, by the Chernoff bounds argument outlined above.

As in the Count-Sketch case, this sketch can be "compacted" by observing that it is possible to directly maintain $C[j,2k-1]-C[j,2k]$ in a single entry, by introducing a second hash function g_j which maps $\mathscr{U}$ uniformly onto $\{-1,+1\}$. Technically, a slightly stronger guarantee is needed on g_j: because the analysis studies the variance of the row estimator, which is based on the expectation of the square of the estimate, the analysis involves looking at products of the frequencies of four items and their corresponding g_j values. To bound this requires g_j to appear independent when considering sets of four items together: this adds the requirement that g_j be *four-wise* independent. This condition is slightly more stringent than the pairwise independence needed of h_j.

Practical Considerations for Hashing. Although the terminology of pairwise and four-wise independent hash functions may be unfamiliar, they should not be thought of as exotic or expensive. A family of pairwise independent hash functions is given by the functions $h(x) = ax+b \mod p$ for constants a and b chosen uniformly between 0 and $p-1$, where p is a prime. Over the random choice of a and b, the probability that two items collide under the hash function is $1/p$. Similarly, a family of four-wise independent hash functions is given by $h(x) = ax^3+bx^2+cx+d \mod p$ for a,b,c,d chosen uniformly from $[p]$ with p prime. As such, these hash functions can be computed very quickly, faster even than more familiar (cryptographic) hash functions such as MD5 or SHA-1. For scenarios which require very high throughput, Thorup has studied how to make very efficient implementations of such hash functions, based on optimizations for particular values of p, and partial precomputations [280, 281].

Consequently, this sketch can be very fast to compute: each update requires only d entries in the sketch to be visited, and a constant amount of hashing work done to apply the update to each visited entry. The depth d is set as $O(\log 1/\delta)$, and in practice this is of the order of 10-30, although empirically d can be set as low as 3 or 4 without any obvious problem [65].

AMS sketch with Averaging versus Hashing. In fact, the original description of the AMS sketch gave an algorithm the was considerably slower: the original AMS sketch was essentially equivalent to the sketch we have described with $w = 1$ and $d = O(\varepsilon^{-2} \log 1/\delta)$. Then the mean of $O(\varepsilon^{-2})$ entries of the sketch was taken, to reduce the variance, and the final estimator found as the median of $O(\log 1/\delta)$ such independent estimates. We refer to this as the "averaging version" of the AMS sketch. This estimator has the same space cost as the version we present here, which was the main objective of [7]. The faster version, based on the "hashing trick" is sometimes referred to as "fast AMS" to distinguish it from the original sketch, since each update is dramatically faster.

Historical Notes. Historically, the AMS Sketch [7] was the first to be proposed as such in the literature, in 1996. The "Random Subset Sums" technique can be shown to be equivalent to the AMS sketch for estimating single frequencies [130]. The Count-Sketch idea was presented first in 2002. Crucially, this seems to be the first work where it was shown that hashing items to w buckets could be used instead of taking the mean of w repetitions of a single estimator, and that this obtains the same accuracy. Drawing on this, the Count-Min sketch was the first to obtain a guarantee in $O(1/\varepsilon)$ space, albeit for an F_1 instead of F_2 guarantee [70]. Applying the "hashing trick" to the AMS sketch make it very fast to update seems to have been discovered in parallel by several people. Thorup and Zhang were among the first to publish this idea [281], and an extension to inner-products appeared in [63].

5.3.4 Approximate Query Processing with Frequency Based Sketches

Now that we have seen the definitions and basic properties of the Count-Min Sketch, Count-Sketch and AMS Sketch, we go on to see how they can be applied to approximate the results of various aggregation queries.

5.3.4.1 Point Queries and Heavy Hitters

Via Count-Min and Count Sketch, we have two mechanisms to approximate the frequency of any given item i. One has accuracy proportional to εN, the other proportional to $\varepsilon\sqrt{F_2}$. To apply either of these, we need to have in mind some particular item i which is of interest. A more common situation arises when we have no detailed *a priori* knowledge of the frequency distribution, and we wish to find which are the most significant items. Typically, those most significant items are those which have high frequencies – the so-called "heavy hitters". More formally, we define the set of heavy hitters as those items whose frequency exceeds a ϕ fraction of the total frequency, for some chosen $0 < \phi < 1$.

The naive way to discover the heavy hitters within a relation is to exhaustively query each possible i in turn. The accuracy guarantees indicate that the sketch should correctly recover those items with $f(i) > \varepsilon N$ or $f(i) > \varepsilon\sqrt{F_2}$. But this procedure can be costly, or impractical, when the domain size M is large—consider searching a space indexed by a 32 or 64 bit integer. The number of queries is so high that there may be false positives unless d is chosen to be sufficiently large to ensure that the overall false positive probability is driven low enough.

In the cash-register streaming model, where the frequencies only increase, a simple solution is to combine update with search. Note that an item can only become a heavy hitter in this model following an arrival of that item. So the current set of heavy hitters can be tracked in a data structure separate to the sketch, such as a heap or list sorted by the estimated frequency [45]. When the frequency of an item increases, at the same time the sketch can be queried to obtain the current estimated frequency. If the item exceeds the current threshold for being a heavy hitter, it can be added to the data structure. At any time, the current set of (approximate) heavy hitters can be found by probing this data structure.

We can compare this to results from sampling: standard sampling results argue that to find the heavy hitters with εN accuracy, a sample of size $O(1/\varepsilon^2)$ items is needed. So the benefits of the Count-Min sketch are clear: the space required is quadratically smaller to give the same guarantee ($O(1/\varepsilon)$ compared to $O(1/\varepsilon^2)$). However, the benefits become more clear in the turnstile case when, if there is significant numbers of deletions in the data, causing the

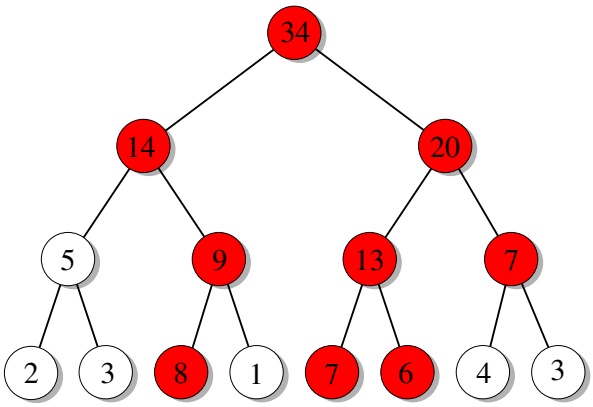


Fig. 5.5 Searching for heavy hitters via binary-tree search

set of Heavy Hitters to change considerably over time. In such cases it is not possible to draw a large sample from the input stream.

Heavy Hitters over Strict Distributions. When the data arrives in the turnstile model, decreases in the frequency of one item can cause another item to become a heavy-hitter implicitly—because its frequency now exceeds the (reduced) threshold. So the method of keeping track of the current heavy hitters as in the cash-register case will no longer work.

It can be more effective to keep a more complex sketch, based on multiple instances of the original sketch over different views of the data, to allow more efficient retrieval of the heavy hitters. The "divide and conquer" or "hierarchical search" technique conceptually places a fixed tree structure, such as a binary tree structure, over the domain. Each internal node is considered to correspond to the set of items covered by leaves in the induced subtree. Each internal node is treated as a new item, whose frequency is equal to the sum of the frequencies of the items associated with that node. In addition to a sketch of the leaf data, a sketch of each level of the tree (the frequency distribution of each collection of nodes at the same distance from the root) is kept.

Over frequency data in the strict model, it is the case that each ancestor of a heavy hitter leaf must be at least as frequent, and so must appear as a heavy hitter also. This implies a simple divide-and-conquer approach to finding the heavy hitter items: starting at the root, the appropriate sketch is used to es-

timate the frequencies of all children of each current "candidate" node. All nodes whose estimated frequency makes them heavy hitters are added to the list of candidates, and the search continues down the tree. Eventually, the leaf level will be reached, and the heavy hitters should be discovered. Figure 5.5 illustrates the process: given a frequency distribution at the leaf level, a binary tree is imposed, where the frequency of the internal nodes is the sum of the leaf frequencies in the subtree. Nodes which are heavy (in this example, whose frequency exceeds 6) are shaded. All unshaded nodes can be ignored in the search for the heavy hitters.

Using a tree structure with a constant fan-out at each node, there are $O(\log M)$ levels to traverse. Defining a heavy hitter as an item whose count exceeds ϕN for some fraction ϕ, there are at most $1/\phi$ possible (true) heavy hitters at each level. So the amount of work to discover the heavy hitters at the leaves is bounded by $O(\log M/\phi)$ queries, assuming not too many false positives along the way. This analysis works directly for the Count-Min sketch. However, it does not quite work for finding the heavy hitters based on Count-Sketch and an F_2 threshold, since the F_2 of higher levels can be much higher than at the leaf level.

Nevertheless, it seems to be an effective procedure in practice. A detailed comparison of different methods for finding heavy hitters is performed in [65]. There, it is observed that there is no clear "best" sketch for this problem: both approaches have similar accuracy, given the same space. As such, it seems that the Count-Min approach might be slightly preferred, due to its faster update time: the Count-Min sketch processed about 2 million updates per second in speed trials, compared to 1 million updates per second for the Count Sketch. This is primarily since Count-Min requires only one hash function evaluation per row, to the Count Sketch's two [3].

Heavy Hitters over General Distributions. For general distributions which include negative frequencies, this procedure is not guaranteed to work: consider two items, one with a large positive frequency and the other with a large negative frequency of similar magnitude. If these fall under the same node in the tree, their frequencies effectively cancel each other, and the search

[3] To address this, it would be possible to use a single hash function, where the last bit determines g_j and the preceding bits determine h_j

procedure may mistakenly fail to discover them. General distributions can arise for a variety of reasons, for example in searching for items which have significantly different frequencies in two different relations, so it is of interest to overcome this problem. Henzinger posed this question in the context of identifying which terms were experiencing significant change in their popularity within an Internet search engine [166]. Consequently, this problem is sometimes also referred to as the "heavy changers" problem.

In this context, various "group testing" sketches have been proposed. They can be thought of as turning the above idea inside out: instead of using sketches inside a hierarchical search structure, the group testing places the hierarchical search structure inside the sketch. That is, it builds a sketch as usual, but for each entry in the sketch keeps $O(\log M)$ additional information based, for instance, on the binary expansion of the item identifiers. The idea is that the width of the sketch, w, is chosen so that in expectation at most one of the heavy hitters will land in each entry, and the sum of (absolute) frequencies from all other items in the same entry is small in comparison to the frequency of the heavy hitter. Then, the additional information is in the form of a series of "tests" designed to allow the identity of the heavy hitter item to be recovered. For example, one test may keep the sum of frequencies of all items (within the given entry) whose item identifier is odd, and another keeps the sum of all those whose identifier is even. By comparing these two sums it is possible to determine whether the heavy hitter item identifier is odd or even, or if more than one heavy hitter is present, based on whether one or both counts are heavy. By repeating this with a test for each bit position (so $O(\log M)$ in total), it is possible to recover enough information about the item to correctly identify it. The net result is that it is possible to identify heavy hitters over general streams with respect to N or F_2 [69]. Other work in this area has considered trading off more space and greater search times for higher throughput [269]. Recent work has experimented with the hierarchical approach for finding heavy hitters over general distributions, and shown that on realistic data, it is still possible to recover most heavy hitters in this way [31].

5.3.4.2 Join Size Estimation

Given two frequency distributions over the same domain, f and f', their inner product is defined as

$$f \cdot f' = \sum_{i=1}^{M} f(i) * f'(i)$$

This has a natural interpretation, as the size of the equi-join between relations where f denotes the frequency distribution of the join attribute in the first, and f' denote the corresponding distribution in the second. In SQL, this is

```
SELECT COUNT(*) FROM D, D'
WHERE D.id = D'.id
```

The inner product also has a number of other fundamental interpretations that we shall discuss below. For example, it also can be used when each record has an additional "measure" value, and the query is to compute the sum of products of measure values of joining tuples (e.g. finding the total amount of sales given by number of sales of each item multiplied by price of each item). It is also possible to encode the sum of those $f(i)$s where i meets a certain predicate as an inner product where $f'(i) = 1$ if and only if i meets the predicate, and 0 otherwise. This is discussed in more detail below.

Using the AMS Sketch to estimate inner products. Given AMS sketches of f and f', C and C' respectively, that have been constructed with the same parameters (that is, the same choices of w, d, h_j and g_j), the estimate of the inner product is given by

$$\sum_{k=1}^{w} C[j,k] * C'[j,k].$$

That is, the row estimate is the inner product of the rows.

The bounds on the error follow for similar reasons to the F_2 case (indeed, the F_2 case can be thought of as a special case where $f = f'$). Expanding out the sum shows that the estimate gives $f \cdot f'$, with additional cross-terms due to collisions of items under h_j. The expectation of these cross terms in $f(i)f'(i')$ is zero over the choice of the hash functions, as the function g_j is equally likely to add as to subtract any given term. The variance of the

row estimate is bounded via the expectation of the square of the estimate, which depends on $O(F_2(f)F_2(f')/w)$. Thus each row estimate is accurate with constant probability, which is amplified by taking the median of d row estimates.

Comparing this guarantee to that for F_2 estimation, we note that the error is bounded in terms of the product $\sqrt{F_2(f)F_2(f')}$. In general, $\sqrt{F_2(f)F_2(f')}$ can be much larger than $f \cdot f'$, such as when each of f and f' is large but $f \cdot f'$ can still be small or even zero. However, this is unavoidable: lower bounds show that no sketch (more strongly, no small data structure) can guarantee to estimate $f \cdot f'$ with error proportional to $f \cdot f'$ unless the space used is at least M [211].

In fact, we can see this inner product estimation as a generalization of the previous methods. Set $f'(i) = 1$ and $f'(i') = 0$ for all $i' \neq i$. Then $f \cdot f' = f(i)$, so computing the estimate of $f \cdot f'$ should approximate the single frequency $f(i)$ with error proportional to $\sqrt{F_2(f)F_2(f')/w} = \sqrt{F_2(f)/w}$. In retrospect this is not surprising: when we consider building the sketch of the constructed frequency distribution f' and making the estimate, the resulting procedure is identical to the procedure of estimating $f(i)$ via the Count-Sketch approach. Using the AMS sketch to estimate inner-products was first proposed in [6]; the "fast" version was described in [63].

Using the Count-Min sketch to estimate inner products. The Count-Min sketch can be used in a similar way to estimate $f \cdot f'$. In fact, the row estimate is formed the same way as the AMS estimate, as the inner product of sketch rows:

$$\sum_{k=1}^{w} C[j,k] * C'[j,k].$$

Expanding this sum based on the definition of the sketch results in exactly $f \cdot f'$, along with additional error terms of the form $f(i)f'(i')$ from items i and i' which happen to be hashed to the same entry by h_j. The expectation of these terms is not too large, which is proved by a similar argument to that used to analyze the error in making point estimates of $f(i)$. We expect about a $1/w$ fraction of all such terms to occur over the whole summation. So with constant probability, the total error in a row estimate is NN'/w (where $N = \sum_{i=1}^{M} f(i)$ and $N' = \sum_{i=1}^{M} f'(i)$). Repeating and taking the minimum of d

estimates makes the probability of a large error exponentially small in d.

Applying this result to $f = f'$ shows that Count-Min sketch can estimate $f \cdot f = F_2$ with error N^2/w. In general, this will be much larger than the corresponding AMS estimate with the same w, unless the distribution is skewed. For sufficiently skewed distribution, a more careful analysis (separating out the k largest frequencies) gives tighter bounds for the accuracy of F_2 estimation [71]. Setting f' to pick out a single frequency $f(i)$ has the same bounds as the point-estimation case. This is to be expected, since the resulting procedure is identical to the point estimation protocol.

Comparing AMS and Count-Min sketches for join size estimation. The analysis shows the worst case performance of the two sketch methods can be bounded in terms of N or F_2. To get a better understanding of their true performance, Dobra and Rusu performed a detailed study of sketch algorithms [89]. They gave a careful statistical analysis of the properties of sketches, and considered a variety of different methods to extract estimates from sketches. From their empirical evaluation of many sketch variations for the purpose of join-size estimation across a variety of data sets, they arrive at the following conclusions:

- The errors from the hashing and averaging variations of the AMS sketches are comparable for low-skew (near-uniform) data, but are dramatically lower for the hashing version (the main version presented in this chapter) when the skew is high.
- The Count-Min sketch does not perform well when the data has low-skew, due to the impact of collisions with many items on the estimation. But it has the best overall performance when the data is skewed, since the errors in the estimation are relatively much lower.
- The sketches can be implemented to be very efficient: each update to the sketch in their experiments took between 50 and 400 nanoseconds, translating to a processing rate of millions of updates per second.

As a result, the general message seems to be that the (fast) AMS version of the sketches are to be preferred for this kind of estimation, since they exhibit

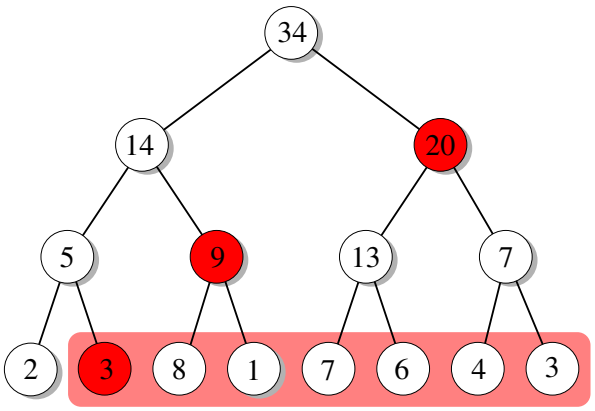


Fig. 5.6 Using dyadic ranges to answer a range query

fast updates and accurate estimations across the broadest range of data types.

5.3.4.3 Range Queries and Quantiles

Another common type of query is a range-count , e.g.

```
SELECT COUNT(*) FROM D
WHERE D.val >= l AND D.val <= h
```

for a range $l \ldots h$. Note that these are the exactly the same as the range-count and range-sum queries considered over histogram representations (defined in Section 3.1.1).

A direct solution to this query is to pose an appropriate inner-product query. Given a range, it is possible to construct a frequency distribution f' so that $f'(i) = 1$ if $l \leq i \leq h$, and 0 otherwise. Then $f \cdot f'$ gives the desired range-count. However, when applying the AMS approach to this, the error scales proportional to $\sqrt{F_2(f)F_2(f')}$. So here the error grows proportional to the square root of the length of the range. Using the Count-Min sketch approach, the error is proportional to $N(h-l+1)$, i.e. it grows proportional to the length of the range, clearly a problem for even moderately sized ranges. Indeed, this is the same error behavior that would result from estimating each frequency in the range in turn, and summing the estimates.

Range Queries via Dyadic Ranges. The hierarchical approach, outlined in Section 5.3.4.1, can be applied here. A standard technique which appears in many places within streaming algorithms is to represent any range canonically as a logarithmic number of so-called *dyadic* ranges. A dyadic range is a range whose length power of two, and which begins at a multiple of its own length (the same concept is used in the Haar Wavelet Transform, see Section 4.2.1). That is, it can be written as $[j2^a + 1 \ldots (j+1)2^a]$. Examples of dyadic ranges include $[1 \ldots 8]$, $[13 \ldots 16]$, $[5 \ldots 6]$ and $[27 \ldots 27]$. Any arbitrary range can be canonically partitioned into dyadic ranges with a simple procedure: greedily find the longest possible dyadic range from the start of the range, and repeat on what remains. So for example, the range $[18 \ldots 38]$ can be broken into the dyadic ranges

$$[18 \ldots 18], [19 \ldots 20], [21 \ldots 24], [25 \ldots 32], [33 \ldots 36], [37 \ldots 38]$$

Note that there are at most two dyadic ranges of any given length in the canonical decomposition.

Therefore, a range query can be broken up into $O(\log M)$ pieces, and each of these can be posed to an appropriate sketch over the hierarchy of dyadic ranges. A simple example is shown in Figure 5.6. To estimate the range sum of $[2 \ldots 8]$, it is decomposed into the ranges $[2 \ldots 2], [3 \ldots 4], [5 \ldots 8]$, and the sum of the corresponding nodes in the binary tree is found as the estimate. So the range sum is correctly found as 32 (here, we use exact values). When using the Count-Min sketch to approximate counts, the result is immediate: the accuracy of the answer is proportional to $(N \log M)/w$: a clear advantage over the previous accuracy of $N(h-l)/w$. For large enough ranges, this is an exponential improvement in the error.

In the AMS/Count-sketch case, the benefit is less precise: each dyadic range is estimated with error proportional to the square root of the sum of the frequencies in the range. For large ranges, this error can be quite large. However, there are relatively few really large dyadic ranges, so a natural solution is to maintain the sums of frequencies in these ranges exactly [70, 89]. With this modification, it has been shown that the empirical behavior of this technique is quite accurate [89].

Quantiles via Range Queries. The quantiles of a frequency distribution on an ordered domain divide the total "mass" of the distribution into equal

parts. More formally, the ϕ quantile q of a distribution is that point such that $\sum_{i=1}^{q} f(i) = \phi N$. The most commonly used quantile is the median, which corresponds to the $\phi = 0.5$ point. Other quantiles describe the shape of the distribution, and the likelihood that items will be seen in the "tails" of the distribution. They are also commonly used within database systems for approximate query answering, and within simple equidepth histograms Section 3.1.1).

There has been great interest in finding the quantiles of distributions defined by streams – see the work of Greenwald and Khanna, and references therein [132]. Gilbert *et al.* were the first to use sketches to track the quantiles of streams in the turnstile model [130]. Their solution is to observe that finding a quantile is the dual problem to a range query: we are searching for a point q so that the range query on $[1 \ldots q]$ gives ϕN. Since the result of this range query is monotone in q, we can perform a binary search for q. Each queried range can be answered using the above techniques for approximately answering range queries via sketches, and in total $O(\log M)$ queries will be needed to find a q which is (approximately) the ϕ-quantile.

Note here that the error guarantee means that we will guarantee to find a "near" quantile. That is, the q which is found is not necessarily an item which was present in the input—recall, in Section 5.2.5 we observed that whenever we store a data structure that is smaller than the size of the input, there is not room to recall which items were or were not present in the original data. Instead, we guarantee that the range query $[1 \ldots q]$ is approximately ϕN, where the quality of the approximation will depend on the size of the sketch used. Typically, the accuracy is proportional to εN, so when $\varepsilon \ll \phi$, the returned point will close to the desired quantile. It also means that extreme quantiles (like the 0.001 or the 0.999 quantile) will not be found very accurately. The most extreme quantiles are the maximum and minimum values in the data set, and we have already noted that such values are not well suited for linear sketches to find.

5.3.4.4 Sketches for Measuring Differences

A *difference query* is used to measure the difference between two frequency distributions. The *Euclidean difference* treats the frequency distributions as vectors, and measures the Euclidean distance between them. That is, given

two frequency distributions, it computes

$$\sqrt{F_2(f-f')} = \sqrt{\sum_{i=1}^{M}(f(i)-f'(i))^2}$$

Note that this is quite similar in form to an F_2 calculation, except that this is being applied to the *difference* of two frequency distributions. However, we can think of this as applying to an *implicit* frequency distribution, where the frequency of i is given by $(f(i)-f'(i))$. This can be negative, if $f'(i) > f(i)$. Here, the flexibility of sketches which can process general streams comes to the fore: it does not matter that there are negative frequencies. Further, it is not necessary to directly generate the difference distribution. Instead, given a sketch of f as C and a sketch of f' as C', it is possible to generate the sketch of $(f-f')$ as the array subtraction $(C-C')$. This is correct, due to the linearity properties of sketches. Therefore, from the two sketches an approximation of $\sqrt{F_2(f-f')}$ can be immediately computed. The accuracy of this approximation varies with $\sqrt{F_2(f-f')}/\sqrt{w}$. This is a powerful guarantee: even if $F_2(f-f')$ is very small compared to $F_2(f)$ and $F_2(f')$, the sketch approach will give a very accurate approximation of the difference.

More generally, arbitrary arithmetic over sketches is possible: the F_2 of sums and differences of frequency distributions can be found by performing the corresponding operations on the sketch representations. Given a large collection of frequency distributions, the Euclidean distance between any pair can be approximated using only the sketches, allowing them to be clustered or otherwise compared. The mathematically inclined can view this as an efficient realization of the Johnson-Lindenstrauss Lemma [195].

5.3.5 Advanced Uses of Sketches

In this section we discuss how the sketches already seen can be applied to higher dimensional data, more complex types of join size estimation, and alternate estimation techniques.

5.3.5.1 Higher Dimensional Data

Sketches can naturally summarize higher dimensional data. Given a multidimensional frequency distribution such as $f(i_1,i_2,i_3)$, it is straightforward for most of the sketches to summarize this: we just hash on the index (i_1,i_2,i_3)

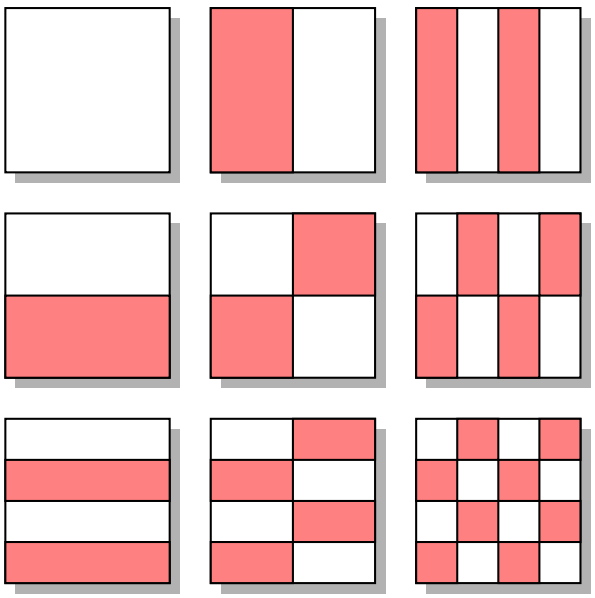


Fig. 5.7 Dyadic decomposition of a 4 by 4 rectangle

where needed. This lets us compute, for example, point queries, join size queries and distinct counts over the distributions. But these queries are not really fundamentally different for multi-dimensional data compared to single dimensional data. Indeed, all these results can be seen by considering applying some linearization to injectively map the multidimensional indices to a single dimension, and solving the one-dimensional problem on the resulting data.

Things are more challenging when we move to range queries. Now the query is specified by a product of ranges in each dimension, which specifies a hyper-rectangle. With care, the dyadic range decomposition technique can be lifted to multiple dimensions. In a single dimension, we argued that any range could be decomposed into $O(\log M)$ dyadic ranges. Analogously, any ℓ dimensional range over $\{1,\ldots,M\}^\ell$ can be decomposed into $O(\log^\ell M)$ dyadic hyper-rectangles: rectangles formed as the product of dyadic ranges. Figure 5.7 shows the decompositions of a small two dimensional rectangle of dimensions 4 by 4. Each of the nine subfigures shows a different combination of dyadic ranges on the x and y axes.

Applying this approach, we quickly run into a "curse of dimensionality" problem: there are $\log^{\ell} M$ different types of hyper-rectangle to keep track of, so the space required (along with the time to process each update and each query) increases by at least this factor. So for range queries using Count-Min sketches, for example, the space cost to offer εN accuracy grows in proportion to $(\log M)^{2\ell}/\varepsilon$.[4] For ℓ more than 2 or 3, this cost can be sufficiently large to be impractical. Meanwhile, a sample of $O(1/\varepsilon^2)$ items from the distribution is enough to estimate the selectivity of the range with accuracy ε, and hence the size of the range sum with accuracy εN, irrespective of the dimensionality. Therefore, if it is possible to maintain a sample, this will often be preferable for higher dimensional data.

Alternatively, we can make independence assumptions, as discussed in the histogram case (Section 3.5.2): if we believe that there are no correlations between the dimensions, we can keep a sketch of each dimension independently, and estimate the selectivity over the range as the product of the estimated selectivities. However, this does not seem representative of real data, where we expect to see many correlated dimensions. A compromise is to decompose the dimensions into pairs which are believed to be most correlated, and sketch the pairwise distribution of such pairs. Then the product of selectivities on each correlated pair can estimate the overall selectivity, and hence the range sum.

The work of Thaper *et al.* [278] uses multidimensional sketches to derive approximate histogram representations. Given a proposed bucketing (set of hyper-rectangles and weights), it computes the error by measuring the difference between a sketch of the data and a sketch of the histogram. The algorithm can then search over bucketings to find the best (according to the approximations from sketching). Various methods are applied to speed up the search. Considering rectangles in a particular order means that sketches can be computed incrementally; [278] also suggests the heuristic of only considering buckets that are dyadic hyper rectangles. Empirical study shows that the method finds reasonable histograms, but the time cost of the search increases dramatically as the domain size increases, even on two-dimensional data.

[4] This exponent is 2ℓ, because we need to store $(\log M/)^{\ell}$ sketches, and each sketch needs to be created with parameter w proportional to $(\log M)^{\ell}/\varepsilon$ so that the overall accuracy of the range query is εN.

5.3.5.2 More complex join size estimation

Multi-way join size estimation. Dobra *et al.* propose a technique to extend the join-size estimation results to multiple join conditions [88]. The method applies to joins of r relations, where the join condition is of the form

```
WHERE R1.A1 = R2.A1 AND R2.A2 = R3.A2 AND ...
```

The main idea is to take the averaging-based version of the AMS sketch (where all items are placed into every sketch entry). A sketch is built for each relation, where each update is multiplied by r independent hash functions that map onto $\{-1,+1\}$. Each hash function corresponds to a join condition, and has the property that if a pair of tuples join under that condition then they both hash to the same value (this is a generalization of the technique used in the sketches for approximating the size of a single join). Otherwise, there is no correlation between their hash values. A single estimate is computed by taking the product of the same entry in each sketch. Putting this all together, it follows that sets of r tuples which match on all the join conditions get their frequencies multiplied by 1, whereas all other sets of tuples contribute zero in expectation. Provided the join graph is acyclic, the variance of the estimation grows as $2^a \prod_{j=1}^{r} F_2(R_j)$, where a is the number of join attributes, and $F_2(R_j)$ denotes the F_2 (self-join size) of relation R_j. This gives good results for small numbers of joins.

A disadvantage of the technique is that is uses the slower averaging version of the AMS sketch: each update affects each entry of each sketch. To apply this using the hashing trick, we need to ensure that every pair of matching tuples get hashed into the same entry. This seems difficult for general join conditions, but is achievable in a multi-way join over the same attribute, i.e. a condition of the form

```
WHERE R1.A = R2.A AND R2.A = R3.A AND R3.A = R4.A
```

Now the sketch is formed by hashing each tuple into a sketch based on its A value, and multiplying by up to two other hash functions that map to $\{-1,+1\}$ to detect when the tuple from R_j has joining neighbors from R_{j-1} and R_{j+1}.

Note that such sketches are valuable for estimating join sizes with additional selection conditions on the join attribute: we have a join between R_1

and R_2 on A, and R_3 encodes an additional predicate on A which can be specified at query time. This in turn captures a variety of other settings: estimating the inner product between a pair of ranges, for example.

Non-equi joins. Most uses of sketches have focused on equi-joins and their direct variations. There has been surprisingly little study of using sketch-like techniques for non-equi joins. Certain natural variations can be transformed into equi-joins with some care. For example, a join condition of the form

```
WHERE R1.A <= R2.B
```

can be incorporated by modifying the data which is sketched. Here, the frequency distribution of R_1 is sketched as usual, but for R_2, we set $f(i)$ to count the total number of rows where attribute B is greater than or equal to i. The inner product of the two frequency distributions is now equal to the size of join. This approach has the disadvantage that it is slow to process the input data: each update to R_2 requires significant effort to propagate the change to the sketch. An alternate approach may be to use dyadic ranges to speed up the updates: the join can be broken into joins of attribute values whose difference is a power of two.

There has been more effort in using sketches for *spatial joins*. This refers to cases where the data is considered to represent points in a d dimensional space. A variety of queries exist here, such as (a) the spatial join of two sets of (hyper) rectangles, where two (hyper)rectangles are considered to join if they overlap; and (b) the distance join of two sets of points, where two points join if they are within distance r of each other. Das *et al.* [76] use sketches to answer these kinds of queries. They assume a discretized domain, where the underlying d dimensional frequency distribution encodes the number of points (if any) at each coordinate location. In one dimension, it is possible to count how many intervals from one set intersect intervals from the second set. The key insight is to view an interval intersection as the endpoint of one interval being present within the other interval (and vice-versa). This can then be captured using an equi-join between the endpoint distribution of one set of intervals and the range of points covered by intervals from the other set. Therefore, sketches can be applied to approximate the size of the spatial join. A little care is needed to avoid double counting, and to handle some boundary cases, such as when two intervals share a common endpoint.

This one dimensional case extends to higher dimensions in a fairly natural way. The main challenge is handling the growth in the number of boundary cases to consider. This can be reduced by either assuming that each coordinate of a (hyper)rectangle is unique, or by forcing this to be the case by tripling the range of each coordinate to encode whether an object starts, ends, or continues through that coordinate value. This approach directly allows spatial joins to be solved. Distance joins of point sets can be addressed by observing that replacing each point in one of the sets with an appropriate object of radius r and then computing the spatial join yields the desired result.

5.3.5.3 Alternate Estimation Methods and Sketches.

There has been much research into getting the best possible accuracy from sketches, based on variations on how they are updated and how the estimates are extracted from the sketch data structure.

Domain Partitioning. Dobra *et al.* propose reducing the variance of sketch estimators for join size by partitioning the domain into p pieces, and keeping (averaging) AMS sketches over each partition [88]. The resulting variance of the estimator is the sum of the products of the self-join sizes of the partitioned relations, which can be smaller than the product of the self-join sizes of the full relations divided by p. With *a priori* knowledge of the frequency distributions, optimal partitions can be chosen. However, it seems that gains of equal or greater accuracy arise from using the fast AMS sketch (based on the hashing trick) [89]. The hashing approach can be seen as a random sketch partitioning, where the partition is defined implicitly by a hash function. Since no prior knowledge of the frequency distribution is needed here, it seems generally preferable.

Skimmed Sketches. The *skimmed sketch* technique [103] observes that much of the error in join size estimation using sketches arises from collisions with high frequencies. Instead, Ganguly *et al.* propose "skimming" off the high frequency items from each relation by extracting the (approximate) heavy hitters, so each relation is broken into a "low" and a "high" relation. The join can now be broken into four pieces, each of which can be estimated from either the estimated heavy hitters, or from sketches after the

contributions of the sketches have been subtracted off. These four pieces can be thought of as (a) high-high (product of frequencies of items which are heavy hitters in both relations) (b) low-low (inner product estimation from the skimmed sketches) and (c) high-low and low-high (product of heavy hitter items with corresponding items from the other sketch). This is shown to be an improvement over the original scheme based on averaging multiple estimates together (Section 5.3.3.1). However, it is unclear whether there is a significant gain over the hashing version of AMS sketches where the hashing randomly separates the heavy hitters with high probability.

Conservative Update. The *conservative update* method can be applied on Count-Min sketches (and also on Bloom Filters with counters) when the data is presented in the cash-register model. It tries to minimize overestimation by increasing the counters by the smallest amount possible given the information available. However, in doing so it breaks the property that the summary is a linear transform of the input. Consider an update to item i in a Count-Min sketch. The update function maps i to a set of entries in the sketch. The current estimate $\hat{f}(i)$ is given by the least of these: this has to increase by at least the amount of the update u to maintain the accuracy guarantee. But if other entries are larger than $\hat{f}(i)+u$, then they do not need to be increased to ensure that the estimate is correct. So the conservative update rule is to set

$$C[j,h_j(i)] \leftarrow \max(\hat{f}(i)+u, C[j,h_j(i)])$$

for each row j. The same technique can be applied to Bloom Filters that use counters [57], and was first proposed by Estan and Varghese [94].

Least Squares Estimation. The approach of taking the minimum value as the estimate from Count-Min sketch is appealing for its simplicity. But it is also open to criticism: it does not take full account of all the information available to the estimator. Lee *et al.* studied using a least-squares method to recover estimated frequencies of a subset of items from a Count-Min sketch [214]. That is, using the fact that the sketch is a linear transform of the input, write the sketch as a multiplication between a version of the sketch matrix and a vector of the frequencies of the items of interest. To avoid generating too large a problem to solve, all items that are not of interest are modeled as a small number of extra variables which add "noise" to the sketch entries. This

linear system can be solved by applying the matrix (pseudo)inverse of the sketch matrix, and the result minimizes the difference between the sketch of the reconstructed data and the sketch of the original data. This should be no worse than the simple min-estimator, and could be much better. Experiments in [214] indicate that this approach is reduces error, but as more items are recovered, the time cost to solve the equations grows rapidly.

Several other methods have been considered for squeezing more accuracy out of simple sketch data structures. Lu *et al* use Message Passing, which also tries to find a distribution of counts which is consistent with the values recorded in the sketch of the observed data [218]. Jin *et al* empirically measure the accuracy of an instance of a Count-Min sketch [193]. They estimate the frequency of some items which are known to have zero count, say $M+1, M+2 \ldots$ etc. The average of these estimates is used as τ, the expected error, and all estimated counts are reduced by τ. Likewise, Deng and Rafiei propose changing the row estimate of $f(i)$ to the value of the entry containing i, less the average value of the other entries in the same row, and analyze the variance of the resulting estimate [83]. A similar notion was used by Thorup and Zhang within their "new" estimator for F_2, which is shown to give guaranteed accuracy [281].

Skipping and Sampling. Over truly massive data, and extremely high update speeds, even the "fast" sketches Count Min, Count Sketch and (fast) AMS can fail to scale. A natural idea is that if there is so much data, it surely can't be necessary to observe it all to capture the main shape of the frequency distribution. Instead, we can "skip over" some fraction of the input. Bhattacharyya *et al.* [20] study the idea of skipping over items for heavy hitter and self-join size queries. To determine when to sketch and when to skip, they keep track of the volume of data that has been skipped, and only skip when the net effect of the skipped data (whatever the value happens to be) on the estimation cannot be too large.

Rusu and Dobra [266] study sketching over a Bernoulli sample of the data. They analyze the accuracy, and show how much of the variance arises from the sketching, how much from the sampling, and how much from the interaction of the two. As the sampling rate decreases, the sampling has a proportionately greater impact on the overall accuracy. Their experimental study shows that while sampling does decrease the overall accuracy, estimates of

join and self-join size can still be quite good when only sketching a tenth or a hundredth of the input data. Note that none of these methods will extend to distinct value queries: as discussed in Section 2.6.2, it is known that no sampling method can guarantee to give good results for all possible inputs. Hence, applying sketching on top of sampling can be no better than the underlying sampling method.

Hardware Implementations. In addition to the various software implementations discussed so far [89, 65], there has been work on building hardware implementations of sketch methods to further increase their scalability. These can take advantage of the fact that, due to linearity, sketches can be easily parallelized, and even within a single update, there is significant parallelism across the d repetitions. Several teams have studied effective parallel implementations of the Count-Min sketch. Lai and Byrd [212] describe a performance on a SIMD architecture which can achieve high throughput with low energy usage. Thomas *et al.* [279] implement the Count-Min sketch on the Cell processor (multi-core) architecture, and analyze choices in the scheduling and load-balancing issues that arise.

Lower Bounds. All of the variations try different methods to improve the accuracy or speed of the sketching, with varied results. It is natural to ask, can any procedure asymptotically improve the accuracy, for a given amount of space? In general, the answer is no: for many of the queries studied, there are lower bounds proved which show that the space used by the sketches are essentially optimal in their dependence on ε or M. However, typically these lower bounds are proved by considering various "worst case" frequency distributions. Often the frequency distributions seen in reality are far from worst-case, and often can be well modeled by standard statistical distributions (such as Zipfian or Pareto distributions). Here, it is possible to see better space/accuracy trade-offs. Several prior works have analyzed sketches under distributional assumptions and quantified these trade-offs [45, 71].

5.4 Sketches for Distinct Value Queries

Problems relating to estimating the number of distinct items present in a sequence have been heavily studied in the last two decades. The central problem

is equivalent to finding the cardinality of an attribute in a relation. The simple SQL fragment

```
SELECT COUNT (DISTINCT A)
FROM R
```

is sufficiently difficult to approximate in small space that dozens if not hundreds of research papers have tackled the problems and variations. More generally, we are interested in approximating the results of set valued queries: queries which perform a variety of set operations (intersection, union, difference) and then ask for the cardinality of the resulting set. We will see that the key to answering such queries is to first answer the simpler COUNT DISTINCT queries.

5.4.1 Linear Space Solutions

We first present solutions which use space linear in the size of the attribute cardinality. For cash-register streams, a natural solution is to use a compact set representation such as a Bloom filter. For each item in the stream, the procedure then tests whether it is already present in the Bloom filter. If the item is not present in the filter, then it is inserted into the filter, and the current count of distinct items is increased. By the one-sided error nature of the Bloom filter, the resulting count never overestimates the true count, but may underestimate due to collisions. To ensure a small constant rate of under-counting, it is necessary to set the size of the Bloom filter to be proportional to the cardinality being estimated. Due to the compactness of the Bloom filter bit vector, this requires less space than storing the full representation of the set, but only by constant factors.

The linear counting method due to Whang *et al.* [289] takes a related approach. The method can be understood as keeping a Bloom filter with a single hash function ($k = 1$). The number of distinct items is estimated based on the fraction of bits in the filter which remain as 0. If this fraction is z, then the number of distinct items is estimated as $m \ln 1/z$ (where m is the number of bits in the filter). Again, for this to yield an accurate estimation, the m is required to be proportional to (an upper bound on) the number of distinct items. Based on some numerical analysis, this constant of proportionality is shown to be relatively low: to get low error, it is sufficient to have m be a

factor of (roughly) 10 times smaller than the true cardinality of the relation.

Both linear counting and Bloom filters can be modified to allow deletions, by using the trick of replacing bits with counters: each deletion removes the effect of a prior insertion, and the estimators are modified accordingly. However, this extension potentially blows up the space further, since single bits are replaced with, say, 32 bit integers.

Both these approaches have the limitation that some *a priori* knowledge of the cardinality being estimated is needed. That is, to use them practically, it is necessary to know how large to make their filters. If the filter size is underestimated, then the filter will saturate (be almost entirely full of 1s), and the estimation will be useless. On the other hand, if the filter is mostly empty then the estimate will be very accurate, but the unused space will be wasted. Subsequent methods do not require any prior knowledge of the cardinality, and adjust to widely varying cardinalities.

5.4.2 Flajolet-Martin Sketches

The Flajolet-Martin sketch is probably the earliest, and perhaps the best known method for approximating the distinct count in small space [98]. It is also based on a bitmap B, but items are mapped non-uniformly to entries. The size of the bitmap is chosen to be logarithmic in the largest possible cardinality being estimated, so a much weaker upper bound is needed, and typically 32 or 64 bits will suffice for one instance of the bitmap. A hash function h is associated with the bitmap, so that half the items are mapped to 1, a quarter to 2, and so on. That is,

$$\Pr[h(i) = j] = 2^{-j}$$

where the probability is taken over the random choice of the hash function. Such a hash function is easy to generate from a function that maps uniformly onto a range: given a uniform hash function h', we set $h(i)$ to be the number of trailing zeros in the binary representation of $h'(i)$ plus one. So if $h'(i) = 3$, we set $h(i) = 1$, while if $h'(i) = 24$, we set $h(i) = 4$.

The sketch is updated in the same way as a Bloom filter: each update is hashed by h, and we set $B[h(i)]$ to 1. A simple example is shown in Figure 5.8: an item i is hashed to location 4. There is already a 1 in this location, so the sketch does not change. After seeing n distinct items, the low entries in

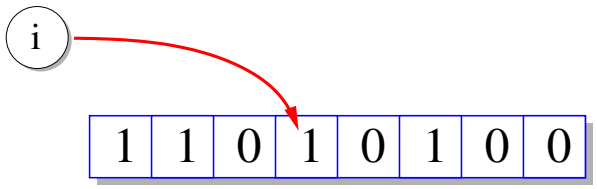


Fig. 5.8 FM Sketch Data Structure

the bitmap are expected to be set to 1, but it is unlikely that any very high entries will be set to one. More precisely, the expected number of items which are hashed to jth entry is (approximately) $n/2^j$. Locations which expect to receive more than 1 item are very likely to be set to 1, while locations which expect to receive a number of items that is less than 1 are more likely to remain as 0. The transition occurs around where $n/2^j = 1$, i.e. around the $\log_2 n$'th entry. So it is unlikely that entries above $\log n$ are set, while most entries below $\log n$ should be set to 1. Around $\log n$ it is expected that there will be a mixture of zeros and ones. A variety of different estimators are possible, such as the position of the leftmost zero in the array, or the position of the rightmost one (indexing the array from the first entry as the leftmost).

Flajolet and Martin advocated using the position of the leftmost zero [98]. Intuitively this is more robust, since it represents the compound event that all n items were not hashed there, whereas a single item can affect the position of the rightmost one. To build an estimator, they take k repetitions of the process with different hash functions, and find the mean position of the leftmost zero across these repetitions as r. The estimated value is given by $\hat{n} = 1.2928 \cdot 2^r$: here, 1.2928 is a scaling constant derived from the analysis assuming that the hash functions are perfectly random. The variance of this estimator grows with $1/\sqrt{k}$, so by taking $O(1/\varepsilon^2 \log 1/\delta)$ repetitions, the resulting estimation $\hat{n}$ is within εn of the true value n with probability at least $1-\delta$.

Alon *et al.* analyze the effect of using the two raised to the power of the position of the rightmost one as the estimator, when using hash functions with only pairwise independence [7]. Under this restricted setting, they show that the probability of overestimating by a factor of $c > 2$ or underestimating by a factor of $c' < 1/2$ is at most $1/c + 1/c'$. In other words, the method gives a constant factor approximation with constant probability using only logarithmic space. In fact, the space needed is only $O(\log \log n)$, since we

only need to record the index of the rightmost one, rather than the full bitmap. Durand and Flajolet refer to this method as "log-log counting", and analyze it further assuming fully independent hash functions. They provide an unbiased estimator based on an appropriate scaling constant [93]. Taking k repetitions has approximately twice the variance of k instances of the original Flajolet-Martin estimator, but each repetition requires much less space to store.

The downside of these approaches as described is that they are slow to process: $O(k)$ hash functions have to be evaluated for each update. Recognizing this, Flajolet and Martin proposed using "stochastic averaging", where now the items are first hashed to one of k FM sketches, which is then updated in the usual way. Here, each update requires only a constant amount of hashing, and so is much faster to update. The stochastic averaging can be viewed as an analogue of the "hashing trick" in Section 5.3. It is can also be seen as a generalization of the linear hashing described in Section 5.4.1: an FM sketch is kept in each entry of a Bloom filter instead of just a single bit.

5.4.2.1 Linear version of FM Sketch

The methods based on the Flajolet Martin sketch and its variants assume a cash-register model of the stream. But over transaction streams in the turnstile model, it is necessary to also process deletions of items. The natural solution is to replace the bits in the sketch with counters which are incremented for each inserted item that touches the entry, and decremented for each deleted item [98]. At any instant, we can extract a corresponding bitmap by setting each non-zero counter to 1, which is exactly the bitmap that would have been obtained by processing just those items which have non-zero counts. It therefore follows immediately that this linear sketch correctly processes deletions.

In the general case, there may be items with negative frequencies. It is less obvious how to interpret a COUNT DISTINCT query over such frequency distributions. However, these distributions can arise implicitly: it has been argued that it is useful to compute the number of items in two different distributions which have different frequencies. By subtracting the two frequency distributions as $f - f'$, the number of items with different frequencies corresponds to the number of items in $f - f'$ that have non-zero frequency. This measure has been dubbed "the Hamming norm" or L_0, as a limiting case of L_p^p norms [61]. Approximating this quantity requires a different solution: the

sum of frequencies of all items which hash to the same entry may be zero, even though not all of the frequencies are zero. Instead of a counter then, we can use a fingerprint of the items mapping to the entry (Section 5.2.2): with high probability, this will identify whether or not the frequency vector of items mapping to an entry is the zero vector.

5.4.3 Distinct Sampling

The idea of distinct sampling (also known as adaptive sampling) is to combine the decreasing probabilities from FM sketches with a sampling technique. Flajolet [97] attributes the invention of the technique to Wegman in the mid 1980s. A similar technique was subsequently proposed by Gibbons and Tirthapura [123], which was shown to require only limited independence hash functions. Extensions to other application domains were subsequently presented in [122].

The method is quite simple: the algorithm maintains a set of at most k items from the input (and possibly some additional information, such as their multiplicity). During the execution of the algorithm, a integer variable l records the current "level" of the sampling. Each item in the input is hashed using a function h which obeys

$$\Pr[h(i) = j] = 2^{-j}$$

i.e. the same conditions as the FM sampling variants. The item is included in the sample if the hash value is at least the current level, so that $h(i) \geq l$ (hence we may say that the level of some item is l, or talk about the items that are at some particular level). Initially, $l = 1$ and so all distinct items are sampled.

When the sample is full (i.e., it contains more than k distinct items), the level is increased by one. The sample is then pruned: all items in the sample whose hash value is less than the current value of l are rejected. Note that when l increases by one, the effective sampling rate halves, and so we expect the sample to decrease in size to approximately $k/2$. At any moment, the current number of distinct items in the whole sequence so far can be estimated as $s2^l$, where s denotes the current number of items in the sample. In the extreme case when $k = 1$, we can see this as being similar to a single instance of log-log counting method. However, because typically $k > 1$, the accuracy should be better since it is less sensitive to a single item with a very high hash level.

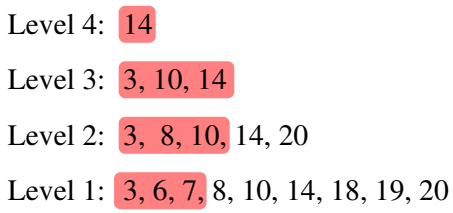


Fig. 5.9 Distinct Sampling with $k = 3$

Figure 5.9 shows a small example of distinct sampling for $k = 3$. Level 1 indicates the full set of distinct items that are present in the data, but this exceeds the capacity of the sample. At level 2, five of the nine items in the original input hash to a level greater than one. There are exactly three items that hash to level 3 or above, so the algorithm would currently be running at level $l = 3$. However, as soon as a new item arrives with $h(i) \geq 3$, the capacity of the sample would be exceeded, and the algorithm would advance to $l = 4$. Based on the information in the figure, items "3" and "10" would be dropped when this happens.

Analysis shows that the process for estimating F_0 has similar variance behavior to the preceding methods. Assuming perfect hash functions, the variance grows with $1/\sqrt{k}$ [97]. With weaker assumptions on the strength of the hash functions, Gibbons and Tirthapura prove a similar results: that setting $k = O(1/\varepsilon^2)$ is sufficient to estimate the true cardinality with relative error ε with constant probability. Taking $O(\log 1/\delta)$ parallel repetitions with different hash functions, and taking the median estimate in the usual way will reduce the failure probability to δ. The proof requires showing that the process stops at the right level, and that the number of items seen at the final level is close to the expected number.

5.4.3.1 Distinct Sampling With Deletions.

Extending these ideas to handle deletions in the stream is less straightforward than for the FM sketch methods. It is certainly possible to apply deletions to the distinct sampling sketch: assuming that cardinality information is kept about the items that are currently being sampled, when the cardinality of a sampled item goes to zero, it is considered to have been deleted from the sample. This procedure is clearly correct, in that the resulting approxima-

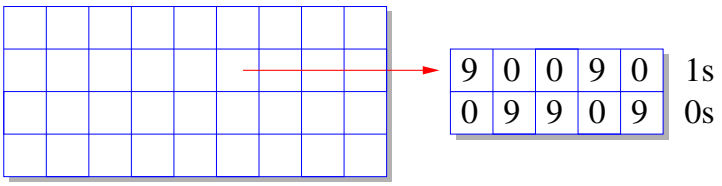


Fig. 5.10 Two-level hash sketch for distinct sampling with deletions

tions correctly reflect the current support set of the underlying data. However, in extreme cases, when many items are deleted, the number of items in the sample may become very small or even zero. In these cases, the accuracy of the approximations becomes very poor: essentially, the current sampling rate is too low for the current data size. Ideally, the algorithm should revert to a lower level with a smaller value l. But this is not possible without a rescan of the data, since simply changing the current value of l will not recover information items that were previously rejected from the sample.

Ganguly *et al.* proposed the "two-level hash" sketch to guarantee recovering a distinct sample in the presence of many deletions, by merging several of the ideas we have seen already in other sketches [102]. Each item is hashed by a function h with the same probability distribution as for FM sketches. A second hash function g maps uniformly onto the range $[k]$. The sketch tracks information about every combination of g and h values: there are $k \log n$ of these entries. For each such entry $C[h(i), g(i)]$, there are two vectors of $\log M$ counters: corresponding to each of the $\log M$ bit positions in the binary expansion of the item identifiers. The jth counter in the first vector counts the number of updates that map to the entry $C[h(i), g(i)]$ and have a 1 in the jth position of the binary representation of i. The second vector does the same for 0s in the jth position. Since this is a linear sketch, it can be updated over streams with deletions.

To recover a sample of items from the data structure, the vector of counters in each entry is analyzed. If there is only one item i with non-zero frequency that has been mapped to a particular $C[a,b]$ entry of the sketch, then the two vectors of counters allow it to be recovered: the non-zero counts in the two vectors encode exactly the binary encoding of i. If, on the other hand, more than one item is mapped to $C[a,b]$ then there will be some index j so that both vectors record a non-zero count, and so that entry is abandoned. This

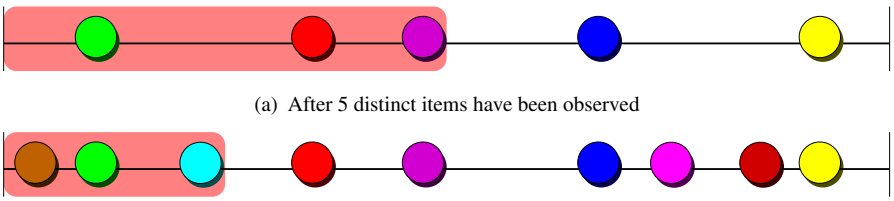
method is conceptually similar to the method described in Section 5.3.4.1 for recovering heavy hitters over strict streams. Several variations on this idea for drawing a sample from the set of distinct items have also been proposed [99, 72].

To extract a distinct sample from this distinct structure, each entry $C[a,b]$ of the sketch is examined, with the aim of recovering a single item that was mapped there. For small values of a, many items are mapped to that row, so few items will be recovered. But for a sufficiently large value of a, many items will be recovered from the row. For a given value of a, all recovered items can be returned as the sample from the sketch. Observe that a here plays a similar role to that of the level l in the distinct sampling method. Based on a, and the number of items recovered, the number of distinct items can be estimated. A partial example is shown in Figure 5.10: the figure shows the two vectors for one entry in the sketch. These vectors encode that there is a unique item that has been hashed to that entry, with multiplicity 9. It has binary representation 10010, i.e. it corresponds to the item with identifier "18".

Ganguly subsequently reduced the space required by observing that, for strict distributions, the items can be recovered from the entries more efficiently [101]. Now each entry $C[a,b]$ maintains just three counters:

$$
\begin{aligned}
T[a,b] &= \sum_{1\leq i\leq M,\ h(i)=a,\ g(i)=b} f(i) \\
U[a,b] &= \sum_{1\leq i\leq M,\ h(i)=a,\ g(i)=b} i\cdot f(i) \\
V[a,b] &= \sum_{1\leq i\leq M,\ h(i)=a,\ g(i)=b} i^2\cdot f(i)
\end{aligned}
$$

Clearly, if $T[a,b]=0$, then there are no items with non-zero counts mapped to $C[a,b]$. If there is only one unique item mapped to $C[a,b]$, then $U^2[a,b] = T[a,b]V[a,b] = i^2f^2(i)$. The converse is also true: if $U^2[a,b] = T[a,b]V[a,b]$, then only one distinct item is mapped to $C[a,b]$. Further, that item is $U[a,b]/T[a,b]$ and it has frequency $T[a,b]$. For the example in Figure 5.10, the statistics computed are $T=9, U=162$ and $V=2916$. From these, we can check that $TV = U^2 = 26244$, and that the item encoded is "18" with frequency 9.



(a) After 5 distinct items have been observed

(b) After 9 distinct items have been observed

Fig. 5.11 Illustration of the k Minimum Values procedure

5.4.4 k Minimum Values

The k minimum values technique for cash-register streams was proposed by Bar-Yossef *et al.* [17]. The technique is quite simple to state: a hash function maps items to the range $\{1 \ldots M^3\}$ (so there are unlikely to be any hash collisions), and the algorithm tracks the k smallest distinct hash values of items seen in the stream. Let h_k denote the kth smallest hash value. The number of distinct items is estimated as $k(M^3/h_k)$. Figure 5.11 illustrates the process, for $k = 3$. Figure 5.11(a) shows five distinct items mapped by the hash function onto the range $\{1 \ldots M^3\}$ (shown schematically as a number line). Of these, information about the $k = 3$ items which have the smallest hash values is stored, identified with a shaded background. As new items arrive, the $k = 3$ smallest changes: Figure 5.11(b) shows that two new items have entered the k smallest, and only information about the current k smallest values is stored.

The intuition for the KMV estimator is straightforward: the smallest hash value is expected to be around M^3/n, but the estimate based on this has high variance. Taking n random values in the range 1 to M^3, we expect about k of them to be less than $(k/n)M^3$. So if the hash values are close to uniform random values, then the result of the estimator should be close to n, the number of distinct items. In our example, the $k = 3$ smallest hash value in Figure 5.11(b) is approximately 1/4 of the way along the range, leading us to estimate that n is (roughly) 12 (in the figure, $n = 9$).

This can be made precise: the probability that the estimate is far from n corresponds to having too many or two few hash values falling in the expected range, and the probability of this event can be made small for k large enough. This is proved for pairwise independent hash functions in [17]. Making stronger assumptions about the hash functions, Beyer *et al.* [19] show

that an unbiased estimator for n is $(k-1)(M^3/h_k)$, and that this has variance proportional to n^2/k. This is gives the same asymptotic guarantees as the previous analysis, but tightens the constant factors involved considerably. Applying this unbiased estimator to Figure 5.11(b), the estimate of n is now 8.

As with distinct sampling, it is possible to process deletions: simply track the cardinality of each item selected as one of the k smallest, and remove any with zero remaining occurrences. It has the same problems: too many deletions reduce the effective k value, and in extreme cases cause the sample to become empty. Methods based on sampling via linear sketches, as in the two-level hash approach, seem to be the only method to deal with inputs which exhibit such pronounced variation in their support set.

Comparing KMV and Distinct Sampling. The KMV technique can be connected to distinct sampling: both progressively sample the data with decreasing probabilities. KMV smoothly decreases the sampling probability so that the sample always has size k, whereas distinct sampling forces the sampling rate to always be a power of two. With care, we can set up a precise correspondence. By appropriate choice of the hash functions, it is possible to arrange that the distinct sample is always a subset of the items sampled by KMV, further indicating the conceptual similarities of the methods.

The concept of hashing items and tracking information about the smallest hash values has appeared many times and has been given many different names. The idea of *min-wise hashing* (also known as *min-wise independent permutations*) is quite similar [25]. There, focus has been on designing compact families of hash functions which have the desired properties without needing to make strong independence assumptions. Estimators based on min-wise hashing typically keep k independent hash functions and take the item with least hash value in each repetition. They therefore take more time to process each update compared to tracking the k minimum values from a single hash function. Cohen and Kaplan's work on *bottom-k* sketches generalizes KMV ideas to cases when items have weights which are combined with the hash values to determine which items should be retained in the sketch [56]. Beyer *et al.* [19] have also identified connections between estimators for KMV and for priority sampling (which does not consider duplicate elim-

ination) [92].

5.4.5 Approximate Query Processing on Set Valued Queries

As remarked above, the data structures which provide us with estimates for count distinct queries can also be extended to a variety of other "distinct" queries.

5.4.5.1 Union and Sum Queries

The FM sketch variants are mainly limited to estimating the simple count-distinct queries initially discussed. However, they (in common with all the methods discussed for count distinct estimation) naturally allow the size of unions to be approximated with the same accuracy. That is, given some arbitrary collection of sets, each of which is sketched using the same hash function(s), it is possible to accurately approximate the cardinality of the union of certain sets. For FM sketches, it suffices to simply build a new FM sketch where each entry is the bitwise-or of all the corresponding entries in the sketches of the sets in the union, and apply the estimation procedure to this new sketch. This follows because the resulting sketch is exactly that which would have been obtained had the union of the sets been sketched directly.

Similar results apply for KMV and distinct sampling methods: in those cases, the procedure just takes the items sampled in the union of the sketches as the input to a new sketch (with the same hash function), and extracts the estimate from the resulting sketch. The correctness of the resulting sketch follows immediately by observing that no other items from the input could be present in the sketch of the union.

It is also possible to use these sketches to approximate various kinds of "distinct sum" queries. Here, the stream may contain multiple values of $f(i)$, and the aim is to compute the sum of the max of each value for a given i. This can be accomplished by replacing each $f(i)$ in the stream with new items $(i,1),(i,2),\ldots(i,f(i))$. The number of distinct items in the new stream gives exactly the distinct sum. However, when the $f(i)$s can be very large, it is quite time consuming to generate and sketch so many new items. Instead, various methods have been suggested to more rapidly compute the results, either via simulating the effect of adding $f(i)$ items quickly [60], or by designing "range efficient" algorithms for count distinct [250].

5.4.5.2 Distinct Prefix queries

FM sketches and Distinct samples also allow a limited kind of COUNT DISTINCT with selection queries to be answered accurately over cash-register streams. We refer to these as Distinct Prefix queries: the query is to approximate the number of distinct items seen whose identifier is less than a certain value. For FM sketches, instead of keeping a single bit in the jth sketch entry, the sketch instead records the smallest identifier of any item that has been hashed to $B[j]$. Then, given a query value q, the estimation procedure extracts a bitmap from the sketch, by setting the jth bit to 1 if $B[j] < q$, and 0 otherwise. The result is exactly the FM sketch that would have been obtained if only those items less than q had been inserted.

Likewise, for distinct sampling, rather than advancing the level whenever the sample becomes full, the sketch instead keeps samples for all levels l. At each level l, it retains the k smallest identifiers which hash to that level or above. Then, given the query q, the estimate is formed by finding the first level l where there are some items greater than q, and estimating the answer as $r2^l$, where r is the number of items at level l that are less than q. Again, it is possible to argue that we recover exactly the information that would have been seen had only the items less than q arrived. Figure 5.9 shows what would happen when applying this for a sample of size 3: the items in the shaded region at each level would be retained.

These variations are quite natural, and were originally proposed to address estimating the number of distinct items seen within a recent time window [78, 124], which can be interpreted exactly as a Distinct Prefix Query. Note that the results here are quite strong: the approximate answers are within relative error of the true answer, since it is possible to argue that there is enough information to build the synopsis of the selected data.

5.4.5.3 Predicates on Items

The Distinct Prefix can be thought of as applying a predicate to items and asking for the number of distinct items satisfying the predicate. More generally, we might want to know how many distinct items pass any given predicate at query time. Since KMV and Distinct Sampling both provide a sample of the distinct items, it is natural to apply the predicate to the sample, and build an

estimate: take the fraction of items in the sample which pass the predicate, $\hat{\rho}$, and multiply this by the estimate of the total number of distinct items $\hat{n}$. It can be argued that this a good estimator for the true answer [122].

However, the error in this estimator is proportional not to the true answer, but rather to the accuracy with which $\hat{n}$ is estimated. This should not be surprising: the same behavior follows for approximating the selectivity of a predicate via sampling without the DISTINCT keyword (see Section 2.4.3). For example, if a predicate is highly selective, then it is unlikely that many items passing the predicate happened to be placed in the sample, and so we do not have a good handle on the exact selectivity. More precisely, if the sample size is k, then the variance of the estimator of ρ, the true fraction of items, behaves like ρ/k. So to get a selectivity estimate which is within relative error of ε, the size of the samples k needs to be $O(\varepsilon^{-2}\rho^{-1})$. This technique for estimating the selectivity of predicates has been applied in a variety of situations: Frahling *et al.* use it to estimate quantities over geometric streams such as the weight of the minimum spanning tree [99].

5.4.5.4 Set Operations

Many queries are based on estimating the cardinality of the results of performing operations on a collection of sets. We have already seen that set unions can be easily answered by sketch data structures for count distinct. We now show how methods which draw a sample, such as KMV and distinct sampling, can use the samples from different sets to approximate more complex set-based queries.

Consider first estimating the size of the intersection between two sets. Given sketches that draw k samples uniformly over each relation R_A and R_B, then these can be combined to find a sample from the union. However, it is not correct to just take the union of the samples: this would not be a uniform sample if one relation were much larger than the other. Instead, the correct thing to do is to take the samples corresponding to the k distinct smallest hash values. This is a uniform sample over the union $R_A \cup R_B$.

We can now view estimating the intersection size as a special case of estimating the selectivity of a predicate: the predicate selects those items which are present in both R_A and R_B. Although there is not enough information to evaluate this predicate over arbitrary sets, there is enough to evaluate it over

the sample that has been built: just count how many items in the union sample are present in the samples of both relations. The fraction of matching items, ρ is an unbiased estimator for the true fraction, i.e.

$$\mathsf{E}[\rho] = \frac{|R_A \cap R_B|}{|R_A \cup R_B|}.$$

So multiplying the estimate for $|R_A \cup R_B|$, the size of the union, gives a good estimator for the intersection size.

The accuracy of this estimator for intersection depends primarily on the size of the union. This is because, if the intersection is very small, it is unlikely that the procedure would sample items from the intersection unless the samples of both relations are large. It is more likely that it would sample many items from the union of the relations outside the intersection. The variance of the selectivity estimation can be shown to scale with $|R_A \cap R_B|/(|R_A \cup R_B|\sqrt{k})$.

The same concept extends to arbitrary set expressions over relations $R_A, R_B, R_C \ldots$. Again, the estimation procedure takes items with the k smallest hash values from the union of all the sketches of the relations, and then evaluates the selectivity of the set expression on this sample as ρ. This can also be viewed as a more complex predicate which can be evaluated exactly on the sampled items. With this view, the variance of the estimator can be analyzed in a similar way to before. This technique is implicit in the work of Ganguly *et al.* [102], and is generalized and made explicit in the work of Beyer *et al.* [19].

The same idea works for estimating the cardinality of multiset operations such as multiset difference. Here, it is necessary to include the counts of items in the samples. Operations on the samples are applied in the natural way: union operations take the sum of counts of sampled items, while multiset differences make appropriate subtractions of counts. The fraction of matching items in the samples is found by counting those that have non-zero counts after applying the appropriate computations on the sample [19].

Certain set operations can also be evaluated using Flajolet-Martin sketch variants. This follows by using certain identities. For example, for sets A and B,

$$|A \cap B| = |A| + |B| - |A \cup B|.$$

Therefore, since Flajolet-Martin sketches can be combined to approximate

the size of the union of sets, the estimates can also yield approximations of the intersection size. However, here the (absolute) error here is proportional to $|A \cup B|/\sqrt{k}$, which is greater than the error of the estimators derived based on KMV and distinct sampling.

5.4.5.5 Comparison between sketches for Distinct Value Estimation

Analytically, all the estimators proposed for distinct value estimation have broadly the same performance: as a function of a size parameter k, their variance is proportional to $1/\sqrt{k}$. Consequently, they can all provide ε relative error using space $O(1/\varepsilon^2 \log 1/\delta)$. The exact space required is a function of the exact constants in the variance (which are understood very well for most methods), and on exactly what information needs to be retained by the sketch (bitmaps, hash values, or sampled items).

Still, to fully understand which methods are preferable for particular tasks, empirical evaluation is necessary. A recent study by Metwally *et al.* performed a thorough comparison of algorithms for the core problem of distinct count estimation [227]. They gave each method the same amount of space for a sketch, and compared the accuracy and time cost on a networking dataset. Their experiments shows that methods using bitmaps could be the most accurate—perhaps unsurprising, since it is possible to squeeze in more information into bitmaps, compared to retaining item identifiers. Indeed, the provocative conclusion of their study is that one of the earliest, methods, Linear Counting (Section 5.4.1), is preferable to the more complex methods that have come since. This is due in part to its speed and accuracy on the data in question, for which approximate cardinalities are known in advance. However, in the wider context of Approximate Query Processing, it could be argued that the extra flexibility that arises from having a sample of (hashed) items to answer broader classes of queries is more desirable.

Distinct sampling and KMV methods show an order of magnitude worse accuracy than bitmap based methods in the experiments in [227]: this is perhaps to be expected, since the (hashed) item identifiers are also at least tens of bits in size. The processing times reported for all methods are comparable, thought: approximately a million updates per second.

However, the conclusion is not the same for queries more complex than simple distinct counting. Beyer *et al.* performed experiments comparing the

KMV approach to a version of log-log counting for set operation estimation [19]. They allocated space equivalent to $k = 8192$ for the KMV estimator. Across a variety of queries, the KMV approach obtained accuracy of around 2-3% average relative error, whereas the log-log method incurred around two to three time as much error in the same space. Beyer *et al.* also observe that the choice of hash functions can empirically affect the quality of estimators, which may explain their differing behavior seen across experimental studies.

5.4.6 Lower Bounds

Ultimately, all the methods for count-distinct estimation take $\tilde{O}(1/\varepsilon^2)$ space to give ε relative accuracy (where $\tilde{O}$ notation hides logarithmic dependency on other factors like m). In fact, this dependency has been shown to be tight, in that no method can have a lower dependency on ε. Indyk and Woodruff demonstrated this by analyzing the complexity of an abstract problem called "Gap-Hamming" [176]. They showed that approximating Count-Distinct with sufficient accuracy can solve Gap-Hamming, and that Gap-Hamming has a high complexity, implying the lower bound. Subsequently, Jayram *et al.* considerably simplified the hardness proof for Gap-Hamming [188].

5.5 Other topics in sketching

In this section, we briefly survey some perspectives on sketch-based methods for Approximate Query Processing.

5.5.1 Sketches for Building other summaries

One demonstration of the flexibility of sketch-based methods is the fact that there has been considerable work on using sketches to build different types of summary. In fact, for all the other major summaries in approximate query processing—samples, histograms, and wavelets—there are methods to extract such a summary from sketch information. This can be useful when users are familiar with the semantics of such summaries, or have well-established methods for visualizing and processing such summaries.

Sketches for Sampling. The methods discussed in Sections 5.4.3 and 5.4.4 produce samples from the set of distinct items in a relation in a sketch-like manner. Further, the two-level hash structure described in Section 5.4.3.1 gives a linear sketch structure to achieve this. However, there are fewer results for using sketches to sample from the raw frequency distribution, e.g. to include item i in the sample with probability $f(i)/N$. Only recently have their been schemes to implement so-called L_p sampling, which aim to draw i with probability $f(i)/\sum_i |f(i)|^p$ [232, 197]. These methods guarantee always drawing a sample of size $O(k)$ over turnstile streams, but the sampling probabilities only approximate the desired distribution within a $(1 \pm \varepsilon)$ factor.

Sketches for Histograms. Gilbert *et al.* [128] used sketches to build histogram representations of data. The sketch needed is similar to other sketch constructions, but augmented to allow range sums to be computed rapidly. This is needed when trying to compute the impact of picking a particular bucket to be part of the histogram. Based on a dynamic-programming method for constructing the histogram, the final result guarantees an approximation to the optimal histogram under a given error metric (L_1 or L_2 error). Thaper *et al.* used sketches to find histograms of multi-dimensional data, based on a greedy search [278].

Sketches for Wavelets. There has been much interest in building sketches to recover wavelet representations. Gilbert *et al.* [129] introduce the problem. They observe that the problem can be solved exactly when the data is presented in the timeseries model (i.e. sorted and aggregated), since there are only $O(\log N)$ coefficients that "straddle" any index in the data, which can be tracked to find the largest coefficients. They also show that, in the cash-register model, any wavelet coefficient can be found via appropriate range sum computations, and proposed using sketches to estimate the largest coefficients. It was subsequently suggested that it may be more efficient to compute the wavelet transform "on the fly": since the HWT is a linear transform, it is possible to build a sketch of the coefficients, without materializing the coefficient vector explicitly [64]. The problem is then reduced to finding the heavy hitters under the F_2 measure, i.e. to find all items whose frequency exceeds ϕF_2 for some fraction ϕ. This is solved by a variant of the AMS sketch with multiple levels of hashing and grouping. More details on this technique

are provided in Section 4.6.1.

5.5.2 Other Sketch Methods

The ideas within sketches have been used to build sketches for different queries. Many common ideas are present: use of limited-independence hash functions, hierarchical decompositions of ranges and so on. Some additional ideas are also commonly used: picking random values from appropriate distributions, combining multiple sketch methods together, and use of pseudo-random number generators to minimize the space needed. We briefly survey some of these, and see how they are connected to the methods we have discussed already.

5.5.2.1 Sketches for L_p norms and Entropy.

The F_2 estimation problem is a special case of a more general class of functions over sketches, the L_p norms. For a general distribution of frequencies, the L_p norm is defined as

$$L_p = \left(\sum_{i=1}^{M} |f(i)|^p \right)^{1/p}$$

The AMS sketch therefore accurately approximates the L_2 norm. Within the streaming community, there is considerable interest in being able to provide accurate approximations of other L_p norms. As $p < 1$ approaches 0, the norm approaches the "Hamming norm", i.e. the number of non-zero entries in the vector. Computations based on such norms (e.g. clustering) vary whether the emphasis is on the magnitude or the number of differences between vectors. Estimating L_p norms for $p = 1 \pm \delta$ for small values of δ has also been instrumental in some methods for estimating entropy [162].

A general technique for $0 < p \leq 2$ is based on sketches that use *stable distributions*. Each entry in the sketch is formed as the inner product of the frequency distribution with a vector of entries each drawn randomly from a stable distribution. A stable distribution is a statistical distribution that has a stability parameter α. They have the property that sums of multiples of stable distributions are also distributed as a stable distribution—this can be viewed as a generalization of the central limit theorem, and indeed the normal distribution is stable with parameter $\alpha = 2$ [299].

By fixing the stability parameter of all distributions to be $\alpha = p$, the resulting sketch entry is distributed as a stable distribution scaled by L_p. Taking the median of all the sketch entries was shown by Indyk to be an accurate approximation of L_p [175]. To make this a small space approximation, the entries of the sketch matrix are not stored. Instead, they are generated on demand using a pseudo-random number generator so that the entry is the same every time it is accessed. Such sketches have been used in a variety of settings, such as for approximately clustering large data under L_p distance [66].

There has also been a lot of work on approximating L_p for $p > 2$. Here, strong lower bounds indicate that the size of the sketch needed is at least $\Omega(M^{1-2/p})$ [292]. Various solutions have been proposed which achieve this space cost (up to logarithmic factors) [177, 21]. The HSS (for Hierarchical Sampling from Sketches) is based on randomly selecting subsets of items via hash functions, and then extracting the heavy hitters from these subsets of items using Count Sketches. The information about the approximate frequencies of the recovered items is then combined to form an estimate of the overall L_p norm of the full frequency distribution. Owing to the lower bounds, the resulting data structure can be very large.

The ideas within the HSS sketch have also been applied to approximate the *empirical entropy* of the frequency distribution. That is, to estimate

$$H = \sum_{i=1}^{M} \frac{f(i)}{N} \log(\frac{N}{f(i)})$$

However, tighter error bounds result from using methods based on sampling and tracking information about the sampled elements [40], or by via interpolating estimates from sketches for L_p norms [162, 8].

5.5.2.2 Combinations of Sketches

We have previously discussed several examples of combining different types of sketches: take, for example, the use of fingerprints within Flajolet-Martin sketches to track count-distinct over general frequency distributions (Section 5.4.2.1). For linear sketches in particular, it is often possible to combine multiple sketches via nesting to answer more complex queries.

A particular example arises when we wish to make sketches "duplicate resilient". The "distinct heavy hitters" problem arises when the input consists

of a sequence of tuples (i, j), and the frequency of i in the multiset input $\mathscr{D}$ is now defined as $d(i) = |\{j : (i, j) \in \mathscr{D}\}|$, the number of distinct js that i occurs with. This models a variety of problems, particularly ones which occur in networking settings [282]. One solution combines the Count-Min Sketch with Flajolet-Martin sketches: instead of keeping a counter of the items mapped there, each entry of the Count-Min sketch can keep the Flajolet-Martin sketch summary of the items [59]. This sketch allows the estimation of $d(i)$ for a given i by applying the Count-Min estimation procedure to the estimates from each Flajolet-Martin sketch in the data structure. A careful analysis is needed to study the space-accuracy tradeoff of the resulting "CM-FM sketch".

Several other combinations of sketches have been studied. In general, it is not possible to form arbitrary combinations of sketches: instead, a more cautious approach is needed, with the new estimators requiring new analysis on a case-by-case basis.

5.5.2.3 Deterministic Sketches

Most of the sketches we have seen so far involve randomization. Apart from trivial sketches (for sum, count, average etc.), the accuracy guarantees hold only with some probability over the random choice of hash functions. It is natural to ask whether the more complex queries truly require randomization to approximate. It turns out that this is not always the case: there exist sketches for certain queries which do not require randomness. However, they require substantially more space and time to create than their randomized equivalents, so they may be primarily of academic interest.

The *CR-Precis* data structure as analyzed by Ganguly and Majumder [105] (first suggested by Gasieniec and Muthukrishnan [236]) is similar to the Count-Min sketch, but determines which items to count together in each entry based on residues modulo prime numbers. The accuracy guarantees for point queries are shown based on the Chinese Remainder theorem, hence the CR in the name. The sketch keeps $d = O(1/\varepsilon \log M)$ rows, but the rows are of different lengths. The jth row corresponds to the jth prime number that is greater than k (k is a parameter of the data structure), p_j. The ℓth entry in the jth row contains the sum of all frequencies of items whose identifier is ℓ

mod p_j. That is, the entries of the sketch C are given by

$$C[j,\ell] = \sum_{1 \le i \le M,\ i \bmod p_j = \ell} f(i).$$

Due to the choice of parameters of the sketch, no pair of items i and i' can collide in every row (by the Chinese Remainder theorem). In fact, any pair can collide in at most $\log_k M$ rows. Point queries can be answered as in the Count-Min case, by extracting the count of every entry where i is placed, and taking the minimum of these. Using the bound on collisions, the error in the estimate is at most $N(\log_k M)/d$. Choosing d to be large enough makes this error as small as desired. To guarantee εN error, the total space needed grows proportional to $1/\varepsilon^2$ — in comparison to the $1/\varepsilon$ growth of the (randomized) Count-Min sketch. Given this deterministic bound for point queries, deterministic results for heavy hitters, range-queries and quantiles all follow using the same reductions as in Section 5.3.4. However, for other queries, no deterministic sketch is possible: it is known that randomness is required to estimate the number of distinct items and to estimate F_2 in small space [7, 167].

6

Conclusions and Future Research Directions

In this final chapter, we conclude our survey by looking more broadly across synopses for massive data. We draw comparisons across the four classes of synopses surveyed, and summarize the strengths and weaknesses of existing approaches. We also discuss the role of approximate query processing via synopses in existing and future systems. Lastly, we look to the future and discuss the prospects for broader adoption of AQP ideas, as well as new research directions.

6.1 Comparison Across Different Methods

So far, we have primarily considered each type of synopsis in isolation. We now recap the methods, and identify their strengths and weaknesses for different aspects of AQP problems.

Sampling Pros and Cons. The chief strength of sampling-based approximation methods is that they are very flexible with regard to the queries that can be answered: they offer the generic solution of simply evaluating the query over sampled data, instead of over the full original data. Sampling also adapts very naturally to higher dimensional data (since the sample itself is es-

sentially "dimension agnostic"). This flexibility makes sample synopses ideal for general-purpose querying, when the set of queries is not fully known in advance. The accuracy of a sampling-based estimate of a query answer will be highly dependent on the query being evaluated, but such estimates usually come with query-specific error bounds, thereby permitting assessment of the accuracy. This powerful feature of sampling—as well as the ability to improve estimation accuracy on the fly by simply increasing the number of samples—ties in well with the notion of online aggregation, where the error bounds narrow in real time as more and more samples are obtained. Note that samples are the only one of the synopses discussed here that can easily be incrementally adjusted to improve estimation accuracy.

Sampling is well-suited for detecting "broad patterns" within the data, that is, features that are likely to be present in any subset of the data. It is much less suited to problems when the object of interest is a rare event (a "needle in a haystack"). Sampling does poorly on applications such as fraud or anomaly detection, where the unusual activity is unlikely to be included in any sample. For the same underlying reasons, sampling gives lower accuracy when the data is highly variable. It can do badly at estimating distinct counts, because values with a small number of duplicates are easily missed when sampling.

Sampling becomes more complex when the data is subject to many insertions and deletions, requiring more complex incremental sample maintenance techniques to ensure that the stored data remains a uniform sample of the underlying data. Lastly, a more pragmatic weakness of sampling is that interpreting the guarantees from sampling can require a degree of statistical knowledge and experience. It may therefore be inappropriate to present the raw guarantees from a sampling estimate to a statistically unsophisticated user.

Histograms Pros and Cons. A main advantage of histograms is that they are relatively simple to interpret, which makes it easier for system builders to construct and interrogate histograms. In particular, the simplicity of dividing up domains into ranges and keeping basic statistics like counts has led to the relative popularity of histograms within query optimizers and within packages for business intelligence, data analysis, and data visualization. The large amount of research dedicated to histograms has shown how they can be used

for a variety of estimation purposes—although further work is needed to identify the full range of applicability. For various classes of queries, such as point and range queries, the theory supporting these queries has been quite well-developed, so that there are strong notions of optimality. Advanced types of histogram are also sufficiently powerful to very accurately capture the structure of a data-value frequency distribution.

As with most summary structures other than samples, histograms do not adapt well to higher dimensional data. In theory, to give good error bounds their size must scale exponentially with the number of dimensions. This problem occurs in practice not only for simple bucket constructions, which track buckets that are the cross-product of one-dimensional divisions, but also for more complex hierarchical constructions.

Also, as with synopses other than samples, there has been much more work devoted to quantifying the average or maximum error over a broad class of queries than in providing error estimates that are specific to the query posed by the user. Providing such feedback is crucial when making decisions based on approximate answers to queries.

Building accurate histograms requires deciding which bucket boundaries to adopt, and computing statistics about each bucket. Naively, this requires at least two passes over the data. This approach does not adapt well when the data is dynamic and shifting. Techniques based on sketches can help in this case, but the overhead is quite high: the space for the sketches to construct the histograms is much larger than the size of the histograms themselves. It is often unclear in such scenarios whether direct use of the sketch would be simpler and more efficient.

Lastly, many histogram techniques have several parameters which have to be set a priori, such as the number of buckets, statistics to keep within each bucket, and other parameters that determine when to split or merge buckets. Setting these parameters can be off-putting, and wrong choices can lead to summaries with poor performance.

Wavelets Pros and Cons. Wavelets share with histograms the notion of storing simple statistics (primarily counts) of collections of items from the input domain. Whereas histograms excel at capturing the local structure of contiguous data values, wavelets are particularly well suited to capturing non-local structures.

The linearity of the basic Haar transform makes them more amenable than histograms to maintenance under dynamic data, although it is still non-trivial to maintain wavelet coefficients under arbitrary update patterns.

As with histograms, extending wavelets to higher dimensions brings several challenges. There are multiple definitions of how to construct a multi-dimensional wavelet, and in practice it seems that a large number of coefficients must be retained to guarantee accurate reconstruction of the data distribution. Also, as with histograms, techniques for providing query-specific error estimates are not well developed.

Sketches Pros and Cons. Sketches were first proposed in the context of high speed streams of data, and consequently the most refined implementations of sketches are very efficient to process updates to the data. The algorithms themselves are of necessity quite simple, to enable high throughput implementations; the complexity is mostly mathematical in nature, and lies in the analysis of estimation accuracy. Sketching algorithms rely on hash functions with certain mathematical properties to give their guarantees, but these hash functions are relatively simple and fast in practice, with reference implementations widely available.

The main limitation of sketching techniques—especially in contrast to the general-purpose sampling paradigm—is that each sketch tends to be focused on answering a single type of query. A reasonable range of query types can be answered via sketching, but the techniques do not seem to extend well to more complex queries which combine multiple sub-queries. Ongoing research aims to broaden the set of queries that can be answered by sketching, but general purpose techniques that are closed for a large class seem out of reach. An additional drawback is that sketches have a number of parameters that affect their accuracy and probability of failure, and these parameters may be unintuitive in some cases.

6.2 Approximate Query Processing in Systems

Several research prototype systems have placed synopses front and center, and have typically focused on sampling-based methods. These include Aqua (for Approximate QUery Answering system), developed at Bell Labs in the 1990s [3, 2]; CONTROL (Continuous Output and Navigation Technology

with Refinement On-Line), a project at UC Berkeley in the 1990s focusing on the use of sampling to provide a user with online, continually-refining answers to queries of various types [11, 260, 169]; and DBO (DataBase Online), which aimed to achieve performance similar to traditional DBMS systems while refining an approximate answer with probabilistic error bounds.

With respect to real-world systems, acceptance of synopses has been slow at best, but there have been a few successes. The notion of approximate query processing (AQP) would doubtless have been anathema to the developers of the earliest commercial database systems in the 1960's and 70's. The initial focus of such systems was transaction processing, and approximations to a customer's bank balance or a store's inventory level are unacceptable. Several key developments and trends since that time, however, have made approximate query processing an essential component of under-the-hood query processing technology, and are paving the way for wider acceptance of approximate answering of user queries.

Perhaps the earliest use of approximation technology in database management systems (DBMSs) resulted from the development of relational DBMSs, starting in the mid 1970's. The cost-based query optimizers in these systems needed to quickly evaluate the size of various intermediate query results, in order to cost competing query plans. These result sizes, called "cardinalities," could be viewed as the answers to a set of SQL COUNT queries. Importantly, cardinalities need to be determined only to a degree of accuracy sufficient for query-plan comparison. As a result, quick approximation techniques based on histograms, sampling, and distinct-count sketching rapidly found their way into query optimizers, and remain an important application of synopses. Initial estimation schemes were rather crude, and usually assumed that the frequency distribution was uniform, so that any two distinct values had the same frequency. However, over time, systems began to employ histograms and distinct-count sketches to approximate the distribution of values within an attribute of a relation more precisely, leading to better cost estimates in the presence of highly non-uniform attribute-value distributions. More recently, optimizers have begun to use sampling to reduce the I/O and CPU costs of computing optimizer statistics. Current systems supplement these techniques with summaries that are very specific to the queries being computed, such as actual measurements of the sizes of partial query results; these latter techniques are most effective when the data does not change too

rapidly. See [149] for a recent discussion of statistics and query optimizers.

With respect to approximate answers for user queries, sampling has clearly made the greatest inroads into commercial systems. Indeed, the latest version of the SQL standard, which has been implemented in virtually all current commercial relational database systems, allows specification of sampling queries. For example, the query

```
SELECT SUM SALES / 0.02
FROM transactions AS t TABLESAMPLE BERNOULLI(2)
WHERE t.product = ‘widget‘
GROUP BY t.city;
```

estimates the total sales of widgets in each city represented in the `transactions` table by means of a 2% Bernoulli sample of the rows of the table.

With the development of data mining technology in the early 1990's, enterprises became increasingly aware that their large stores of operational data were an untapped source of valuable information for strategic planning and decision support. The emphasis in this setting was often discovery of interesting patterns in large amounts of data, a problem ideally suited to AQP techniques. Sampling-based methods were commonly used for this purpose. For example, a large retailer might generate a column of pseudo-random numbers, sort the data on this column, and then run a query of interest on the first 1% of the rows of the sorted table to get a quick, rough estimate of the exact query answer obtained by processing all of the records. Similarly, as large scientific datasets were increasingly stored on DBMSs, the need for quick pattern discovery further motivated interest in approximate query processing, especially sampling. Though flexible and fast, sampling procedures could not always handle certain types of queries well, such as COUNT DISTINCT queries and highly selective join queries. For this reason, much research was devoted to alternative schemes that take a complete pass through the data, but have small memory requirements.

One objection to bringing synopses fully into commercial data management systems is that this would require a major re-architecting of the existing system. However, this is no barrier for new systems designed to handle new data scenarios. One particular example that has arisen since the start of the 21st century is in Data Stream Management Systems (DSMS). These systems

are designed to handle data which arrives online as a stream, such as network data or financial transactions. Such data can arrive in large quantities and at high speed, which motivates the adoption of approximate methods to answer complex aggregate queries, in addition to exact methods for simpler or highly selective queries.

Systems such as Gigascope (a network diagnosis and debugging tool used at AT&T [74]) and StreamInsight (from Microsoft [228]) allow User Defined Aggregate Functions (UDAFs) and User Defined Operators (UDOPs) to be called by queries. UDAFs and UDOPs are specified in the form of source code, which is then compiled into queries. In this context, there has been much effort to build libraries of implementations of synopses, such as sketch and sampling methods [67]. These systems demonstrate that there is a concrete need for approximate query processing in situations when exact query processing is not scalable, and approximate answers are acceptable.

6.3 Challenges and Future Directions for Synopses

Given that, outside of sampling, historical applications of synopses have been either behind the scenes or in niche/prototype systems, it is reasonable to ask what are the prospects for broader usage of the synopses described in this survey. Can we expect to see greater use of approximate answers in information management, or will they remain a minority interest to the majority goal of providing exact answers? We conclude by examining some of the obstacles to adoption of synopses for AQP. These obstacles can also be viewed as a spur to innovation, both in educating information-system designers and users, and in overcoming the limitations of current AQP technology.

6.3.1 Understanding Approximation

A key challenge is how to present results to the end user. Users pose a query knowing that there exists an exact answer; they may struggle to understand an answer which may be presented as a confidence interval. Somehow, the transition from an "exact" answer (over data which almost certainly contains minor errors and omissions) to an "approximate" one introduces more uncertainty than the typical user is currently able to accept. Part of the challenge here is in education and presentation: how to educate users to accept uncertainty in their query results, and how to present these uncertainties as palat-

ably and intuitively as possible. The stochastic-simulation community has recently discussed issues related to presentation of uncertain results [290]. The idea is to describe uncertainty at a level that matches a user's sophistication, without being misleading at any point; such research may also be helpful in the AQP setting. Similarly, research in human-computer interaction and collaborative data analysis could be useful in communicating uncertainty in a visual manner, as well as making users more comfortable with approximate query answers.

In some sense, users are already comfortable with approximations; they just fail to completely realize this. For example, opinion polls are widely used, though there is always a statistical margin of error (usually described either with mentioning this error at all or using rough language such as "correct to within an accuracy of $\pm$ three percentage points.") Weather forecasts are in their nature approximate, and driving directions, which apparently promise the shortest route, more commonly compute merely a "shortish" route, to make the computation more scalable. Search engines that say "there are about 10 millions answers" are approximating wildly. In all of these cases, people can live with the uncertainty; the challenge is to make them aware of what they are already doing.

Besides merely making people comfortable with uncertain answers, it is further necessary to educate users as to the potential benefits of using synopses. A key point to communicate is that in many common circumstances, an approximate answer can be highly accurate. For a query that touches a reasonably large number of tuples—say, 100,000 rows or more—a sample of 1,000 tuples will be highly accurate while representing a 1% fraction of the data. Explaining to users that they can have an accurate answer in 1% of the time using the same hardware should be an easy argument to make. For example, Figure 6.1 gives an example that has been effective in explaining the potential effectiveness of AQP; the figure has been adapted from an article on database sampling written for business users of DB2 [146]. Visual comparison of the two pie charts (and their corresponding computation times) shows how the approximate result still manages to effectively convey the "big picture" at a fraction of the computational cost.

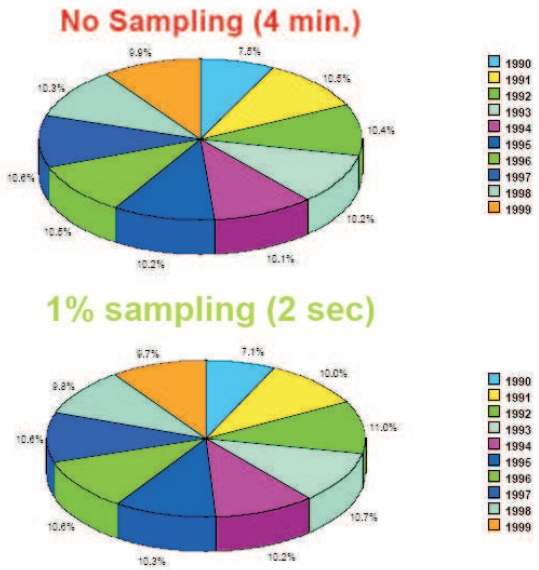


Fig. 6.1 Exact and approximate answer to sum-of-sales query, grouped by year

6.3.2 Parallelism

The alternative to approximation is to continue building bigger systems. Popular architecture paradigms are trending towards increased parallelism and distributed processing (as per the MapReduce/Hadoop model) which still fixes exact solutions as the default goal. Still, it has long been observed that data volumes are growing at faster rates than computing power [291], and this trend continues. This suggests that approximation is increasingly a necessary tool to adopt in order to facilitate interactive querying. There has been some initial work on combining approximate processing techniques with MapReduce architectures [58, 249], and this seems to be a promising direction for future research.

Moreover, as mentioned in the introduction, the advent of Cloud computing and green computing will likely result in users become more sensitive to the dollar and energy costs of executing a given query. (Given the rising cost of energy, these latter costs might often serve as a proxy for dollar costs.) In this setting, synopses provide an increasingly important tool for letting a user obtain useful results at a lower cost. Thus an ongoing technical challenge is

to extend synopsis techniques to work in a parallel and distributed setting; some work along these lines has already been described, and new methods are being actively developed; see, for example, Cormode et al. [73].

6.3.3 Incorporation into Commercial Systems

Objections to using synopses include the fact that full integration into existing data management systems may require significant re-architecting, possibly negating the investment of many hundreds if not thousands of person-years of effort to build the current systems. In recent years, however, there has been a revolution in DBMS architecture, driven by cloud computing [1], multicore processors [262], and scientific data applications [277]. There is a unique opportunity, therefore, to build AQP techniques into such systems from the outset. This comprehensive integration could focus not only on delivery of approximate answers to end users, but also incorporation of synopses into the processing engine. Hence, though it may be difficult to build new AQP techniques into existing commercial query optimizers, such synopses might be useful for new query optimization tasks such as load balancing in MapReduce systems. As discussed previously, newer applications such as network monitoring, sensor systems—i.e., settings in which stringent CPU and memory constraints are prevalent—are relatively accepting to the use of synopses. Finally, another path for incorporating AQP techniques into emerging systems is through the development of analytical libraries, such as in the ongoing MADlib project [221].

6.3.4 Uncertain Data

This article has focused on approximate answers to queries that are executed over exact, deterministic data and hence have an exact answer (though perhaps unknown and expensive to compute). In recent years, however, there has been a surge of activity related to query processing over uncertain data [75, 77]. This uptick in interest corresponds to an increasing awareness that most data is not, in fact, exact—uncertainty arises in numerous ways, including information extraction from text, data integration, noisy sensor readings, and deliberate anonymization to preserve privacy. These developments have two important implications for AQP.

First, acknowledging that one's data is uncertain leads naturally to a gen-

eral willingness to accept an quick approximate query answers. I.e., it does not make much sense to spend enormous effort computing an exact answer if the data itself does not justify such high precision. There has been increasing, fruitful interaction between the tradition stochastic analysis, stochastic optimization, and statistics communities on one hand, and the database community on the other. This convergence has been accelerated by the embrace of advanced statistical methods by the search community. As a consequence, it is plausible that the general database user community will feel increasingly comfortable with non-exact data processing and AQP in particular.

Second, a new technical challenge arises. What should synopses look like in the context of a probabilistic database? What is the interplay between data uncertainty and query approximation? There has been almost no research on this topic to date.

6.3.5 Other Technical Challenges

As can be seen from our survey, many technical challenges remain. Perhaps the chief of these, at least for the non-sampling based synopses, is to accurately represent high dimensional distributions. A promising future direction is to describe the broad dependencies via high-level statistical models, such as Bayesian and Markov networks, and then describe low-dimensional correlations via the synopses we have described.

Another challenge is to extend synopses to other kinds of data. For example, researchers have been exploring concise graph-based synopses for XML data [216, 253, 254]. This work has also inspired a purely graph-based approach to approximating relational data [274].

Besides improving on the synopses themselves, an important challenge is to find higher-level methods of working with synopses. The goal here is to allow complex query plans to be translated to operate entirely in the AQP domain rather than on the exact data.

A related problem is to gain a better theoretical understanding of the applicability, accuracy, and performance of synopses. For example, can one prove that a given class of synopses is complete and consistent (in some appropriate sense) for a given class of queries? Can the information-theoretic perspective in [205] be extended to other synopses and query classes? It is also possible that approximation theory in its full force [53] might provide

additional theoretical insights.

Acknowledgements. The work of Minos Garofalakis was partially supported by the European Commission under FP7-FET Open (Future and Emerging Technologies) ICT-2009.8.0 grant no. 255957 (LIFT).

References

[1] A. Abouzeid, K. Bajda-Pawlikowski, D. J. Abadi, A. Rasin, and A. Silberschatz. HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads. *PVLDB*, 2(1):922–933, 2009.

[2] S. Acharya, P. B. Gibbons, and V. Poosala. Aqua: a fast decision support system using approximate query answers. In *International Conference on Very Large Data Bases*, 1999.

[3] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua approximate query answering system. In *ACM SIGMOD International Conference on Management of Data*, 1999.

[4] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 275–286, New York, NY, USA, 1999. ACM.

[5] C. C. Aggarwal. On biased reservoir sampling in the presence of stream evolution. In *VLDB*, pages 607–618, 2006.

[6] N. Alon, P. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *ACM Principles of Database Systems*, 1999.

[7] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *ACM Symposium on Theory of Computing*, 1996.

[8] A. Andoni, R. Krauthgamer, and K. Onak. Streaming algorithms from precision sampling. *CoRR*, abs/1011.1263, 2010.

[9] G. Antoshenkov. Random sampling from pseudo-ranked B+ trees. In *VLDB '92: Proceedings of the 18th International Conference on Very Large Data Bases*, pages 375–382, 1992.

[10] P. M. Aoki. Algorithms for index-assisted selectivity estimation. In *Proc. ICDE*, page 258, 1999.

[11] R. Avnur, J. M. Hellerstein, B. Lo, C. Olston, B. Raman, V. Raman, T. Roth, and K. Wylie. CONTROL: Continuous Output and Navigation Technology with Refinement On-Line. In *SIGMOD Conference*, pages 567–569, 1998.

[12] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *ACM Principles of Database Systems*, 2002.

[13] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 539–550, New York, NY, USA, 2003. ACM.

[14] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *SODA*, pages 633–634, 2002.

[15] W. Baek and T. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *Proc. PLDI*, pages 198–209, 2010.

[16] L. Baltrunas, A. Mazeika, and M. H. Böhlen. Multi-dimensional histograms with tight bounds for the error. In *Proc. IDEAS*, pages 105–112, 2006.

[17] Z. Bar-Yossef, T. Jayram, R. Kumar, D. Sivakumar, and L. Trevisian. Counting distinct elements in a data stream. In *Proceedings of RANDOM 2002*, 2002.

[18] R. Bellman. On the approximation of curves by line segments using dynamic programming. *Commun. ACM*, 4(6):284, 1961.

[19] K. S. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *ACM SIGMOD International Conference on Management of Data*, 2007.

[20] S. Bhattacharrya, A. Madeira, S. Muthukrishnan, and T. Ye. How to scalably skip past streams. In *Scalable Stream Processing Systems (SSPS) Workshop with ICDE 2007*, 2007.

[21] L. Bhuvanagiri, S. Ganguly, D. Kesh, and C. Saha. Simpler algorithm for estimating frequency moments of data streams. In *ACM-SIAM Symposium on Discrete Algorithms*, 2006.

[22] P. Billingsley. *Probability and Measure*. Wiley, third edition, 1999.

[23] B. Blohsfeld, D. Korus, and B. Seeger. A comparison of selectivity estimators for range queries on metric attributes. In *Proc. SIGMOD*, pages 239–250, 1999.

[24] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[25] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *ACM Symposium on Theory of Computing*, 1998.

[26] A. Z. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4), 2003.

[27] P. G. Brown and P. J. Haas. Techniques for warehousing of sample data. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 6, Washington, DC, USA, 2006. IEEE Computer Society.

[28] B. Bru. The estimates of Laplace. an example: research concerning the population of a large empire, 1785-1812. In *Journal de la Société de statistique de Paris*, volume 129, pages 6–45, 1988.

[29] N. Bruno and S. Chaudhuri. Exploiting statistics on query expressions for optimization. In *Proc. SIGMOD*, pages 263–274, 2002.

[30] N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: A multidimensional workload-aware histogram. In *Proc. SIGMOD*, pages 211–222, 2001.

[31] T. Bu, J. Cao, A. Chen, and P. P. C. Lee. A fast and compact method for unveiling significant patterns in high speed networks. In *IEEE INFOCOMM*, 2007.

[32] F. Buccafurri, F. Furfaro, G. Lax, and D. Saccà. Binary-tree histograms with tree indices. In *Proc. DEXA*, pages 861–870, 2002.

[33] F. Buccafurri, F. Furfaro, and D. Saccà. Estimating range queries using aggregate data with integrity constraints: A probabilistic approach. In *Proc. ICDT*, pages 390–404, 2001.

[34] F. Buccafurri, F. Furfaro, D. Saccà, and C. Sirangelo. A quad-tree based multiresolution approach for two-dimensional summary data. In *Proc. SSDBM*, pages 127–137, 2003.

[35] F. Buccafurri and G. Lax. Fast range query estimation by n-level tree histograms. *Data Knowl. Eng.*, 51(2):257–275, 2004.

[36] F. Buccafurri and G. Lax. Reducing data stream sliding windows by cyclic tree-like histograms. In *PKDD*, pages 75–86, 2004.

[37] F. Buccafurri, G. Lax, D. Saccà, L. Pontieri, and D. Rosaci. Enhancing histograms by tree-like bucket indices. *VLDB J.*, 17(5):1041–1061, 2008.

[38] C. Buragohain, N. Shrivastava, and S. Suri. Space efficient streaming algorithms for the maximum error histogram. In *Proc. ICDE*, pages 1026–1035, 2007.

[39] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[40] A. Chakrabarti, G. Cormode, and A. McGregor. A near-optimal algorithm for computing the entropy of a stream. In *ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[41] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate Query Processing Using Wavelets. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 111–122, Cairo, Egypt, Sept. 2000.

[42] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *The VLDB Journal*, 10(2-3):199–223, Sept. 2001. (Best of VLDB'2000 Special Issue).

[43] D. Chamberlin. *A Complete Guide to DB2 Universal Database*. Morgan Kaufmann, 1998.

[44] M. Charikar, S. Chaudhuri, R. Motwani, and V. R. Narasayya. Towards estimation error guarantees for distinct values. In *Proc. Nineteenth ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems*, pages 268–279. ACM, 2000.

[45] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2002.

[46] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theoret. Comput. Sci.*, 312(1):3–15, 2004.

[47] S. Chaudhuri, G. Das, and V. R. Narasayya. A robust, optimization-based approach for approximate answering of aggregate queries. In *SIGMOD Conference*, pages 295–306, 2001.

[48] S. Chaudhuri, G. Das, and V. R. Narasayya. Optimized stratified sampling for approximate query processing. *ACM Trans. Database Syst.*, 32(2):9, 2007.

[49] S. Chaudhuri, G. Das, and U. Srivastava. Effective use of block-level sampling in statistics estimation. In *SIGMOD Conference*, pages 287–298, 2004.

[50] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. *SIGMOD Rec.*, 28(2):263–274, 1999.

[51] S. Chaudhuri, R. Motwani, and V. R. Narasayya. Random sampling for histogram construction: How much is enough? In *Proc. SIGMOD*, pages 436–447, 1998.

[52] K. K. Chen. Influence query optimization with optimization profiles and statistical views in DB2 9: Optimal query performance in DB2 9 for Linux, UNIX, and Windows. Available at `www.ibm.com/developerworks/db2/library/techarticle/dm-0612chen`, 2006.

[53] W. Cheney and W. Light. *A Course in Approximation Theory*. Brooks/Cole, Pacific Grove, CA, 2000.

[54] E. Cohen, G. Cormode, and N. G. Duffield. Structure-aware sampling on data streams. In *SIGMETRICS*, pages 197–208, 2011.

[55] E. Cohen, N. Grossaug, and H. Kaplan. Processing top-k queries from samples. In *Proc. CoNext*, page 7, 2006.

[56] E. Cohen and H. Kaplan. Summarizing data using bottom-k sketches. In *ACM Conference on Principles of Distributed Computing (PODC)*, 2007.

[57] S. Cohen and Y. Matias. Spectral bloom filters. In *ACM SIGMOD International Conference on Management of Data*, 2003.

[58] T. Condie, N. Conway, P. Alvaro, J. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce online. In *NSDI*, 2010.

[59] J. Considine, M. Hadjieleftheriou, F. Li, J. W. Byers, and G. Kollios. Robust approximate aggregation in sensor data management systems. *ACM Transactions on Database Systems*, 34(1), Apr. 2009.

[60] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *IEEE International Conference on Data Engineering*, 2004.

[61] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using Hamming norms. In *International Conference on Very Large Data Bases*, 2002.

[62] G. Cormode, A. Deligiannakis, M. N. Garofalakis, and A. McGregor. Probabilistic histograms for probabilistic data. *PVLDB*, 2(1):526–537, 2009.

[63] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *International Conference on Very Large Data Bases*, 2005.

[64] G. Cormode, M. Garofalakis, and D. Sacharidis. Fast Approximate Wavelet Tracking on Streams. In *Proceedings of the 10th International Conference on Extending Database Technology (EDBT'2006)*, Munich, Germany, Mar. 2006.

[65] G. Cormode and M. Hadjieleftheriou. Finding frequent items in data streams. In *International Conference on Very Large Data Bases*, 2008.

[66] G. Cormode, P. Indyk, N. Koudas, and S. Muthukrishnan. Fast mining of tabular data via approximate distance computations. In *IEEE International Conference on Data Engineering*, 2002.

[67] G. Cormode, F. Korn, S. Muthukrishnan, T. Johnson, O. Spatscheck, and D. Srivastava. Holistic UDAFs at streaming speeds. In *ACM SIGMOD International Conference on Management of Data*, pages 35–46, 2004.

[68] G. Cormode, F. Korn, S. M. Muthukrishnan, and D. Srivastava. Space- and time-efficient deterministic algorithms for biased quantiles over data streams. In *Proc. PODS*, pages 263–272, 2006.

[69] G. Cormode and S. Muthukrishnan. What's new: Finding significant differences in network data streams. In *Proceedings of IEEE Infocom*, 2004.

[70] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[71] G. Cormode and S. Muthukrishnan. Summarizing and mining skewed data streams. In *SIAM Conference on Data Mining*, 2005.

[72] G. Cormode, S. Muthukrishnan, and I. Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *International Conference on Very Large Data Bases*, 2005.

[73] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Optimal sampling from distributed streams. In *Proc. PODS*, pages 77–86, 2010.

[74] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *ACM SIGMOD International Conference on Management of Data*, 2003.

[75] N. N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009.

[76] A. Das, J. Gehrke, and M. Riedewald. Approximation techniques for spatial data. In *ACM SIGMOD International Conference on Management of Data*, 2004.

[77] A. Das Sarma, O. Benjelloun, A. Y. Halevy, S. U. Nabar, and J. Widom. Representing uncertain data: models, properties, and algorithms. *VLDB J.*, 18(5):989–1019, 2009.

[78] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[79] I. Daubechies. *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1992.

[80] A. Deligiannakis, M. Garofalakis, and N. Roussopoulos. An Approximation Scheme for Probabilistic Wavelet Synopses. In *Proceedings of the International Conference on Scientific and Statistical Database Management*, Santa Barbara, California, June 2005.

[81] A. Deligiannakis, M. Garofalakis, and N. Roussopoulos. Extended Wavelets for Multiple Measures. *ACM Transactions on Database Systems*, 32(2), June 2007.

[82] A. Deligiannakis and N. Roussopoulos. Extended Wavelets for Multiple Measures. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, San Diego, California, June 2003.

[83] F. Deng and D. Rafiei. New estimation algorithms for streaming data: Count-min can do more. `http://www.cs.ualberta.ca/~fandeng/paper/cmm.pdf`, 2007.

[84] A. Deshpande, M. N. Garofalakis, and R. Rastogi. Independence is good: Dependency-based histogram synopses for high-dimensional data. In *Proc. SIGMOD*, pages 199–210, 2001.

[85] R. A. DeVore. Nonlinear Approximation. *Acta Numerica*, 7:51–150, 1998.

[86] L. Devroye and G. Lugosi. *Combinatorial Methods in Density Estimation*. Springer, 2001.

[87] A. Dobra. Histograms revisited: when are histograms the best approximation method for aggregates over joins? In *Proc. PODS*, pages 228–237, 2005.

[88] A. Dobra, M. Garofalakis, J. E. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *ACM SIGMOD International Conference on Management of Data*, 2002.

[89] A. Dobra and F. Rusu. Statistical analysis of sketch estimators. *ACM Transactions on Database Systems*, 33(3), 2008.

[90] D. Donjerkovic, Y. E. Ioannidis, and R. Ramakrishnan. Dynamic histograms: Capturing evolving data sets. In *Proc. ICDE*, page 86, 2000.

[91] D. Donjerkovic and R. Ramakrishnan. Probabilistic optimization of top N queries. In *Proc. VLDB*, pages 411–422, 1999.

[92] N. Duffield, C. Lund, and M. Thorup. Estimating flow distributions from sampled flow statistics. In *ACM SIGCOMM*, 2003.

[93] M. Durand and P. Flajolet. Loglog counting of large cardinalities. In *European Symposium on Algorithms (ESA)*, 2003.

[94] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *ACM SIGCOMM*, 2002.

[95] C. T. Fan, M. E. Muller, and I. Rezucha. Development of sampling plans by using sequential (item by item) selection techniques and digital computers. *J. of the American Statistical Association*, pages 387–402, 1962.

[96] G. Fishman. *Monte Carlo: Concepts, Algorithms, and Applications*. Springer, 1996.

[97] P. Flajolet. On adaptive sampling. *Computing*, 43(4), 1990.

[98] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for database applications. *Journal of Computer and System Sciences*, 31:182–209, 1985.

[99] G. Frahling, P. Indyk, and C. Sohler. Sampling in dynamic data streams and applications. In *Symposium on Computational Geometry*, June 2005.

[100] D. Fuchs, Z. He, and B. S. Lee. Compressed histograms with arbitrary bucket layouts for selectivity estimation. *Inf. Sci.*, 177(3):680–702, 2007.

[101] S. Ganguly. Counting distinct items over update streams. *Theoretical Computer Science*, 378(3):211–222, 2007.

[102] S. Ganguly, M. Garofalakis, and R. Rastogi. Processing set expressions over continuous update streams. In *ACM SIGMOD International Conference on Management of Data*, 2003.

[103] S. Ganguly, M. Garofalakis, and R. Rastogi. Processing data-stream join aggregates using skimmed sketches. In *International Conference on Extending Database Technology*, 2004.

[104] S. Ganguly, P. B. Gibbons, Y. Matias, and A. Silberschatz. Bifocal sampling for skew-resistant join size estimation. *SIGMOD Rec.*, 25(2):271–281, 1996.

[105] S. Ganguly and A. Majumder. CR-precis: A deterministic summary structure for update data streams. In *ESCAPE*, 2007.

[106] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: You only get one look. In *ACM SIGMOD International Conference on Management of Data*, 2002.

[107] M. Garofalakis, J. Gehrke, and R. Rastogi, editors. *Data Stream Management: Processing High-Speed Data Streams*. Springer, 2011.

[108] M. Garofalakis and P. B. Gibbons. Approximate Query Processing: Taming the Terabytes. Tutorial in *27th Intl. Conf. on Very Large Data Bases*, Roma, Italy, Sept. 2001.

[109] M. Garofalakis and P. B. Gibbons. Wavelet Synopses with Error Guarantees. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 476–487, Madison, Wisconsin, June 2002.

[110] M. Garofalakis and P. B. Gibbons. Probabilistic Wavelet Synopses. *ACM Transactions on Database Systems*, 29(1), Mar. 2004. (SIGMOD/PODS'2002 Special Issue).

[111] M. Garofalakis and A. Kumar. Deterministic Wavelet Thresholding for Maximum-Error Metrics. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Paris, France, June 2004.

[112] M. Garofalakis and A. Kumar. Wavelet Synopses for General Error Metrics. *ACM Transactions on Database Systems*, 30(4), Dec. 2005. (SIGMOD/PODS'2004 Special Issue).

[113] R. Gemulla. *Sampling Algorithms for Evolving Datasets*. PhD thesis, Technische Universität Dresden, 2009. Available at `http://nbn-resolving.de/urn:nbn:de:bsz:14-ds-1224861856184-11644`.

[114] R. Gemulla and W. Lehner. Deferred maintenance of disk-based random samples. In *Proc. Ninth Intl. Conf. Extending Database Technology*, Lecture Notes in Computer Science, pages 423–441. Springer, 2006.

[115] R. Gemulla and W. Lehner. Sampling time-based sliding windows in bounded space. In *SIGMOD Conference*, pages 379–392, 2008.

[116] R. Gemulla, W. Lehner, and P. J. Haas. A dip in the reservoir: Maintaining sample synopses of evolving datasets. In *VLDB*, pages 595–606, 2006.

[117] R. Gemulla, W. Lehner, and P. J. Haas. Maintaining Bernoulli samples over evolving multisets. In *Proc. PODS*, pages 93–102, 2007.

[118] R. Gemulla, W. Lehner, and P. J. Haas. Maintaining bounded-size sample synopses of evolving datasets. *VLDB J.*, 17(2):173–202, 2008.

[119] J. E. Gentle. *Random Number Generation and Monte Carlo Methods*. Springer, second edition, 2003.

[120] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. In *Proc. SIGMOD*, pages 461–472, 2001.

[121] C. Giannella and B. Sayrafi. An information theoretic histogram for single dimensional selectivity estimation. In *Proc. ACM Conf. Appl. Computing*, pages 676–677, 2005.

[122] P. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *International Conference on Very Large Data Bases*, 2001.

[123] P. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2001.

[124] P. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2002.

[125] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 331–342, New York, NY, USA, 1998. ACM.

[126] P. B. Gibbons, Y. Matias, and V. Poosala. Aqua project white paper. Technical report, Bell Laboratories, Murray Hill, NJ, 1997.

[127] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. *ACM Trans. Database Syst.*, 27(3):261–298, 2002.

[128] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *ACM Symposium on Theory of Computing*, 2002.

[129] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *International Conference on Very Large Data Bases*, 2001.

[130] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *International Conference on Very Large Data Bases*, 2002.

[131] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. One-pass wavelet decomposition of data streams. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):541–554, May 2003.

[132] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *ACM SIGMOD International Conference on Management of Data*, 2001.

[133] S. Guha. A note on wavelet optimization. (Manuscript available from: `http://www.cis.upenn.edu/~ sudipto/note.html`.), Sept. 2004.

[134] S. Guha. On the space-time of optimal, approximate and streaming algorithms for synopsis construction problems. *VLDB J.*, 17(6):1509–1535, 2008.

[135] S. Guha and B. Harb. Wavelet synopsis for data streams: minimizing non-euclidean error. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Chicago, Illinois, Aug. 2005.

[136] S. Guha and B. Harb. Approximation Algorithms for Wavelet Transform Coding of Data Streams. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, Miami, Florida, Jan. 2006.

[137] S. Guha, P. Indyk, S. Muthukrishnan, and M. J. Strauss. Histogramming Data Streams with Fast Per-Item Processing. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*, Malaga, Spain, July 2002.

[138] S. Guha, C. Kim, and K. Shim. XWAVE: Optimal and Approximate Extended Wavelets for Streaming Data. In *Proceedings of the 30th International Conference on Very Large Data Bases*, Toronto, Canada, Sept. 2004.

[139] S. Guha, N. Koudas, and K. Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Trans. Database Syst.*, 31(1):396–438, 2006.

[140] S. Guha, N. Koudas, and D. Srivastava. Fast algorithms for hierarchical range histogram construction. In *Proc. PODS*, pages 180–187, 2002.

[141] S. Guha and K. Shim. A note on linear time algorithms for maximum error histograms. *IEEE Trans. Knowl. Data Eng.*, 19(7):993–997, 2007.

[142] S. Guha, K. Shim, and J. Woo. REHIST: Relative error histogram construction algorithms. In *Proc. VLDB*, pages 300–311, 2004.

[143] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. In *Proc. SIGMOD*, pages 463–474, 2000.

[144] A. P. Gurajada and J. Srivastava. Equidepth partitioning of a data set based on finding its medians. In *Proc. Appl. Computing*, pages 92–101, 1991.

[145] P. J. Haas. Large-sample and deterministic confidence intervals for online aggregation. In *SSDBM*, pages 51–63, 1997.

[146] P. J. Haas. The need for speed: speeding up DB2 UDB using sampling. *IDUG Solutions J.*, 10(2):32–34, 2003.

[147] P. J. Haas and J. M. Hellerstein. Join algorithms for online aggregation. *IBM Research Report RJ 10126*, 1998.

[148] P. J. Haas and J. M. Hellerstein. Ripple joins for online aggregation. In *SIGMOD Conference*, pages 287–298, 1999.

[149] P. J. Haas, I. F. Ilyas, G. M. Lohman, and V. Markl. Discovering and exploiting statistical properties for query optimization in relational databases: A survey. *Statistical Analysis and Data Mining*, 1(4):223–250, 2009.

[150] P. J. Haas and C. König. A bi-level bernoulli scheme for database sampling. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 275–286, 2004.

[151] P. J. Haas, Y. Liu, and L. Stokes. An estimator of the number of species from quadrat sampling. *Biometrics*, 62:135–141, 2006.

[152] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *VLDB*, pages 311–322, 1995.

[153] P. J. Haas, J. F. Naughton, S. Seshadri, and A. N. Swami. Fixed-precision estimation of join selectivity. In *Proc. PODS*, pages 190–201, 1993.

[154] P. J. Haas, J. F. Naughton, S. Seshadri, and A. N. Swami. Selectivity and cost estimation for joins based on random sampling. *J. Comput. Syst. Sci.*, 52(3):550–569, 1996.

[155] P. J. Haas, J. F. Naughton, and A. N. Swami. On the relative cost of sampling for join selectivity estimation. In *PODS*, pages 14–24, 1994.

[156] P. J. Haas and L. Stokes. Estimating the number of classes in a finite population. *J. Amer. Statist. Assoc.*, 93(444):1475–1487, 1998.

[157] P. J. Haas and A. N. Swami. Sequential sampling procedures for query size estimation. In *SIGMOD '92: Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, pages 341–350, New York, NY, USA, 1992. ACM.

[158] T. Hagerup and C. Rüb. A guided tour of chernoff bounds. *Inf. Process. Lett.*, 33(6):305–308, 1990.

[159] P. Hall and C. Heyde. *Martingale Limit Theory and Its Application*. Academic Press, 1980.

[160] M. Hansen. Some history and reminiscences on survey sampling. In *Statistical Science*, volume 2, pages 180–190, 1987.

[161] B. Harb. *Algorithms for Linear and Nonlinear Approximation of Large Data*. PhD thesis, University of Pennsylvania, 2007.

[162] N. J. A. Harvey, J. Nelson, and K. Onak. Sketching and streaming entropy via approximation theory. In *FOCS*, 2008.

[163] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

[164] Z. He, B. S. Lee, and X. S. Wang. Proactive and reactive multi-dimensional histogram maintenance for selectivity estimation. *Journal of Systems and Software*, 81(3):414–430, 2008.

[165] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD Conference*, pages 171–182, 1997.

[166] M. Henzinger. Algorithmic challenges in search engines. *Internet Mathematics*, 1(1):115–126, 2003.

[167] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. Technical Report SRC 1998-011, DEC Systems Research Centre, 1998.

[168] J. Hershberger, N. Shrivastava, S. Suri, and C. D. Tóth. Adaptive spatial partitioning for multidimensional data streams. In *Proc. ISAAC*, pages 522–533, 2004.

[169] C. Hidber. Online association rule mining. In *Proc. SIGMOD*, pages 145–156, 1999.

[170] W. Hoeffding. Probability inequalities for sums of bounded random variables. *JASA*, 58(301):1330, 1963.

[171] D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47:663–695, 1952.

[172] W.-C. Hou, G. Özsoyoglu, and B. K. Taneja. Statistical estimators for relational algebra expressions. In *Proc. Seventh ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems*, pages 276–287. ACM, 1988.

[173] W.-C. Hou, G. Özsoyoglu, and B. K. Taneja. Processing aggregate relational queries with hard time constraints. In *SIGMOD Conference*, pages 68–77, 1989.

[174] IDC. The diverse and exploding digital universe. IDC White Paper, March 2008.

[175] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *IEEE Conference on Foundations of Computer Science*, 2000.

[176] P. Indyk and D. Woodruff. Tight lower bounds for the distinct elements problem. In *IEEE Conference on Foundations of Computer Science*, 2003.

[177] P. Indyk and D. P. Woodruff. Optimal approximations of the frequency moments of data streams. In *ACM Symposium on Theory of Computing*, 2005.

[178] Y. E. Ioannidis. Universality of serial histograms. In *Proc. VLDB*, pages 256–267, 1993.

[179] Y. E. Ioannidis. Approximations in database systems. In *ICDT*, pages 16–30, 2003.

[180] Y. E. Ioannidis. The history of histograms (abridged). In *Proc. VLDB*, pages 19–30, 2003.

[181] Y. E. Ioannidis and S. Christodoulakis. On the propagation of errors in the size of join results. In *Proc. SIGMOD*, pages 268–277, 1991.

[182] Y. E. Ioannidis and S. Christodoulakis. Optimal histograms for limiting worst-case error propagation in the size of join results. *ACM Trans. Database Syst.*, 18(4), 1993.

[183] Y. E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *Proc. SIGMOD*, pages 233–244, 1995.

[184] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *Proc. VLDB*, pages 174–185, 1999.

[185] H. V. Jagadish, H. Jin, B. C. Ooi, and K.-L. Tan. Global optimization of histograms. In *Proc. SIGMOD*, pages 223–234, 2001.

[186] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proc. VLDB*, pages 275–286, 1998.

[187] B. Jawerth and W. Sweldens. An Overview of Wavelet Based Multiresolution Analyses. *SIAM Review*, 36(3):377–412, 1994.

[188] T. S. Jayram, R. Kumar, and D. Sivakumar. The one-way communication complexity of gap hamming distance. `http://www.madalgo.au.dk/img/SumSchoo2007_Lecture_20slides/Bibliography/p14_Jayram_07_Manusc_ghd.pdf`, 2007.

[189] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable approximate query processing with the DBO engine. In *ACM SIGMOD International Conference on Management of Data*, 2007.

[190] C. Jermaine, A. Dobra, S. Arumugam, S. Joshi, and A. Pol. The sort-merge-shrink join. *ACM Trans. Database Syst.*, 31(4):1382–1416, 2006.

[191] C. Jermaine, A. Dobra, A. Pol, and S. Joshi. Online estimation for subset-based sql queries. In *VLDB*, pages 745–756, 2005.

[192] C. Jermaine, A. Pol, and S. Arumugam. Online maintenance of very large random samples. In *SIGMOD Conference*, pages 299–310, 2004.

[193] C. Jin, W. Qian, C. Sha, J. X. Yu, and A. Zhou. Dynamically maintaining frequent items over a data stream. In *CIKM*, 2003.

[194] R. Jin, L. Glimcher, C. Jermaine, and G. Agrawal. New sampling-based estimators for OLAP queries. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 18, Washington, DC, USA, 2006. IEEE Computer Society.

[195] W. Johnson and J. Lindenstrauss. Extensions of Lipshitz mapping into Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[196] S. Joshi and C. Jermaine. Sampling-based estimators for subset-based queries. *VLDB J.*, accepted for publication, 2008.

[197] H. Jowhari, M. Saglam, and G. Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *ACM Principles of Database Systems*, 2011.

[198] C.-C. Kanne and G. Moerkotte. Histograms reloaded: The merits of bucket diversity. In *Proc. SIGMOD*, pages 663–674, 2010.

[199] P. Karras. Optimality and scalability in lattice histogram construction. *PVLDB*, 2(1):670–681, 2009.

[200] P. Karras and N. Mamoulis. Hierarchical Synopses with Optimal Error Guarantees. *ACM Transactions on Database Systems*, 33(3), Aug. 2008.

[201] P. Karras and N. Mamoulis. Lattice histograms: a resilient synopsis structure. In *Proc. ICDE*, pages 247–256, 2008.

[202] P. Karras and N. Manoulis. One-Pass Wavelet Synopses for Maximum-Error Metrics. In *Proceedings of the 31st International Conference on Very Large Data Bases*, Trondheim, Norway, Sept. 2005.

[203] P. Karras and N. Manoulis. The Haar$^+$ Tree: A Refined Synopsis Data Structure. In *Proceedings of the 23rd International Conference on Data Engineering*, Istanbul, Turkey, Apr. 2007.

[204] P. Karras, D. Sacharidis, and N. Manoulis. Exploiting Duality in Summarization with Deterministic Guarantees. In *Proceedings of the Thirteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Jose, California, Aug. 2007.

[205] R. Kaushik, J. F. Naughton, R. Ramakrishnan, and V. T. Chakaravarthy. Synopses for query optimization: A space-complexity perspective. *ACM Trans. Database Syst.*, 30(4):1102–1127, 2005.

[206] R. Kaushik and D. Suciu. Consistent histograms in the presence of distinct value counts. *PVLDB*, 2(1):850–861, 2009.

[207] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proc. FOCS*, pages 482–491, 2003.

[208] S. Khanna, S. Muthukrishnan, and S. Skiena. Efficient array partitioning. In *Proc. ICALP*, pages 616–626, 1997.

[209] A. C. König and G. Weikum. Combining histograms and parametric curve fitting for feedback-driven query result-size estimation. In *Proc. VLDB*, pages 423–434, 1999.

[210] F. Korn, T. Johnson, and H. V. Jagadish. Range selectivity estimation for continuous attributes. In *Proc. SSDBM*, pages 244–253, 1999.

[211] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[212] Y.-K. Lai and G. T. Byrd. High-throughput sketch update on a low-power stream processor. In *Proceedings of the ACM/IEEE symposium on Architecture for networking and communications systems*, 2006.

[213] M. R. Leadbetter, G. Lindgren, and H. Rootzen. *Extremes and Related Properties of Random Sequences and Processes: Springer Series in Statistics*. Springer, 1983.

[214] G. M. Lee, H. Liu, Y. Yoon, and Y. Zhang. Improving sketch reconstruction accuracy using linear least squares method. In *Internet Measurement Conference (IMC)*, 2005.

[215] L. Lim, M. Wang, and J. S. Vitter. SASH: A self-adaptive histogram set for dynamically changing workloads. In *Proc. VLDB*, pages 369–380, 2003.

[216] L. Lim, M. Wang, and J. S. Vitter. CXHist : An on-line classification-based histogram for XML string selectivity estimation. In *Proc. VLDB*, pages 1187–1198, 2005.

[217] X. Lin and Q. Zhang. Error minimization for approximate computation of range aggregate. In *Proc. DASFAA*, pages 165–172, 2003.

[218] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani. Counter braids: a novel counter architecture for per-flow measurement. In *SIGMETRICS*, 2008.

[219] G. Luo, C. J. Ellmann, P. J. Haas, and J. F. Naughton. A scalable hash ripple join algorithm. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 252–262, New York, NY, USA, 2002. ACM.

[220] J. Luo, X. Zhou, Y. Zhang, H. T. Shen, and J. Li. Selectivity estimation by batch-query based histogram and parametric method. In *Proc. 18th Australasian Database Conf. (ADC2007)*, pages 93–102, 2007.

[221] MADlib library for scalable analytics. `http:\\madlib.net`.

[222] S. Mallat. *A Wavelet Tour of Signal Processing (Second Edition)*. Academic Press, 1999.

[223] Y. Matias and D. Urieli. Optimal Workload-based Weighted Wavelet Synopses. In *Proceedings of the 10th International Conference on Database Theory (ICDT'2005)*, Edinburgh, Scotland, Jan. 2005.

[224] Y. Matias and D. Urieli. Optimal Workload-based Weighted Wavelet Synopses. *Theoretical Computer Science*, 371(3):227–246, 2007.

[225] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-Based Histograms for Selectivity Estimation. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 448–459, Seattle, Washington, June 1998.

[226] Y. Matias, J. S. Vitter, and M. Wang. Dynamic Maintenance of Wavelet-Based Histograms. In *Proceedings of the 26th International Conference on Very Large Data Bases*, Cairo, Egypt, Sept. 2000.

[227] A. Metwally, D. Agrawal, and A. E. Abbadi. Why go logarithmic if we can go linear?: Towards effective distinct counting of search traffic. In *International Conference on Extending Database Technology*, 2008.

[228] Microsoft. Microsoft StreamInsight. `http://msdn.microsoft.com/en-us/library/ee362541.aspx`, 2008.

[229] J. Misra and D. Gries. Finding repeated elements. *Science of Computer Programming*, 2:143–152, 1982.

[230] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. CUP, 2005.

[231] G. Moerkotte, T. Neumann, and G. Steidl. Preventing bad plans by bounding the impact of cardinality estimation errors. *PVLDB*, 2(1):982–993, 2009.

[232] M. Monemizadeh and D. P. Woodruff. 1-pass relative-error l_p-sampling with applications. In *ACM-SIAM Symposium on Discrete Algorithms*, 2010.

[233] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[234] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.

[235] M. Muralikrishna and D. J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *Proc. SIGMOD*, pages 28–36, 1988.

[236] S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers, 2005.

[237] S. Muthukrishnan. Subquadratic Algorithms for Workload-Aware Haar Wavelet Synopses. In *Proceedings of the Intl. Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, Hyderabad, India, Dec. 2005.

[238] S. Muthukrishnan, V. Poosala, and T. Suel. On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. In *Proc. ICDT*, pages 236–256, 1999.

[239] S. Muthukrishnan and M. Strauss. Maintenance of multidimensional histograms. In *Proc. Foundations Software Tech. and Theoret. Comput. Sci. (FSTTCS)*, volume 2914 of *Lecture Notes in Computer Science*, pages 352–362. Springer, 2003.

[240] S. Muthukrishnan, M. Strauss, and X. Zheng. Workload-optimal histograms on streams. In *Proc. ESA*, pages 734–745, 2005.

[241] J. Neyman. On the two different aspects of the representative method: The method of stratified sampling and the method of purposive selection. *Journal of the Royal Statistical Society*, 97:558–625, 1934.

[242] A. O'Hagan and J. J. Forster. *Bayesian Inference*. Volume 2B of *Kendall's Advanced Theory of Statistics*. Arnold, second edition, 2004.

[243] F. Olken. Random sampling from databases. Technical Report LBL-32883, Lawrence Berekeley National Laboratory, 1993.

[244] F. Olken and D. Rotem. Simple random sampling from relational databases. In *VLDB*, pages 160–169, 1986.

[245] F. Olken and D. Rotem. Random sampling from B+ trees. In *VLDB*, pages 269–277, 1989.

[246] F. Olken and D. Rotem. Maintenance of materialized views of sampling queries. In *ICDE*, pages 632–641, 1992.

[247] F. Olken, D. Rotem, and P. Xu. Random sampling from hash files. *SIGMOD Rec.*, 19(2):375–386, 1990.

[248] C. Pang, Q. Zhang, D. Hansen, and A. Maeder. Unrestricted Wavelet Synopses under Maximum Error Bound. In *Proceedings of the 12th International Conference on Extending Database Technology (EDBT'2009)*, Saint Petersburg, Russia, Mar. 2009.

[249] N. Pansare, V. Borkar, C. Jermaine, and T. Condie. Online aggregation for large MapReduce jobs. *PVLDB*, 5, 2011. To appear.

[250] A. Pavan and S. Tirthapura. Range-efficient counting of distinct elements in a massive data stream. *SIAM Journal on Computing*, 37(2):359–379, 2007.

[251] H. T. A. Pham and K. C. Sevcik. Structure choices for two-dimensional histogram construction. In *Proc. CASCON*, pages 13–27, 2004.

[252] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proc. SIGMOD*, pages 256–276, 1984.

[253] N. Polyzotis and M. N. Garofalakis. XCluster synopses for structured XML content. In *Proc. ICDE*, page 63, 2006.

[254] N. Polyzotis and M. N. Garofalakis. XSKETCH synopses for XML data graphs. *ACM Trans. Database Syst.*, 31(3):1014–1063, 2006.

[255] V. Poosala and V. Ganti. Fast approximate answers to aggregate queries on a data cube. In *Proc. SSDBM*, pages 24–33, 1999.

[256] V. Poosala, V. Ganti, and Y. E. Ioannidis. Approximate query answering using histograms. *IEEE Data Eng. Bull.*, 22(4):5–14, 1999.

[257] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proc. VLDB*, pages 486–495, 1997.

[258] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proc. SIGMOD*, pages 294–305, 1996.

[259] L. Qiao, D. Agrawal, and A. E. Abbadi. RHist: adaptive summarization over continuous data streams. In *Proc. CIKM*, pages 469–476, 2002.

[260] V. Raman and J. M. Hellerstein. Potter's Wheel: An interactive data cleaning system. In *Proc. VLDB*, pages 381–390, 2001.

[261] V. Raman, B. Raman, and J. M. Hellerstein. Online dynamic reordering. *VLDB J.*, 9(3):247–260, 2000.

[262] V. Raman, G. Swart, L. Qiao, F. Reiss, V. Dialani, D. Kossmann, I. Narang, and R. Sidle. Constant-time query processing. In *Proc. ICDE*, pages 60–69, 2008.

[263] R. L. Read, D. S. Fussell, and A. Silberschatz. Computing bounds on aggregate operations over uncertain sets using histograms. In *Proc. Post-ILPS '94 Workshop on Uncertainty in Databases and Deductive Systems*, pages 107–117, 1994.

[264] F. Reiss, M. N. Garofalakis, and J. M. Hellerstein. Compact histograms for hierarchical identifiers. In *Proc. VLDB*, pages 870–881, 2006.

[265] J. Rissanen. Stochastic complexity and modeling. *Ann. Statist.*, 14(3):1080–1100, 1986.

[266] F. Rusu and A. Dobra. Sketches for size of join estimation. *ACM Transactions on Database Systems*, 33(3), 2008.

[267] D. Sacharidis, A. Deligiannakis, and T. Sellis. Hierarchically-Compressed Wavelet Synopses. *The VLDB Journal*, 18(1):203–231, Jan. 2009.

[268] C.-E. Sarndal, B. Swennson, and J. Wretman. *Model-Assisted Survey Sampling*. Springer, second edition, 1992.

[269] R. T. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. A. Dinda, M.-Y. Kao, and G. Memik. Reversible sketches: enabling monitoring and analysis over high-speed data streams. *IEEE Transactions on Networks*, 15(5), 2007.

[270] D. W. Scott. *Multivariate Density Estimation: Theory Practice, and Visualization*. Wiley, 1992.

[271] S. Seshadri. *Probabilistic methods in query processing*. PhD thesis, University of Wisconsin at Madison, Madison, WI, USA, 1992.

[272] J. Shao and D. Tu. *The Jackknife and Bootstrap*. Springer Series in Statistics, 1995.

[273] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[274] J. Spiegel and N. Polyzotis. TuG synopses for approximate query answering. *ACM Trans. Database Syst.*, 34(1), 2009.

[275] U. Srivastava, P. J. Haas, V. Markl, and N. Megiddo. ISOMER: Consistent histogram construction using query feedback. In *Proc. ICDE*, 2006.

[276] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. *Wavelets for Computer Graphics – Theory and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, 1996.

[277] M. Stonebraker, J. Becla, D. J. DeWitt, K.-T. Lim, D. Maier, O. Ratzesberger, and S. B. Zdonik. Requirements for science data bases and scidb. In *Proc. CIDR*, 2009.

[278] N. Thaper, P. Indyk, S. Guha, and N. Koudas. Dynamic multidimensional histograms. In *ACM SIGMOD International Conference on Management of Data*, 2002.

[279] D. Thomas, R. Bordawekar, C. C. Aggarwal, and P. S. Yu. On efficient query processing of stream counts on the cell processor. In *IEEE International Conference on Data Engineering*, 2009.

[280] M. Thorup. Even strongly universal hashing is pretty fast. In *ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[281] M. Thorup and Y. Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[282] S. Venkataraman, D. X. Song, P. B. Gibbons, and A. Blum. New streaming algorithms for fast detection of superspreaders. In *Network and Distributed System Security Symposium NDSS*, 2005.

[283] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

[284] J. S. Vitter and M. Wang. Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, Philadelphia, Pennsylvania, May 1999.

[285] M. P. Wand and M. C. Jones. *Kernel Smoothing*. Chapman and Hall, 1995.

[286] H. Wang and K. C. Sevcik. Utilizing histogram information. In *Proc. CASCON*, page 16, 2001.

[287] H. Wang and K. C. Sevcik. A multi-dimensional histogram for selectivity estimation and fast approximate query answering. In *Proc. CASCON*, pages 328–342, 2003.

[288] H. Wang and K. C. Sevcik. Histograms based on the minimum description length principle. *VLDB J.*, 17(3), 2008.

[289] K. Y. Whang, B. T. Vander-Zanden, and H. M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems*, 15(2):208, 1990.

[290] J. R. Wieland and B. L. Nelson. How simulation languages should report results: a modest proposal. In *Proc. Winter Simul. Conf.*, pages 709–715, 2009.

[291] R. Winter and K. Auerbach. The big time: 1998 winter VLDB survey. *Database Programming and Design*, Aug. 1998.

[292] D. Woodruff. Optimal space lower bounds for all frequency moments. In *ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[293] M. Wu and C. Jermaine. A bayesian method for guessing the extreme values in a data set. In *VLDB*, pages 471–482, 2007.

[294] K. Yi, F. Li, M. Hadjieleftheriou, G. Kollios, and D. Srivastava. Randomized synopses for query assurance on data streams. In *IEEE International Conference on Data Engineering*, 2008.

[295] Q. Zhang and X. Lin. On linear-spline based histograms. In *Proc. WAIM*, pages 354–366, 2002.

[296] Q. Zhang and W. Wang. A fast algorithm for approximate quantiles in high speed data streams. In *Proc. SSDBM*, page 29, 2007.

[297] X. Zhang, M. L. King, and R. J. Hyndman. Bandwidth selection for multivariate kernel density estimation using MCMC. In *Econometric Society 2004 Australasian Meetings*, 2004.

[298] Q. Zhu and P.-Å. Larson. Building regression cost models for multidatabase systems. In *Proc. PDIS*, pages 220–231, 1996.

[299] V. M. Zolotarev. *One Dimensional Stable Distributions*, volume 65 of *Translations of Mathematical Monographs*. American Mathematical Society, 1983.